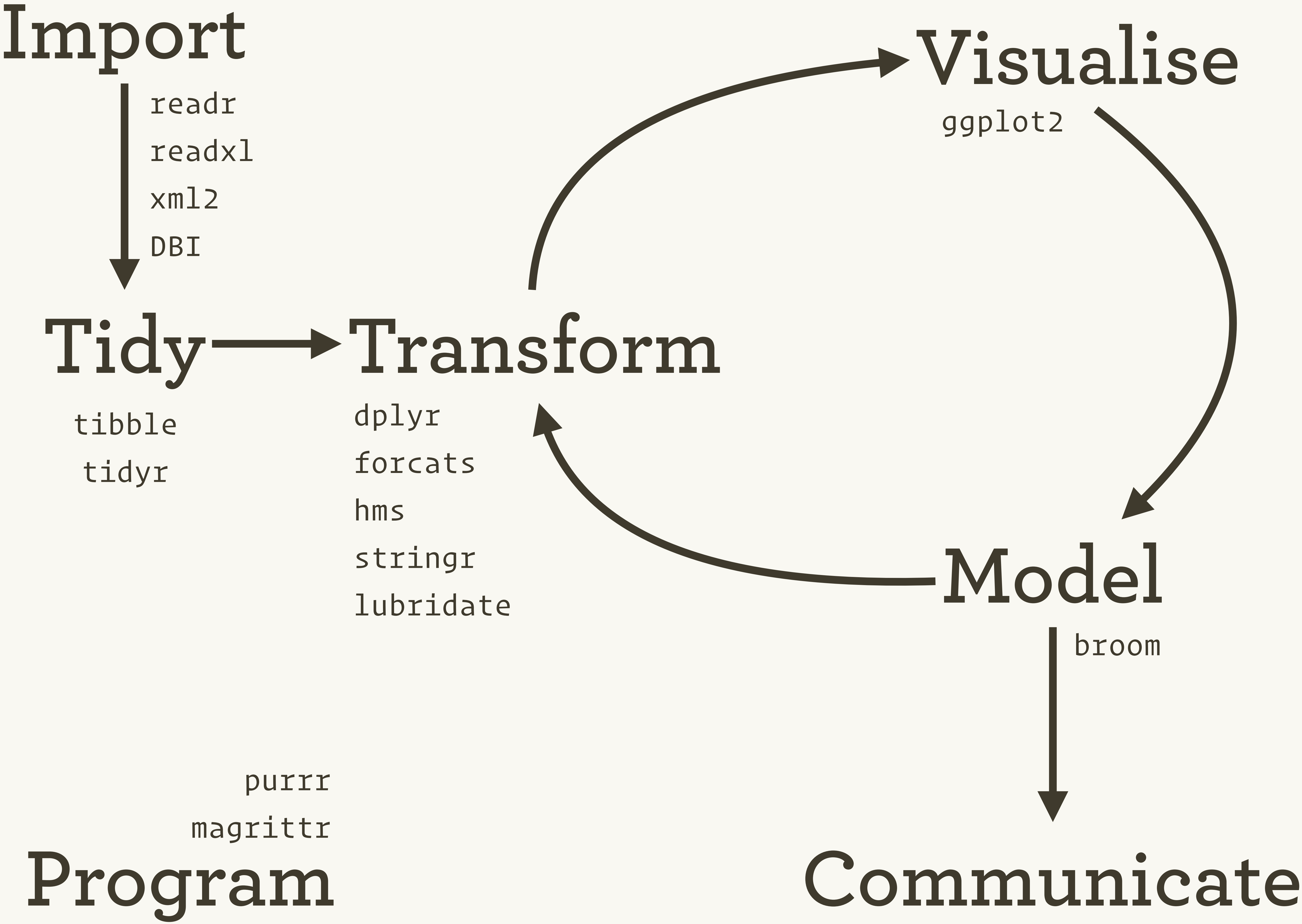
Data science with **R**

January 2017

Hadley Wickham
@hadleywickham
Chief Scientist, **RStudio**

My goal is to make a **pit of success**

http://blog.codinghorror.com/falling-into-the-pit-of-success/



Import

readr

readxl

xml2

DBI

Tidy

tibble

tidyr

Transform

dplyr

forcats

hms

stringr

lubridate

Visualise

ggplot2

Model

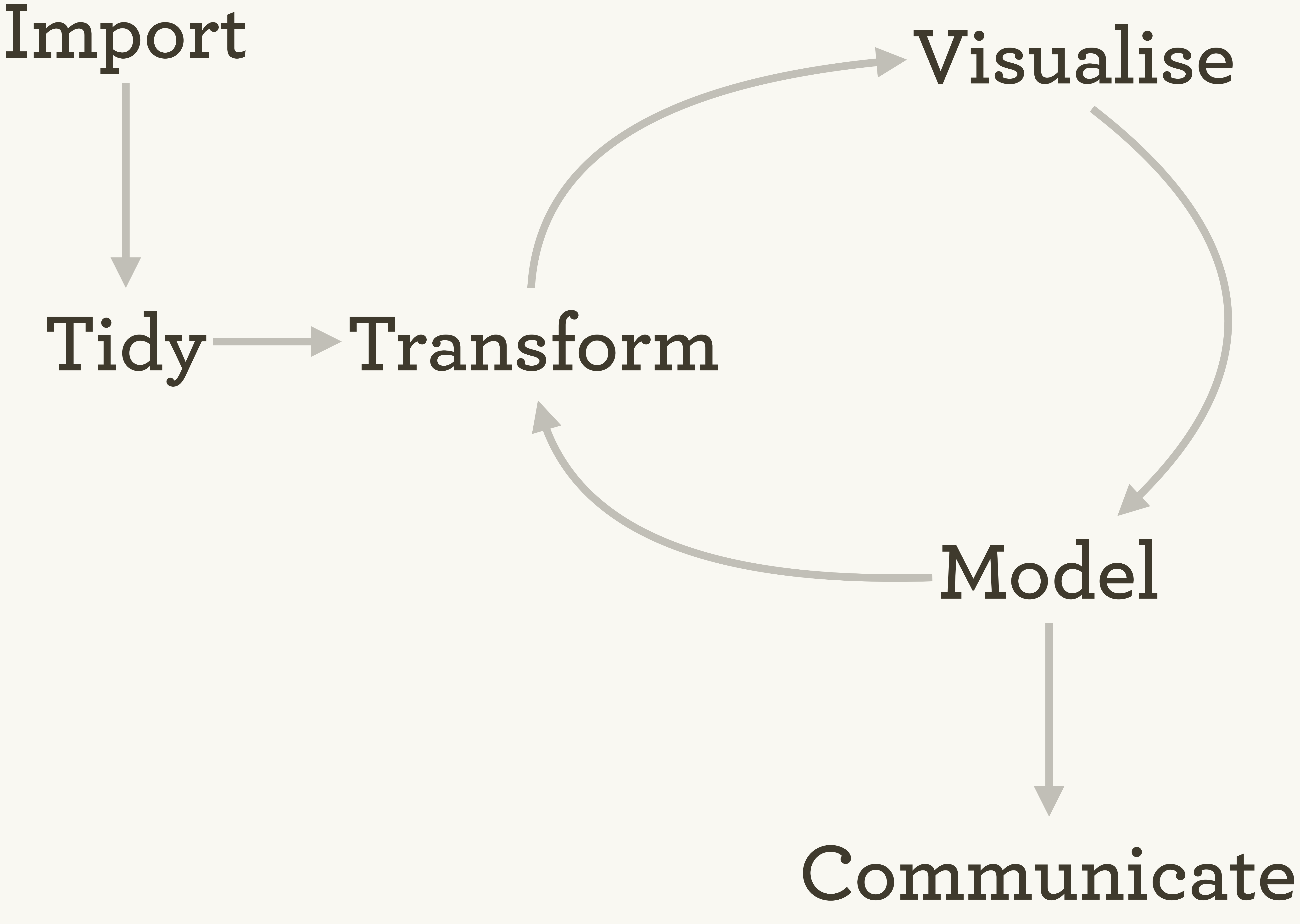
broom

Communicate

purrr

magrittr

Program



Import

Visualise

Tidy

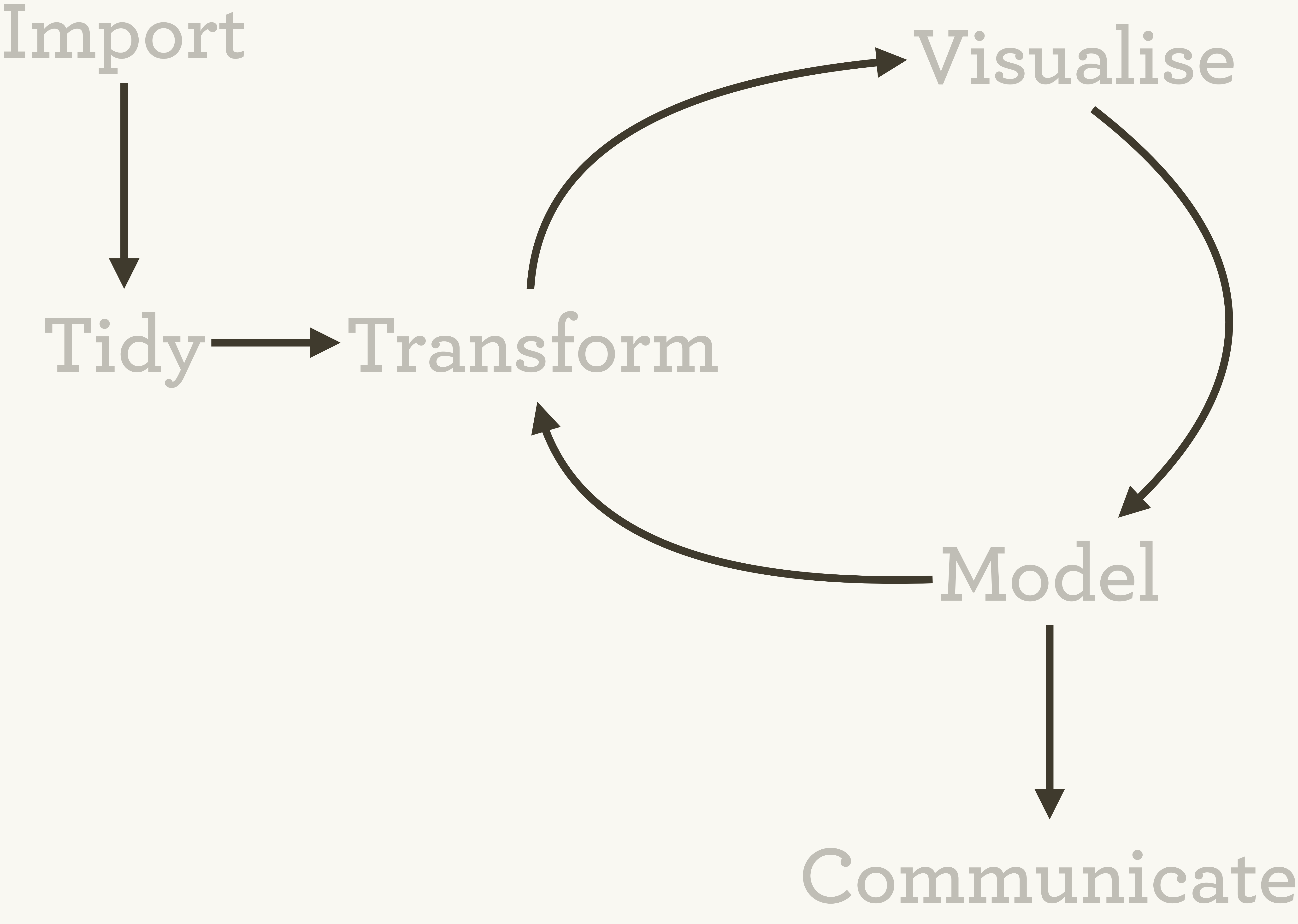
Transform

Model

Communicate

No matter how complex and polished the individual operations are, it is often the quality of the glue that most directly determines the power of the system.

— *Hal Abelson*



Import

Tidy

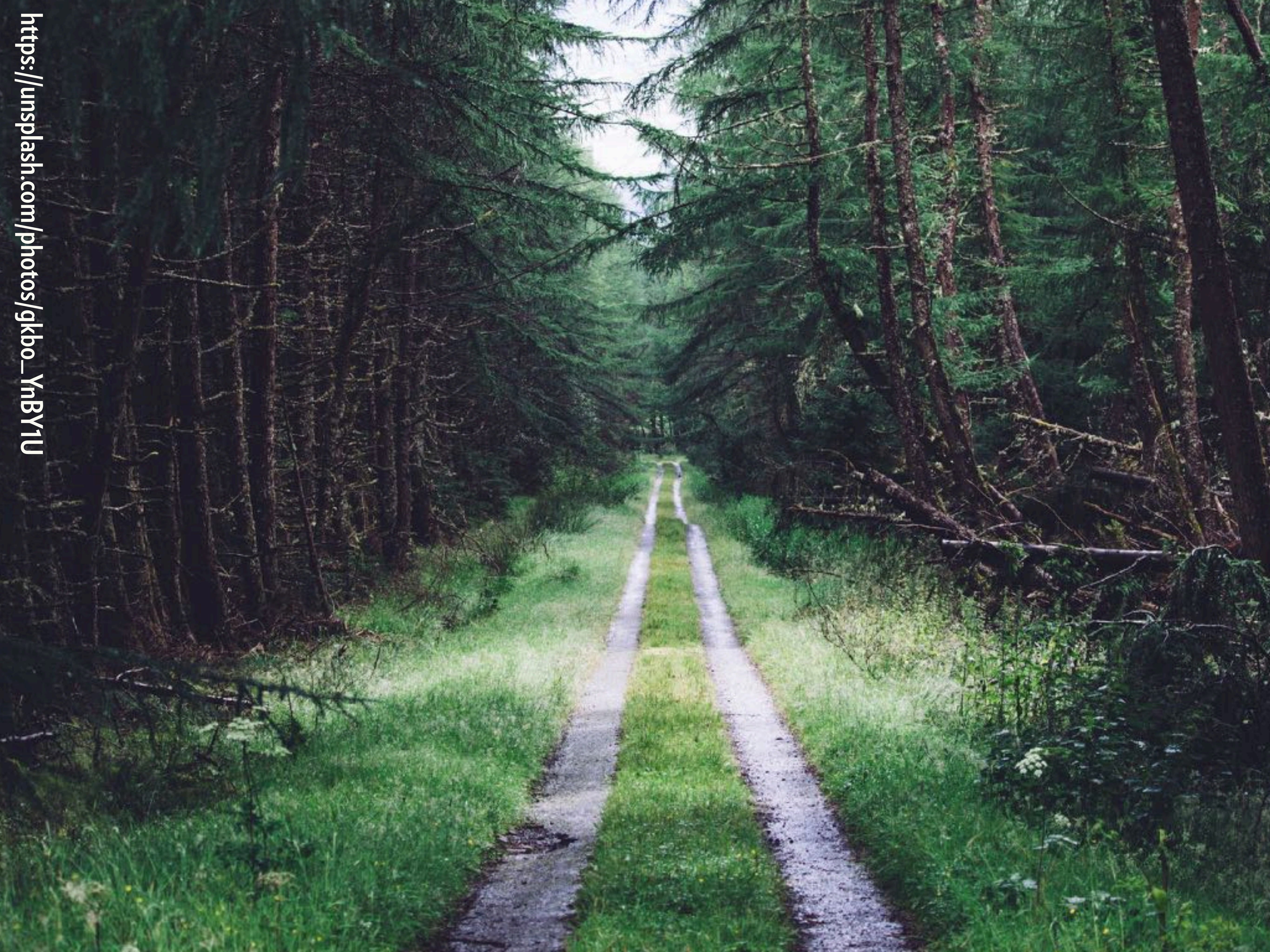
Transform

Visualise

Model

Communicate

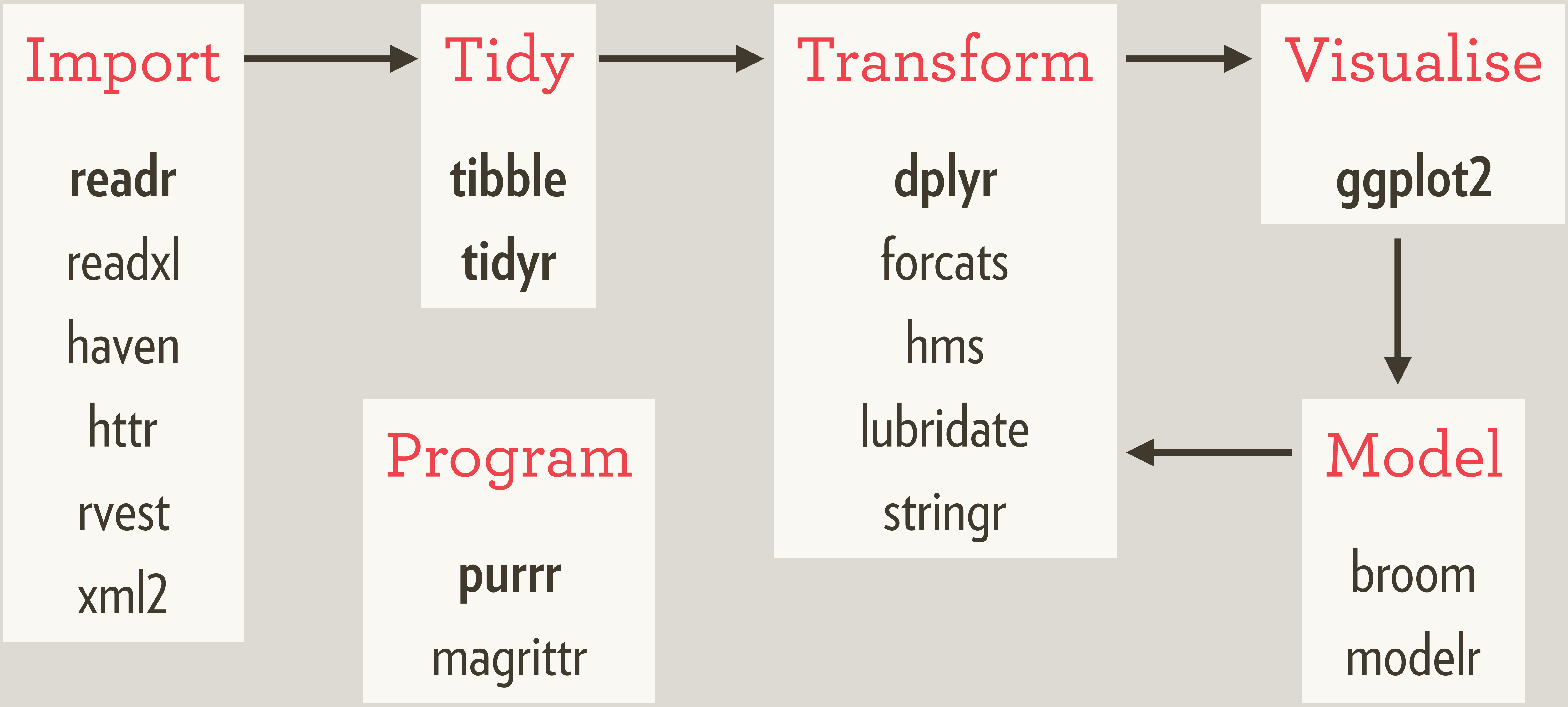
The tidyverse



https://unsplash.com/photos/gkbo_YnBY1U

Four practical things you should know

1. It exists
2. It has a website
3. It has a package
4. It has a book



Import

readr

readxl

haven

httr

rvest

xml2

Tidy

tibble

tidyr

Transform

dplyr

forcats

hms

lubridate

stringr

Visualise

ggplot2

Model

broom

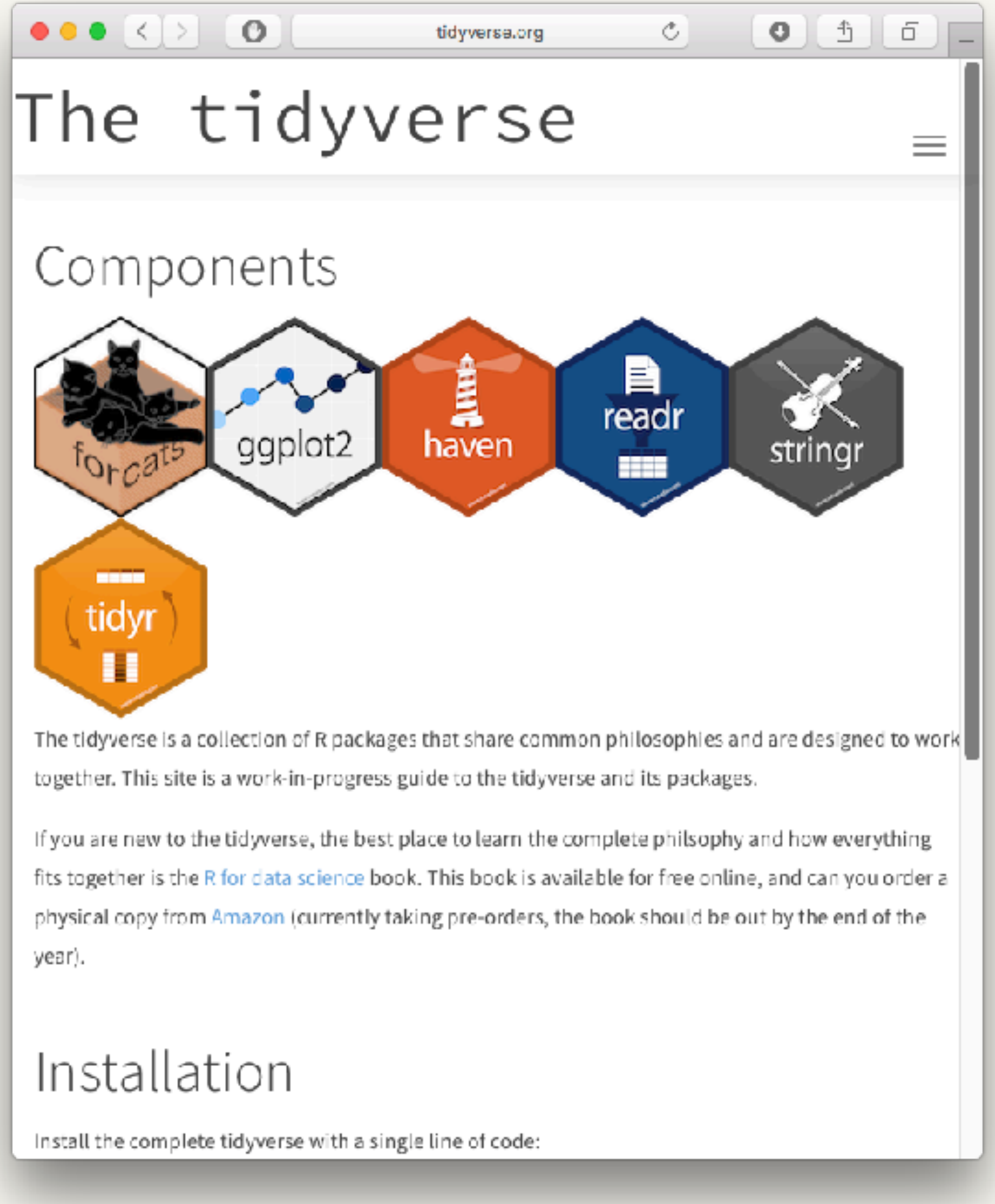
modelr

Program

purrr

magrittr

tidyverse.org



The tidyverse

Components

The tidyverse is a collection of R packages that share common philosophies and are designed to work together. This site is a work-in-progress guide to the tidyverse and its packages.

If you are new to the tidyverse, the best place to learn the complete philsophy and how everything fits together is the R for data science book. This book is available for free online, and can you order a physical copy from Amazon (currently taking pre-orders, the book should be out by the end of the year).

Installation

Install the complete tidyverse with a single line of code:

What am I missing out on?

How can I learn more?

How can I get help?

Installing tidyverse installs everything

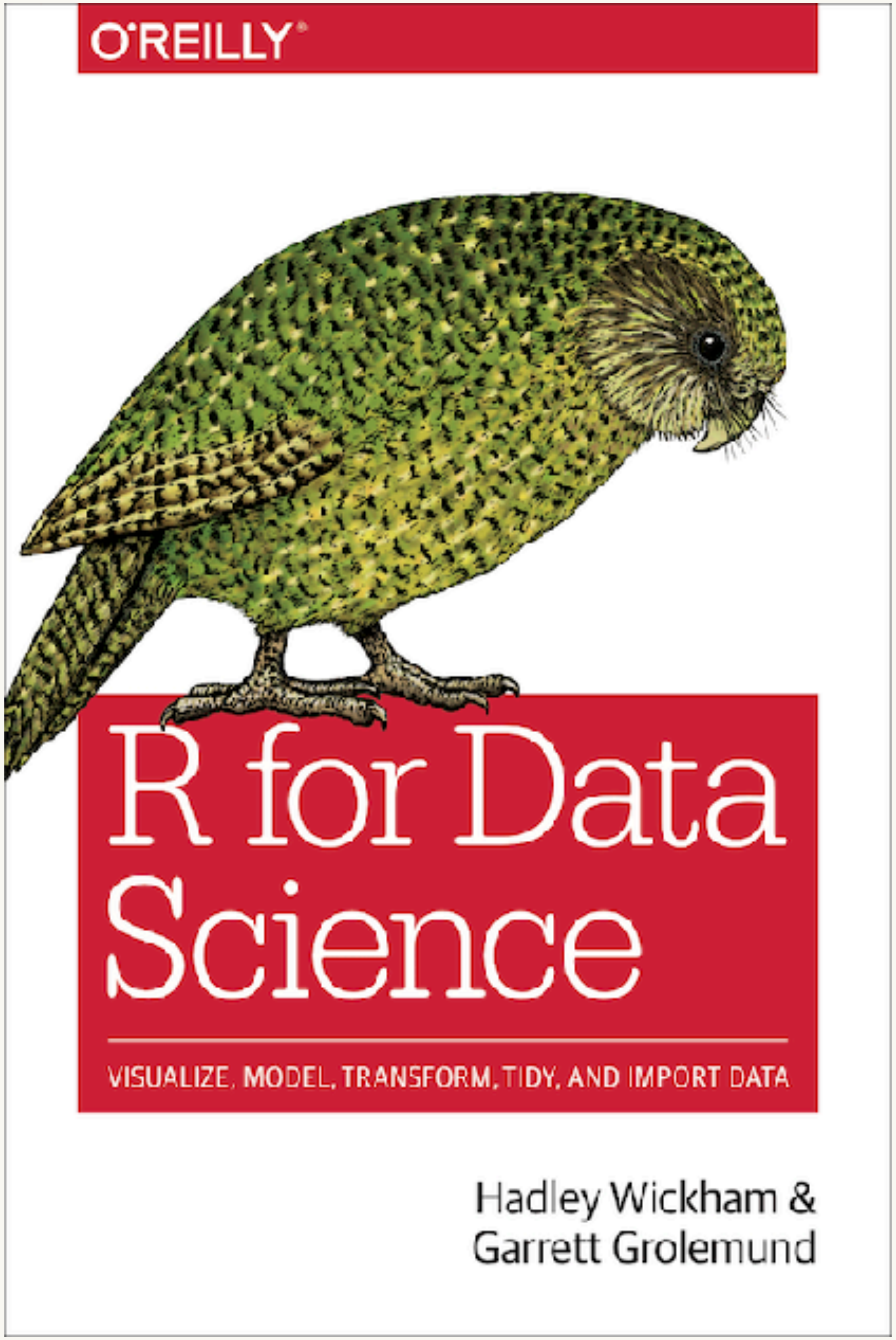
```
install.packages("tidyverse")

# Instead of
install.packages(c(
  "broom", "dplyr", "feather",
  "forcats","ggplot2", "haven",
  "httr", "hms", "jsonlite",
  "lubridate", "magrittr",
  "modelr", "purrr", "readr",
  "readxl", "stringr", "tibble",
  "rvest", "tidyr", "xml2"
))
```

Loading it loads the **core** tidyverse

```r
library(tidyverse)

# Instead of:
library(ggplot2)
library(tibble)
library(tidyr)
library(readr)
library(purrr)
library(dplyr)

# These are the packages you use in almost
# every analysis
```

Read online:
r4ds.had.co.nz

O'Reilly discount:
AUTHD



O'REILLY

R for Data Science

VISUALIZE, MODEL, TRANSFORM, TIDY, AND IMPORT DATA

Hadley Wickham &
Garrett Grolemund

https://unsplash.com/photos/JxLf4KYVT-A

Goal: Solve complex problems by combining simple, uniform pieces.

Consistent functions

The tidyverse separates **commands** and **queries**

A **command** function performs an action

A **query** function computes a value

Closely ideas: *referential transparency* and *functional purity*

Which is which?

mutate()

write_csv()

print()

summarise()

+ geom_line()

<-

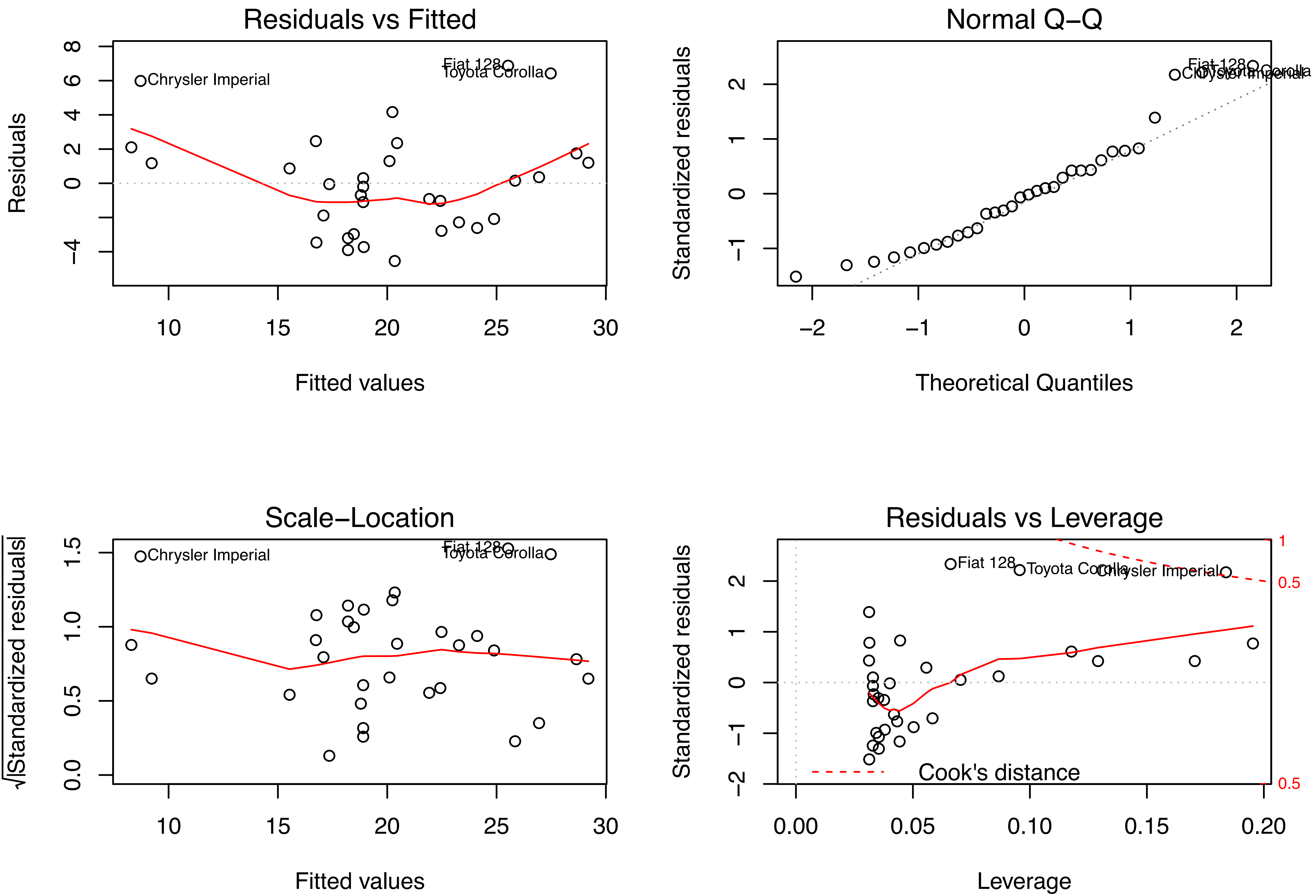
plot()

What’s the difference?

```
# Command
print()
plot()
write_csv()
<-

# Query
summarise()
mutate()
+ geom_line()
```

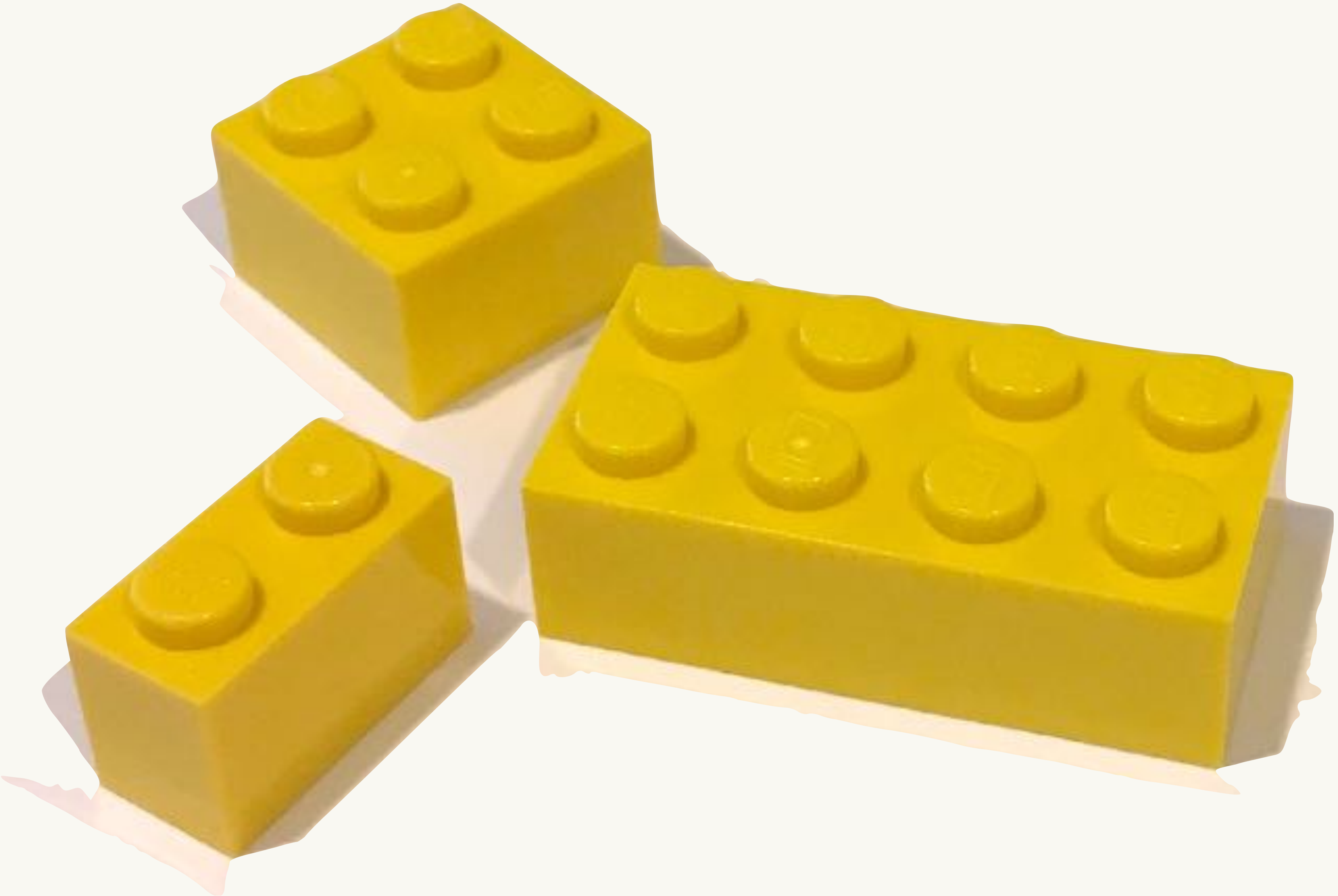
Base R generally sticks to this principle

```
mod <- lm(mpg ~ wt, data = mtcars)
summary(mod)
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)   37.285      1.878   19.86  < 2e-16 ***
#> wt            -5.344      0.559   -9.56  1.3e-10 ***
#> ---
#>
#> Residual standard error: 3.05 on 30 degrees of freedom
#> Multiple R-squared:  0.753, Adjusted R-squared:  0.745
#> F-statistic: 91.4 on 1 and 30 DF,  p-value: 1.29e-10
```


Residuals vs Fitted
Residuals
Fitted values
−4
0
2
4
6
8
10
15
20
25
30
Chrysler Imperial
Fiat 128
Toyota Corolla
Normal Q−Q
Standardized residuals
Theoretical Quantiles
−1
0
1
2
−2
−1
0
1
2
Fiat 128
Toyota Corolla
Chrysler Imperial
Scale−Location
√|Standardized residuals|
Fitted values
0.0
0.5
1.0
1.5
10
15
20
25
30
Chrysler Imperial
Fiat 128
Toyota Corolla
Residuals vs Leverage
Standardized residuals
Leverage
−2
−1
0
1
2
0.00
0.05
0.10
0.15
0.20
Fiat 128
Toyota Corolla
Chrysler Imperial
Cook's distance
1
0.5
0.5

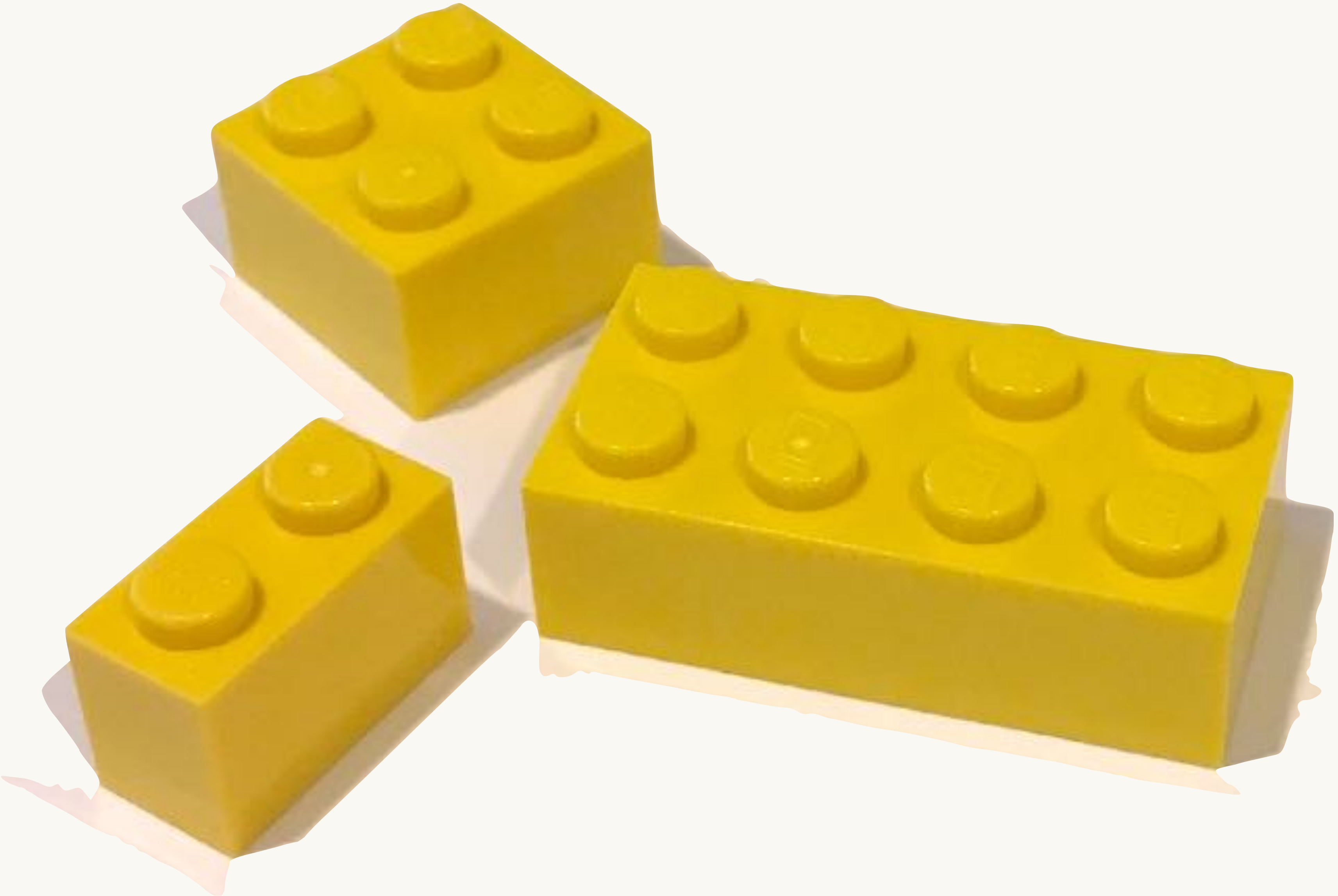
Query functions are like legos

(As long as you pick a consistent data structure)



Command functions are like playmobil

https://unsplash.com/photos/0VNVxhEnkII



http://brickartist.com/gallery/pc-magazine-computer/. CC-BY-NC

The pipe

Goal: Solve complex problems by **combining** uniform pieces.

O-H

https://unsplash.com/photos/hjEesK4KSDs

We already have ways to combine functions

```
by_dest <- group_by(flights, dest)
dest_delay <- summarise(by_dest,
  delay = mean(dep_delay, na.rm = TRUE),
  n = n()
)
big_dest <- filter(dest_delay, n > 100)
arrange(big_dest, desc(delay))
```

But naming is hard work

```
foo <- group_by(flights, dest)
foo <- summarise(foo,
  delay = mean(dep_delay, na.rm = TRUE),
  n = n()
)
foo <- filter(foo, n > 100)
arrange(foo, desc(delay))
```

But naming is hard work

```r
foo1 <- group_by(flights, dest)
foo2 <- summarise(foo1,
  delay = mean(dep_delay, na.rm = TRUE),
  n = n()
)
foo3 <- filter(foo2, n > 100)
arrange(foo3, desc(delay))
```

You *could* nest function calls

```
arrange(
  filter(
    summarise(
      group_by(flights, dest),
      delay = mean(dep_delay, na.rm = TRUE),
      n = n()
    ),
    n > 100
  ),
  desc(delay)
)
```

magrittr::

%>%

This is easy to read & doesn’t require naming

```
flights %>%
  group_by(dest) %>%
  summarise(
    delay = mean(dep_delay, na.rm = TRUE),
    n = n()
  ) %>%
  filter(n > 100) %>%
  arrange(desc(delay))
```

Writing readable code with pipes

Bob Rudis

Friday • 3:10pm

ggplot2 was written before the pipe

```
flights %>%
  group_by(date) %>%
  summarise(n = n()) %>%
  ggplot(aes(date, n)) +
  geom_line()
```

And is inconsistent

```
ggsave(
  flights %>%
    group_by(date) %>%
    summarise(n = n()) %>%
    ggplot(aes(date, n)) +
    geom_line(),
  "my-plot.pdf"
)
```

The command-query distinction is useful for pipes

The body is made up of **queries**

Every pipe is ended by a **command**

Where is the command function?

```
flights %>%
  group_by(dest) %>%
  summarise(
    delay = mean(dep_delay, na.rm = TRUE),
    n = n()
  ) %>%
  filter(n > 100) %>%
  arrange(desc(delay))
```

In the absence of a command, R prints

```
flights %>%
  group_by(dest) %>%
  summarise(
    delay = mean(dep_delay, na.rm = TRUE),
    n = n()
  ) %>%
  filter(n > 100) %>%
  arrange(desc(delay)) %>%
  print()
```

Another common command is **assign**

```
flights %>%
  group_by(dest) %>%
  summarise(
    delay = mean(dep_delay, na.rm = TRUE),
    n = n()
  ) %>%
  filter(n > 100) %>%
  arrange(desc(delay)) ->
  dest_delays
```

But leading with assignment improves readability

```
dest_delays <- flights %>%
  group_by(dest) %>%
  summarise(
    delay = mean(dep_delay, na.rm = TRUE),
    n = n()
  ) %>%
  filter(n > 100) %>%
  arrange(desc(delay))
```

Functions fit best into a pipe when:

1. The first argument is the “data”
2. The data is the same type across a family of functions

Tidy data

Goal: Solve complex problems by combining simple, **uniform pieces.**

Tidy data is a consistent way of storing data

1. Each **dataset** goes in a **data frame**.
2. Each **variable** goes in a **column**.

Happy families are all alike;
every unhappy family is
unhappy in its own way

— *Leo Tolstoy*

Tidy datasets are all alike; every messy dataset is messy in its own way

— *Hadley Wickham*

Messy data has a varied shape

```
# A tibble: 5,769 × 22
    iso2  year   m04  m514  m014 m1524 m2534 m3544 m4554 m5564   m65    mu   f04  f514  f014 f1524
   <chr> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
1     AD  1989    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
2     AD  1990    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
3     AD  1991    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
4     AD  1992    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
5     AD  1993    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
6     AD  1994    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
7     AD  1996    NA    NA     0     0     0     4     1     0     0    NA    NA    NA     0     1
8     AD  1997    NA    NA     0     0     1     2     2     1     6    NA    NA    NA     0     1
9     AD  1998    NA    NA     0     0     0     1     0     0     0    NA    NA    NA    NA    NA
10    AD  1999    NA    NA     0     0     0     1     1     0     0    NA    NA    NA     0     0
11    AD  2000    NA    NA     0     0     1     0     0     0     0    NA    NA    NA    NA    NA
12    AD  2001    NA    NA     0    NA    NA     2     1    NA    NA    NA    NA    NA    NA    NA
13    AD  2002    NA    NA     0     0     0     1     0     0     0    NA    NA    NA     0     1
14    AD  2003    NA    NA     0     0     0     1     2     0     0    NA    NA    NA     0     1
15    AD  2004    NA    NA     0     0     0     1     1     0     0    NA    NA    NA     0     0
16    AD  2005     0     0     0     0     1     1     0     0     0     0     0     0     0     1
17    AD  2006     0     0     0     1     1     2     0     1     1     0     0     0     0     0
# ... with 5,752 more rows, and 6 more variables: f2
#   f5564 <int>, f65 <int>, fu <int>
```

What are the variables in this dataset?

(Hint: f = female, u = unknown, 1524 = 15-24)

Tidy data has a uniform shape

```
# A tibble: 35,750 × 5
   country  year   sex   age     n
     <chr> <int> <chr> <chr> <int>
1       AD  1996     f   014     0
2       AD  1996     f  1524     1
3       AD  1996     f  2534     1
4       AD  1996     f  3544     0
5       AD  1996     f  4554     0
6       AD  1996     f  5564     1
7       AD  1996     f    65     0
8       AD  1996     m   014     0
9       AD  1996     m  1524     0
10      AD  1996     m  2534     0
# ... with 35,740 more rows
```

tidytext

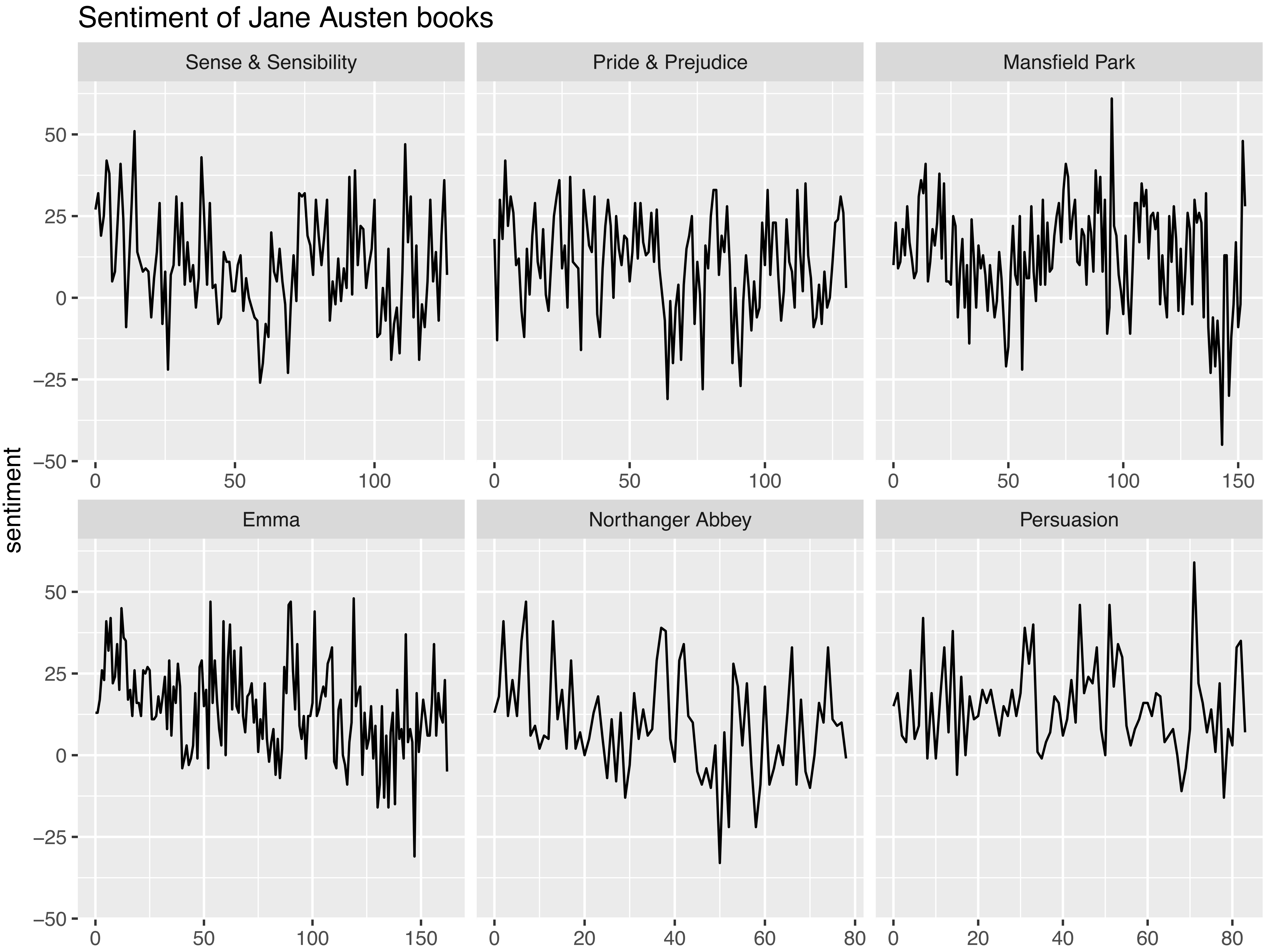
by Julia Silge & David Robinson

The family of Dashwood had long been settled in Sussex. Their estate was large, and their residence was at Norland Park, in the centre of their property, where, for many generations, they had lived in so respectable a manner as to engage the general good opinion of their surrounding acquaintance.

— *Sense & Sensibility*, **Jane Austen**

tidytext provides an answer

```
# A tibble: 724,880 × 4
                  book linenumber chapter     word
                <fctr>      <int>   <int>    <chr>
1  Sense & Sensibility         10       1  chapter
2  Sense & Sensibility         10       1        1
3  Sense & Sensibility         13       1      the
4  Sense & Sensibility         13       1   family
5  Sense & Sensibility         13       1       of
6  Sense & Sensibility         13       1 dashwood
7  Sense & Sensibility         13       1      had
8  Sense & Sensibility         13       1     long
9  Sense & Sensibility         13       1     been
10 Sense & Sensibility         13       1  settled
# ... with 724,870 more rows
```


Sentiment of Jane Austen books
Sense & Sensibility
Pride & Prejudice
Mansfield Park
Emma
Northanger Abbey
Persuasion
sentiment
50
25
0
−25
−50
0
50
100
150
0
20
40
60
80

Text mining, the tidy way

Julia Silge

Friday • 3:51pm

http://tidytextmining.com

list-cols

Tidy tibbles are better than tidy data frames

1. Each **dataset** goes in a tibble.
2. Each **variable** goes in a column.

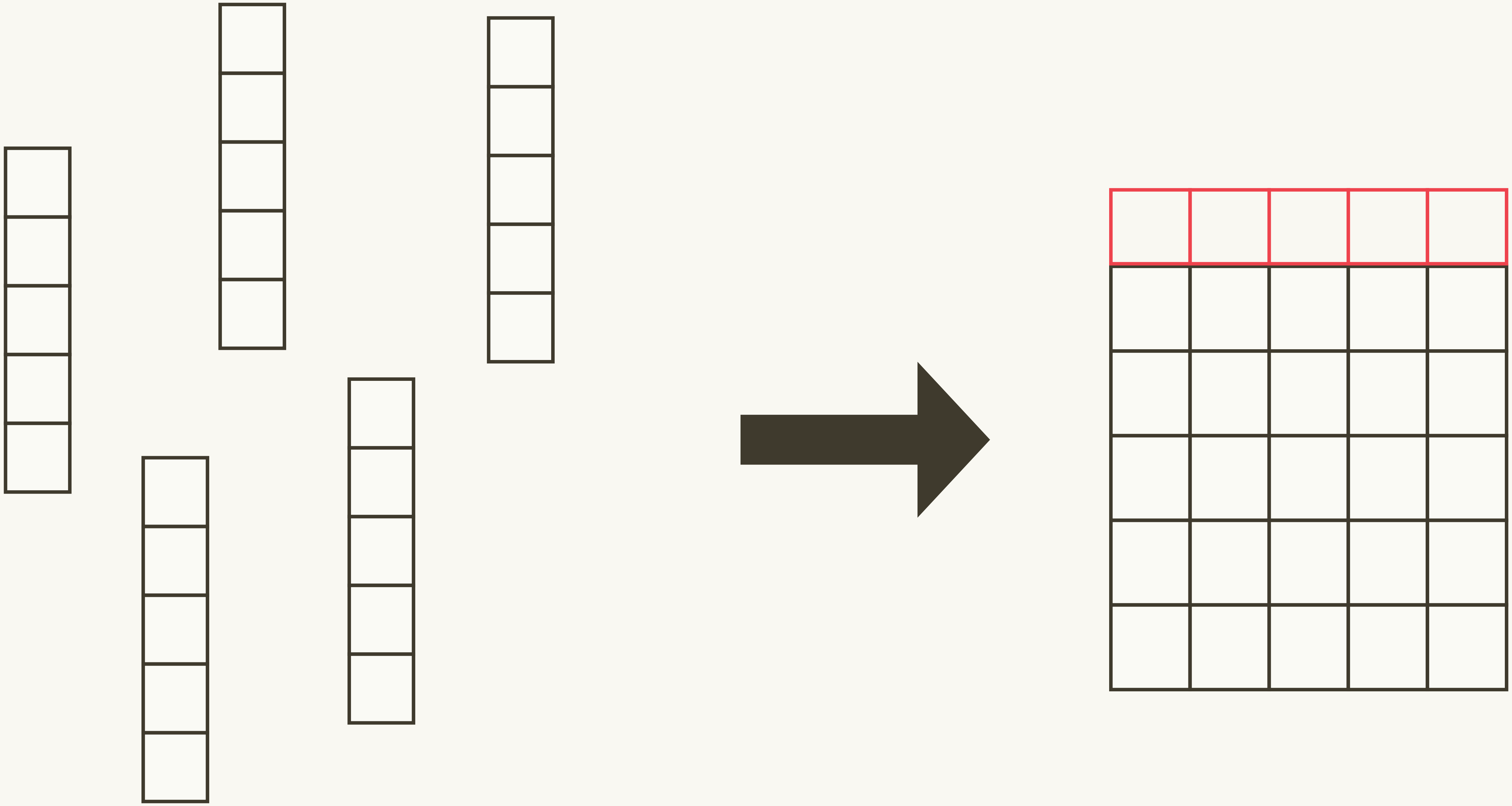
Tibbles are data frames that are **lazy** & **surly**

```
df <- tibble(xyz = "a")

df$x
#> Warning: Unknown column 'x'
#> NULL

df$xyz
#> [1] "a"
```

But also have better support for list-cols

```
data.frame(x = list(1:2, 3:5))
#> Error: arguments imply differing number
#> of rows: 2, 3

tibble(x = list(1:2, 3:5))
#> # A tibble: 2 x 1
#>           x
#>      <list>
#> 1 <int [2]>
#> 2 <int [3]>
```

List-columns keep related things together

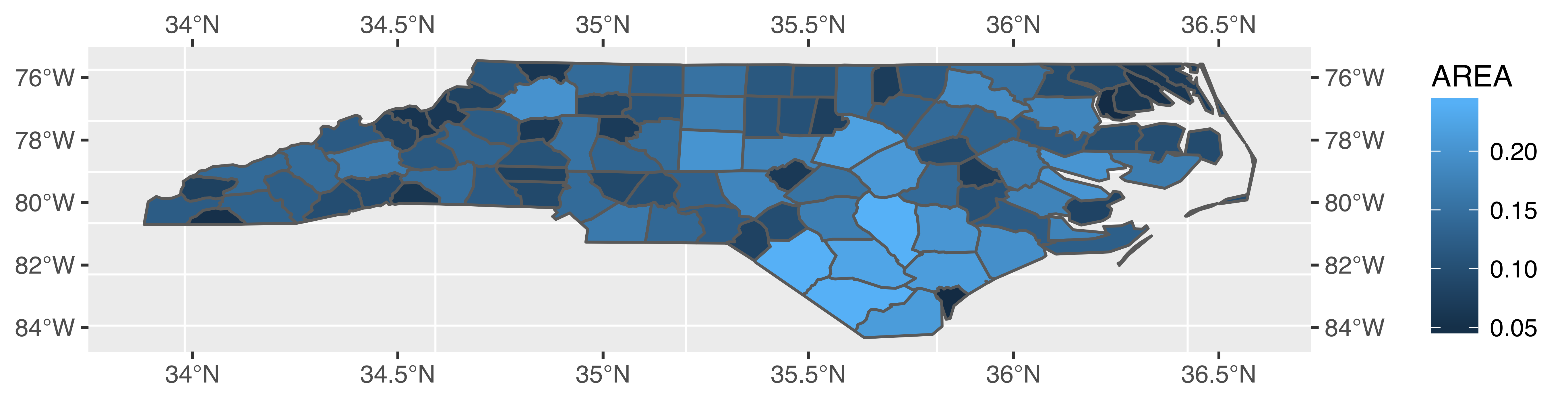


Anything can go in a list & a list can go in a data frame

sf (successor to sp) uses list-cols

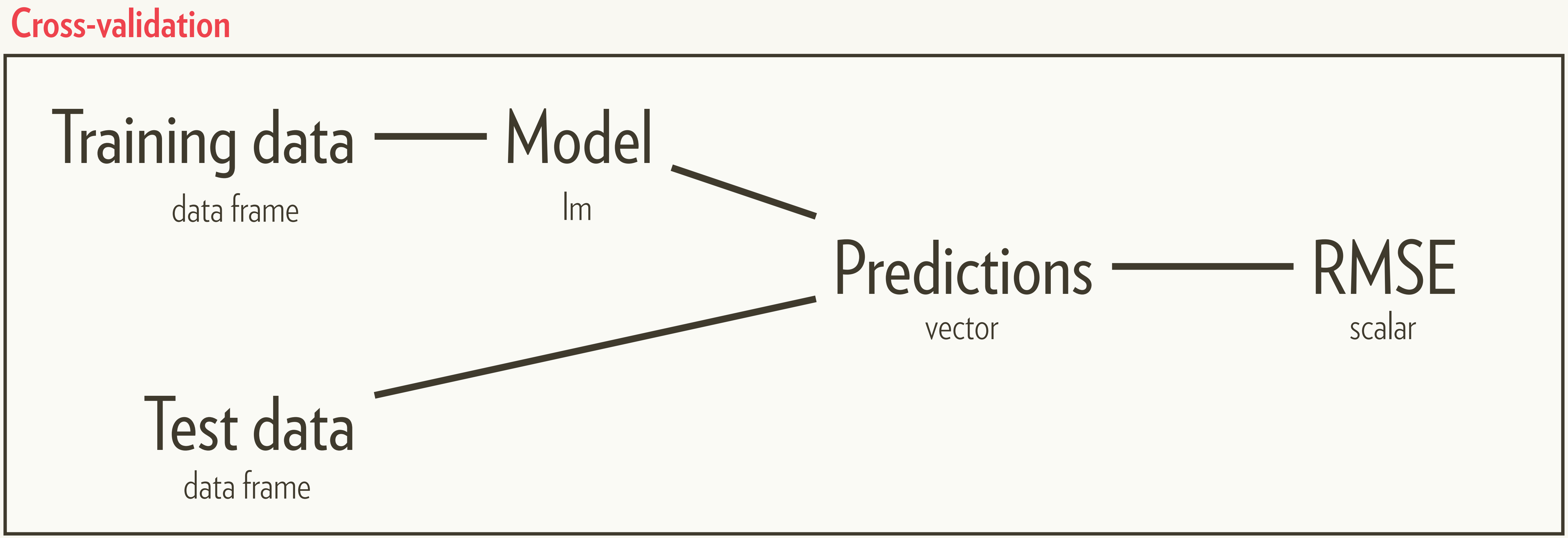
by Edzer Pebesma

```
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"))

nc %>%
  as_tibble() %>%
  select(NAME, FIPS, AREA, geometry)
#> # A tibble: 100 × 4
#>            NAME   FIPS  AREA          geometry
#>          <fctr> <fctr> <dbl>  <simple_feature>
#> 1          Ashe  37009 0.114 <MULTIPOLYGON...>
#> 2     Alleghany  37005 0.061 <MULTIPOLYGON...>
#> 3         Surry  37171 0.143 <MULTIPOLYGON...>
#> 4     Currituck  37053 0.070 <MULTIPOLYGON...>
#> 5   Northampton  37131 0.153 <MULTIPOLYGON...>
#> 6      Hertford  37091 0.097 <MULTIPOLYGON...>
#> 7        Camden  37029 0.062 <MULTIPOLYGON...>
#> 8         Gates  37073 0.091 <MULTIPOLYGON...>
#> 9        Warren  37185 0.118 <MULTIPOLYGON...>
#> 10       Stokes  37169 0.124 <MULTIPOLYGON...>
#> # ... with 90 more rows
```



```
nc %>%
  ggplot(aes(geometry = geometry, fill = AREA)) +
    geom_sf() +
    coord_sf(crs = sf::st_crs(nc))
```

list-cols are a beautiful fit to cross-validation



Each resample becomes one row

```
# A tibble: 100 x 5
             train                test   .id       mod      rmse
            <list>              <list> <chr>    <list>     <dbl>
1  <S3: resample> <S3: resample>   001 <S3: lm> 0.5661605
2  <S3: resample> <S3: resample>   002 <S3: lm> 0.2399357
3  <S3: resample> <S3: resample>   003 <S3: lm> 3.5482986
4  <S3: resample> <S3: resample>   004 <S3: lm> 0.2396810
5  <S3: resample> <S3: resample>   005 <S3: lm> 0.1591336
6  <S3: resample> <S3: resample>   006 <S3: lm> 0.1934869
7  <S3: resample> <S3: resample>   007 <S3: lm> 0.2697834
8  <S3: resample> <S3: resample>   008 <S
9  <S3: resample> <S3: resample>   009 <S
10 <S3: resample> <S3: resample>   010 <S
... with 90 more rows
```

Putting square pegs in round holes

Jenny Bryan

Friday • 3:31pm

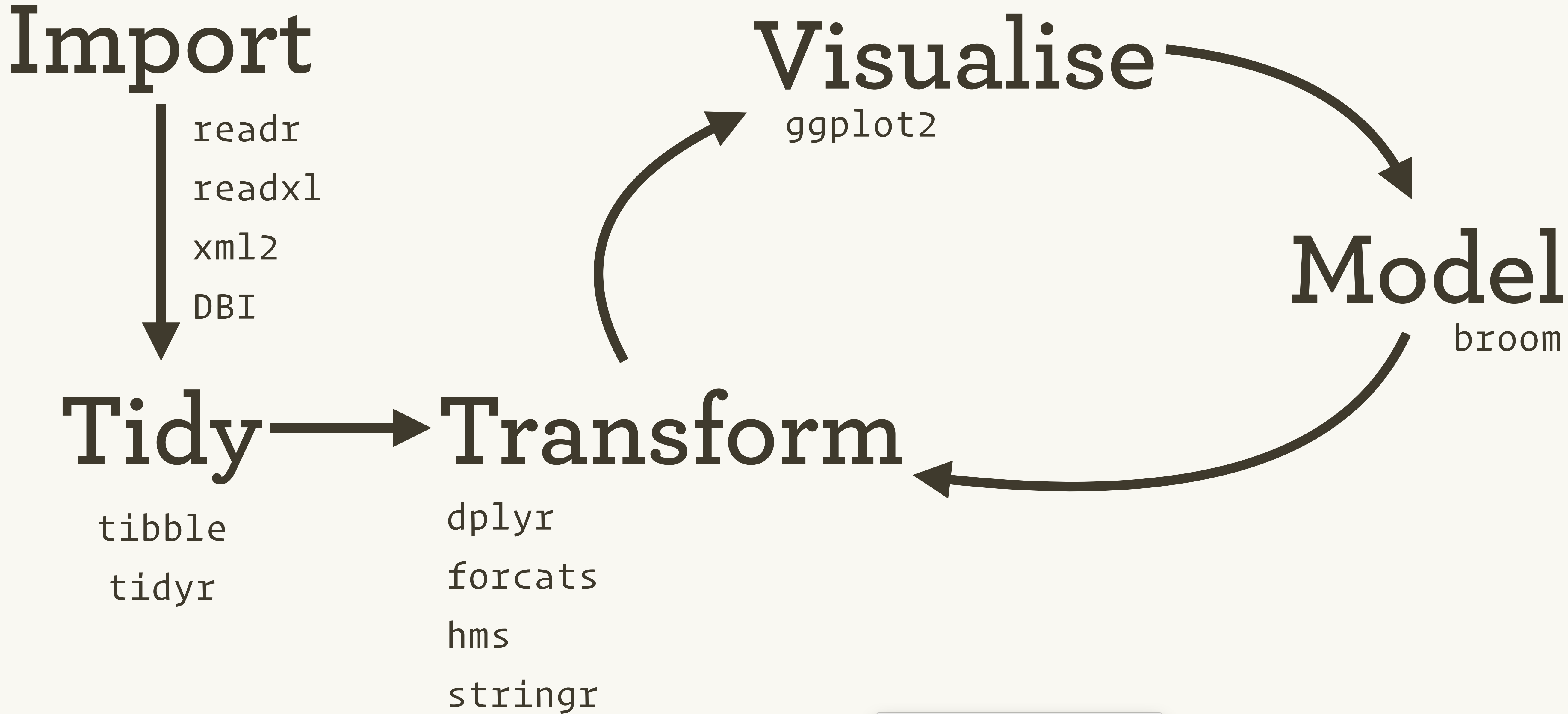
Conclusion

Four important facts:

1. It exists
2. It has a website
3. It has a package
4. It has a book

Four underlying principles:

1. Each function encapsulates one task
2. And is either a query or a command
3. Functions are composed with %>%
4. And use tidy tibbles as primary data structure



Import

readr

readxl

xml2

DBI

Tidy

tibble

tidyr

Transform

dplyr

forcats

hms

stringr

lubridate

Visualise

ggplot2

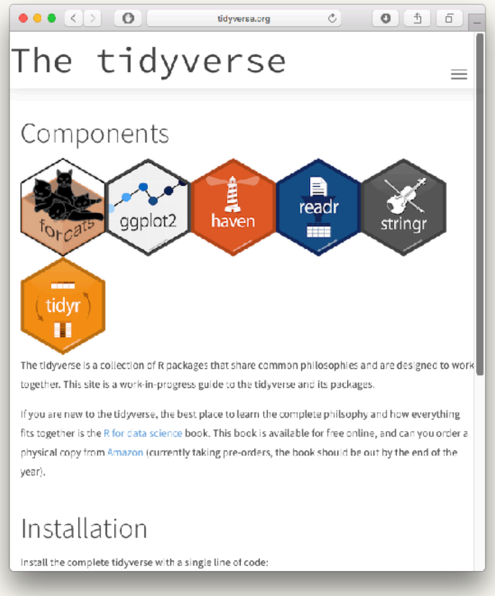
Model

broom

purrr

magrittr

Program



The tidyverse

Components

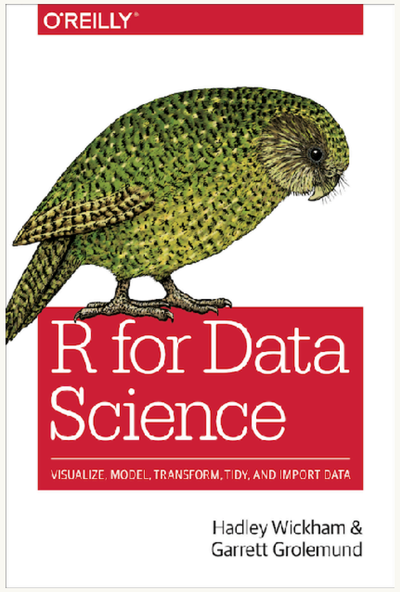
The tidyverse is a collection of R packages that share common philosophies and are designed to work together. This site is a work-in-progress guide to the tidyverse and its packages.

If you are new to the tidyverse, the best place to learn the complete philsophy and how everything fits together is the R for data science book. This book is available for free online, and can you order a physical copy from Amazon (currently taking pre-orders, the book should be out by the end of the year).

Installation

Install the complete tidyverse with a single line of code:

tidyverse.org



r4ds.had.co.nz

Tutorials

Functional programming
Charlotte Wickham

```
install.packages("tidyverse")
```

Shiny dashboards
Winston Chang & Joe Cheng

```
devtools::install_github("jcheng5/dashtutorial")
```