A (space) travel into the Tidyverse 🚀

Filippo Gambarota

University of Padova - Psicostat

10/11/2020

Contents

- What is the **tidyverse** and the **tidy approach**
- The main packages and function
- Other **tidy** packages
- Some examples

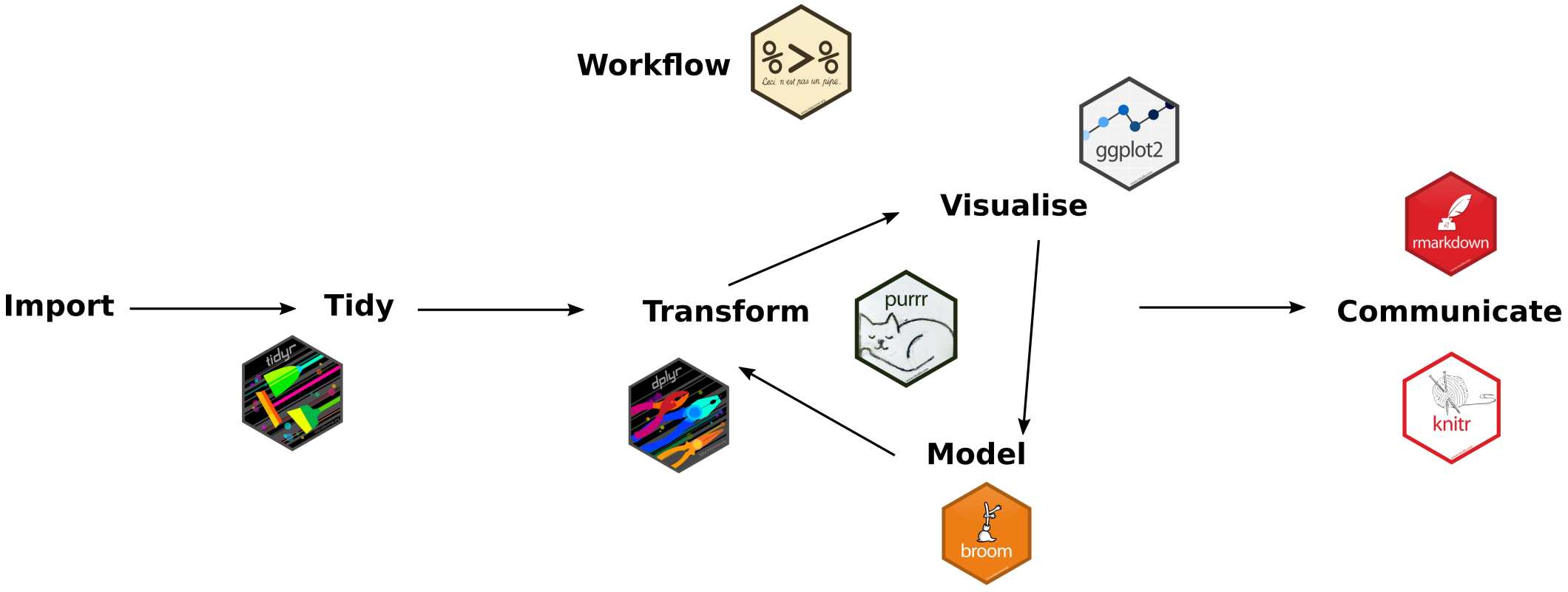
What is the Tidyverse?

Tidyverse

Hadley Wickham - RStudio Data Scientist

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

The big picture



What is the Tidy approach?

What is the tidy approach?

- The best way to format data is the **long format**

- Concatenate operations with **pipes**

- Focus on a **functional programming approach**

Long-format data

- Each row is an **observation** and each column is a **variable**

```
## # A tibble: 6 x 4
##      id      A       B       C
##   <int>  <dbl>   <dbl>   <dbl>
## 1     1  0.812 -0.885   2.13
## 2     2 -0.209 -1.08    1.14
## 3     3  0.930  0.577  -1.48
## 4     4  0.427 -0.558   0.910
## 5     5  0.733  0.426  -0.0918
## 6     6  1.24   0.0672 -0.233
```

```
## # A tibble: 6 x 3
##      id cond     cov
##   <int> <chr>  <dbl>
## 1     1 A      0.812
## 2     1 B     -0.885
## 3     1 C      2.13
## 4     2 A     -0.209
## 5     2 B     -1.08
## 6     2 C      1.14
```

Concatenate operations with **pipes**

- Pipes are some operators from the `magrittr::` package with the aim of **improving the code readability and maintainability** [1]
- There are several different **pipes** but the most used (and useful) is the `%>%`
- Pipes are integrated with all **tidyverse** functions and packages

[1] magrittr website

Concatenate operations with **pipes**

- The `%>%` pipe is another way to **declare the function with an argument**
- If `.f` is a function and `.x` is an object, this `.f(x)` is equivalent to `.x %>% .f`

```
mean(iris$Sepal.Length)
```

```
## [1] 5.843333
```

```
iris$Sepal.Length %>%
  mean()
```

```
## [1] 5.843333
```

Concatenate operations with **pipes**

The previous simple example is not completely appropriate, the `pipe` is useless. However let's assume a more complicated example:

```
head(dat)
```

```
## # A tibble: 6 x 6
##      id cond1 cond2 value   cov1  cov2
##   <int> <chr> <chr> <dbl>  <dbl> <dbl>
## 1     1 A     1     111.  -0.604 0.961
## 2     1 A     2     105.  -0.604 0.961
## 3     1 A     3      86.5 -0.604 0.961
## 4     1 B     1      90.5 -0.604 0.961
## 5     1 B     2      99.8 -0.604 0.961
## 6     1 B     3     112.  -0.604 0.961
```

1. aggregate data by a `factor` using the `mean()`
2. create a new columns with some operations between columns
3. rename a variable

Without pipes and tidyverse

```
dat <- aggregate(value ~ id + cond1 + cov1 + cov2, mean, data = dat) # aggregate by cond1
dat$cov1 <- dat$cov1 - mean(dat$cov1) # center
dat$cov2 <- (dat$cov2 - mean(dat$cov2))/sd(dat$cov2) # z point
names(dat)[1] <- "subject" # rename
head(dat)
```

```
##   subject cond1       cov1       cov2     value
## 1       2     A -1.3466351 -1.9227259  94.34698
## 2       2     B -1.3466351 -1.9227259 101.21024
## 3       2     C -1.3466351 -1.9227259  96.80649
## 4       9     A -0.2361874 -0.7775829 100.67230
## 5       9     B -0.2361874 -0.7775829  94.99126
## 6       9     C -0.2361874 -0.7775829  97.90249
```

- This works fine but is a little bit **redundant**, **difficult to read** and there is a **series of assignment operations**
- Some columns are not in the correct order
- In order to have a new `dat`, you can create a `dat_agg` or overwrite the `dat` object

With pipes and tidyverse

```
dat %>%
  mutate(cov1 = cov1 - mean(cov1),
         cov2 = (cov2 - mean(cov2))/sd(cov2)) %>%
  rename("subject" = id) %>%
  group_by(subject, cond1, cov1, cov2) %>%
  summarise(mean = mean(value),
            sd = sd(value)) %>%
  ungroup() %>%
  head()
```

```
## `summarise()` regrouping output by 'subject', 'cond1', 'cov1' (override with `.groups` argument)
```

```
## # A tibble: 6 x 6
##   subject cond1   cov1   cov2  mean    sd
##     <int> <chr>  <dbl>  <dbl> <dbl> <dbl>
## 1       1 A      0.880  0.239 106.   8.33
## 2       1 B      0.880  0.239  97.4 10.4
## 3       1 C      0.880  0.239 104.  11.4
## 4       2 A     -0.472 -1.29   93.2  5.35
## 5       2 B     -0.472 -1.29  103.  10.1
## 6       2 C     -0.472 -1.29   97.8  2.91
```

With pipes and tidyverse

```
dat %>%
  mutate(cov1 = cov1 - mean(cov1),
         cov2 = (cov2 - mean(cov2))/sd(cov2)) %>%
  rename("subject" = id) %>%
  group_by(subject, cond1, cov1, cov2) %>%
  summarise(mean = mean(value),
            sd = sd(value)) %>%
  ungroup() %>%
  head()
```

- The `dat` object is not modified
- Operations follows an **easy to read workflow of operations**
- If you want to assign you can use `<-` at the beginning as `dat <- dat %>% ...`

Functional Programming

Without technical details, the idea of functional programming is the comparison between a `for loop` and a `apply family` function [2]

```
means <- vector("double", ncol(mtcars))
medians <- vector("double", ncol(mtcars))

for(i in seq_along(mtcars)) {
    means[[i]] <- mean(mtcars[[i]], na.rm = TRUE)
    medians[[i]] <- median(mtcars[[i]], na.rm = TRUE)
}
```

```
means <- lapply(mtcars, function(x) mean(x))
median <- lapply(mtcars, function(x) median(x))
```

[2] From the Hadley Wickam talk - Managing many models with R

Functional Programming with purrr::

- Purrr is as package that provides a series of **apply like** functions in order to perform complex and fast operations
- Furrr is the same package as `purrr` but with a `future` implementation in order to parallelize the operations

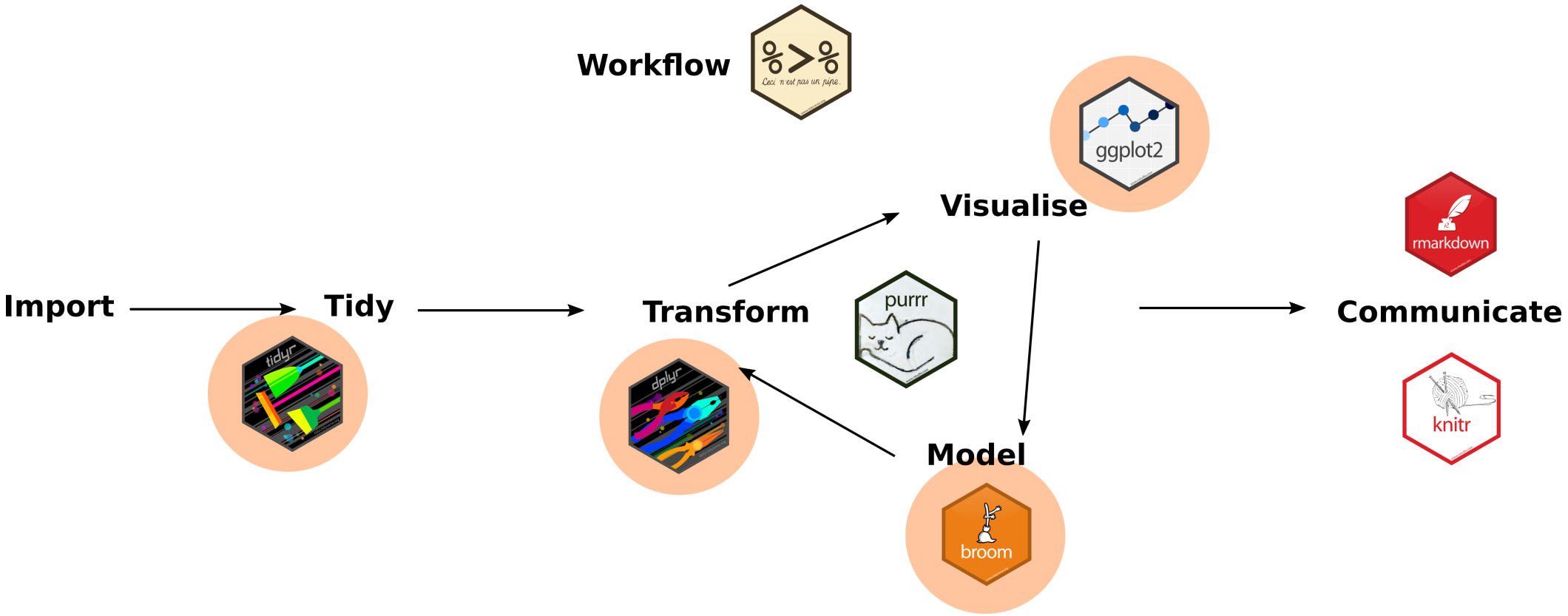
```
means <- map_dbl(mtcars, mean)
means[1:4]
```

```
##        mpg        cyl       disp         hp
##   20.09062    6.18750  230.72188  146.68750
```

```
# sapply(mtcars, function(x) mean(x))
```

Main packages and functions

Main packages and functions



Tidyr

- Manipulate datasets to have **tidy** data
- Main functions are `pivot_longer()`, `pivot_wider`, and `separate()`
- Other functions like `drop_na()` and `nest()`

Pivoting datasets

From long to wide dataset

```
dat %>%
  head()
```

```
## # A tibble: 6 x 6
##      id cond1 cond2 value  cov1    cov2
##   <int> <chr> <chr> <dbl> <dbl>   <dbl>
## 1     1 A     1     101.  0.725 -0.0289
## 2     1 A     2     103.  0.725 -0.0289
## 3     1 A     3     116.  0.725 -0.0289
## 4     1 B     1     105.  0.725 -0.0289
## 5     1 B     2      85.4 0.725 -0.0289
## 6     1 B     3     102.  0.725 -0.0289
```

Tidyr::pivot_wider()

From long to wide dataset

```
dat %>%
  pivot_wider(names_from = c(cond1, cond2), values_from = value) %>%
  head()
```

```
## # A tibble: 6 x 12
##      id   cov1    cov2   A_1   A_2   A_3   B_1   B_2   B_3   C_1   C_2   C_3
##   <int>  <dbl>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1  0.725 -0.0289 101.  103.  116.  105.   85.4 102.  116.  101.   93.9
## 2     2 -0.627 -1.37    96.9  95.7  87.1  98.6  95.3 114.   99.8  94.5  99.2
## 3     3 -0.692 -1.81   104.  119.   94.7 115.   97.2  99.5 106.  107.  100.
## 4     4 -1.07  -0.430   98.0  97.5 134.   87.1  97.1 105.   85.6 112.   92.5
## 5     5  1.23   0.818  102.   77.9  87.3 107.  109.  108.   82.7 125.  106.
## 6     6  0.557  0.183  104.   87.0  94.9  85.8 112.   83.9  82.3  90.8  97.5
```

Tidyr::pivot_longer()

From wide to long dataset

```
dat %>%
  pivot_wider(names_from = c(cond1, cond2), values_from = value) %>%
  pivot_longer(4:12, names_to = "cond", values_to = "value") %>%
  head()
```

```
## # A tibble: 6 x 5
##      id  cov1    cov2 cond  value
##   <int> <dbl>   <dbl> <chr> <dbl>
## 1     1 0.725 -0.0289 A_1   101.
## 2     1 0.725 -0.0289 A_2   103.
## 3     1 0.725 -0.0289 A_3   116.
## 4     1 0.725 -0.0289 B_1   105.
## 5     1 0.725 -0.0289 B_2    85.4
## 6     1 0.725 -0.0289 B_3   102.
```

Tidyr::separate()

Separate a column in multiple columns considering a pattern

```
dat %>%
  pivot_wider(names_from = c(cond1, cond2), values_from = value) %>%
  pivot_longer(4:12, names_to = "cond", values_to = "value") %>%
  separate(cond, into = c("cond1", "cond2"), sep = "_") %>%
  head()
```

```
## # A tibble: 6 x 6
##      id  cov1    cov2 cond1 cond2 value
##   <int> <dbl>   <dbl> <chr> <chr> <dbl>
## 1     1 0.725 -0.0289 A     1     101.
## 2     1 0.725 -0.0289 A     2     103.
## 3     1 0.725 -0.0289 A     3     116.
## 4     1 0.725 -0.0289 B     1     105.
## 5     1 0.725 -0.0289 B     2      85.4
## 6     1 0.725 -0.0289 B     3     102.
```

Dplyr

- This is a very comprehensive package with several functions to create new columns, aggregate datasets, select rows, etc.
- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

Dplyr::mutate()

Create new columns with complex operations and functions

```
dat %>%
  mutate(new_col = some_functions(other_col))
```

```
dat %>%
  mutate(new_col = cov1 * cov2) %>%
  head()
```

```
## # A tibble: 6 x 7
##      id cond1 cond2 value  cov1    cov2 new_col
##   <int> <chr> <chr> <dbl> <dbl>   <dbl>   <dbl>
## 1     1 A     1     101.  0.725 -0.0289 -0.0210
## 2     1 A     2     103.  0.725 -0.0289 -0.0210
## 3     1 A     3     116.  0.725 -0.0289 -0.0210
## 4     1 B     1     105.  0.725 -0.0289 -0.0210
## 5     1 B     2      85.4 0.725 -0.0289 -0.0210
## 6     1 B     3     102.  0.725 -0.0289 -0.0210
```

Dplyr::select()

Select columns in a more readable way

```
dat %>%
  pivot_wider(names_from = c(cond1, cond2), values_from = value) %>%
  select(id, starts_with("A"), ends_with("3")) %>%
  head()
```

```
## # A tibble: 6 x 6
##      id   A_1   A_2   A_3   B_3   C_3
##   <int> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     1 101.  103.  116.  102.   93.9
## 2     2  96.9  95.7  87.1 114.   99.2
## 3     3 104.  119.   94.7  99.5 100.
## 4     4  98.0  97.5 134.  105.   92.5
## 5     5 102.   77.9  87.3 108.  106.
## 6     6 104.   87.0  94.9  83.9  97.5
```

Dplyr::filter()

Select rows with multiple conditions

```
dat %>%
  filter(cond1 == "A" & cond2 == 2) %>%
  head()
```

```
## # A tibble: 6 x 6
##      id cond1 cond2 value   cov1    cov2
##   <int> <chr> <chr> <dbl>  <dbl>   <dbl>
## 1     1 A     2     103.   0.725 -0.0289
## 2     2 A     2      95.7 -0.627 -1.37
## 3     3 A     2     119.  -0.692 -1.81
## 4     4 A     2      97.5 -1.07  -0.430
## 5     5 A     2      77.9  1.23   0.818
## 6     6 A     2      87.0  0.557  0.183
```

Dplyr::arrange()

Reorder rows based on multiple columns

```
dat %>%
  group_by(cond1, cond2) %>%
  summarise(mean = mean(value),
            sd = sd(value)) %>%
  arrange(cond1) %>%
  head()
```

```
## `summarise()` regrouping output by 'cond1' (override with `.groups` argument)
```

```
## # A tibble: 6 x 4
## # Groups:   cond1 [2]
##   cond1 cond2  mean    sd
##   <chr> <chr> <dbl> <dbl>
## 1 A     1     100.   10.3
## 2 A     2      95.4  11.1
## 3 A     3     107.   15.1
## 4 B     1      96.5  10.2
## 5 B     2     102.   11.8
## 6 B     3     102.   10.5
```

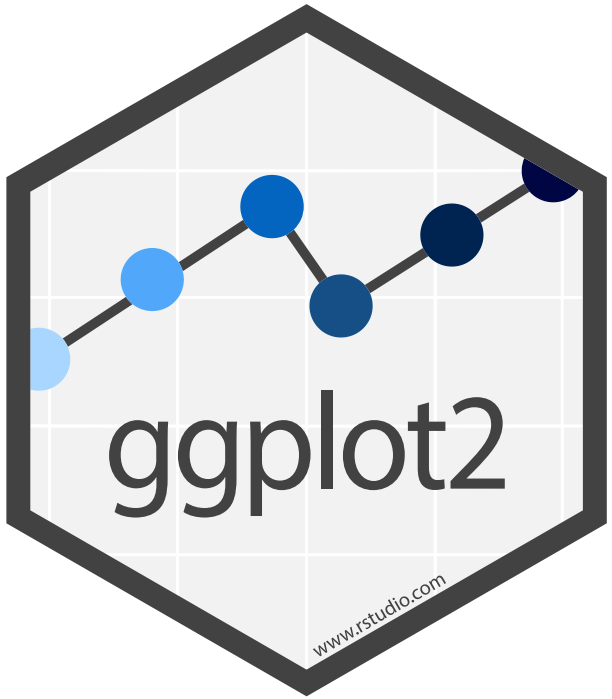
Dplyr::case_when()

Is an extension of a `ifelse()` statement in a more compact way

```
dat %>%
  mutate(new_fac = case_when(value > 100 & cond1 == "A" ~ "level1",
                             value < 50 & cond2 == 2 ~ "level2",
                             value != 100 & cond2 == 1 & cov1 > 0.5 ~ "level3",
                             TRUE ~ "level4")) %>%
  head()
```

```
## # A tibble: 6 x 7
##      id cond1 cond2 value  cov1    cov2 new_fac
##   <int> <chr> <chr> <dbl> <dbl>   <dbl> <chr>
## 1     1 A     1     101.  0.725 -0.0289 level1
## 2     1 A     2     103.  0.725 -0.0289 level1
## 3     1 A     3     116.  0.725 -0.0289 level1
## 4     1 B     1     105.  0.725 -0.0289 level3
## 5     1 B     2      85.4 0.725 -0.0289 level4
## 6     1 B     3     102.  0.725 -0.0289 level4
```

GGplot2

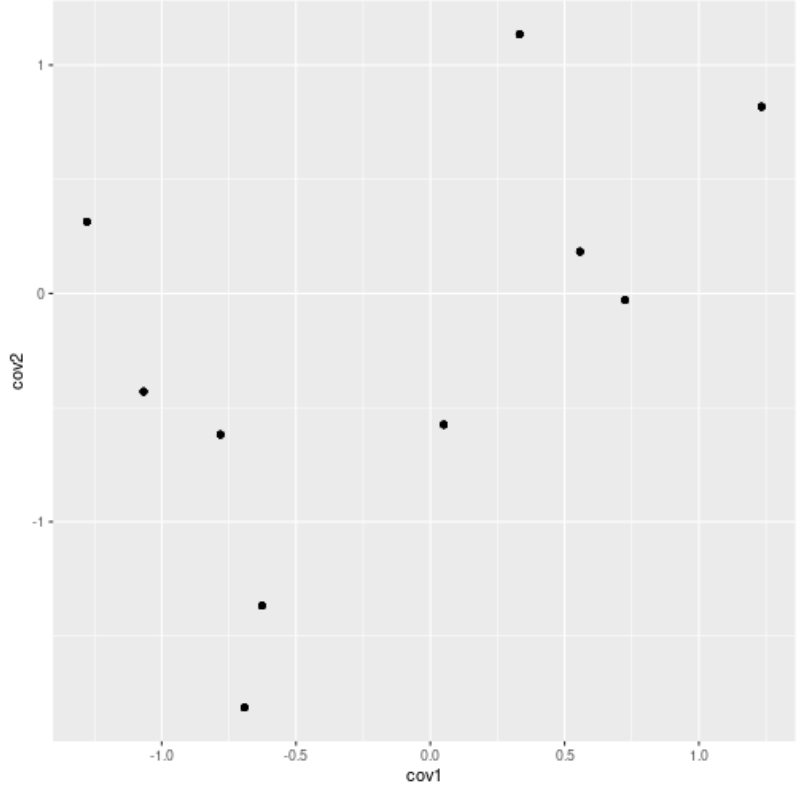


- Easy to integrate with workflow and pipelines
- Lack of %>% functions [3]
- Amazing way to combine different layers of plot components

[3] Hadley Wickam talk about "mistakes" in developing the tidyverse

GGplot2

```
dat %>%
  select(cov1, cov2) %>%
  ggplot(aes(x = cov1, y = cov2)) +
  geom_point()
```



Broom

- Manipulate **fitted models** and return **tidy data**
- Very useful for extracting information from multiple models in a easy way
- There are some expansions like `broom mixed` for `lme4` objects and `broomExtra` for also `brms` models

Broom::tidy() and Broom::glance()

```
fit <- lm(value ~ cond1 * cond2 + cov1 + cov2, data = dat)
tidy(fit) %>%  head()
```

```
## # A tibble: 6 x 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)  100.         3.59    27.8   1.33e-42
## 2 cond1B        -3.73       5.06    -0.738 4.63e- 1
## 3 cond1C        -3.83       5.06    -0.758 4.51e- 1
## 4 cond22        -4.82       5.06    -0.952 3.44e- 1
## 5 cond23         6.36       5.06     1.26  2.12e- 1
## 6 cov1           0.906      1.75     0.519 6.05e- 1
```

```
glance(fit)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic p.value    df logLik   AIC   BIC deviance
##       <dbl>         <dbl> <dbl>     <dbl>   <dbl> <dbl>  <dbl> <dbl> <dbl>    <dbl>
## 1     0.155        0.0477  11.3      1.45   0.176    10  -340.  704.  734.   10106.
## # … with 2 more variables: df.residual <int>, nobs <int>
```

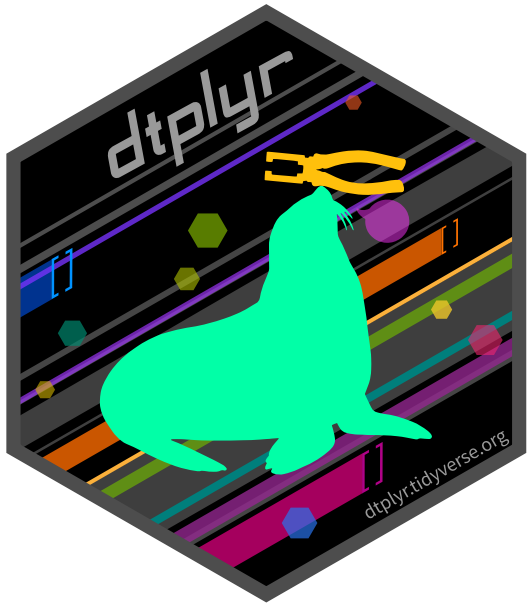
Other packages

Strings manipulation

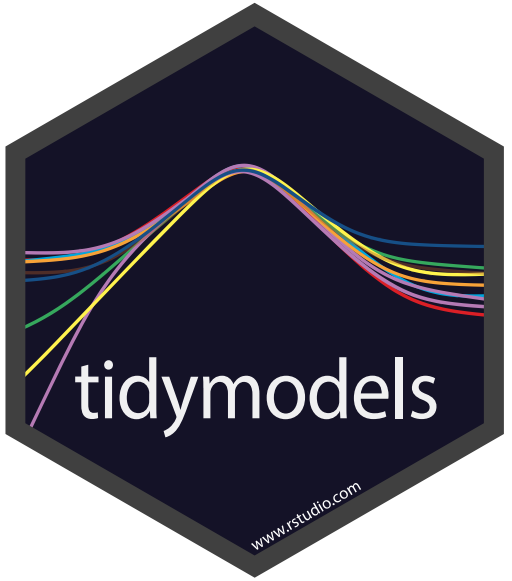
Work with dates

Other packages

Datatable power, dyplr code



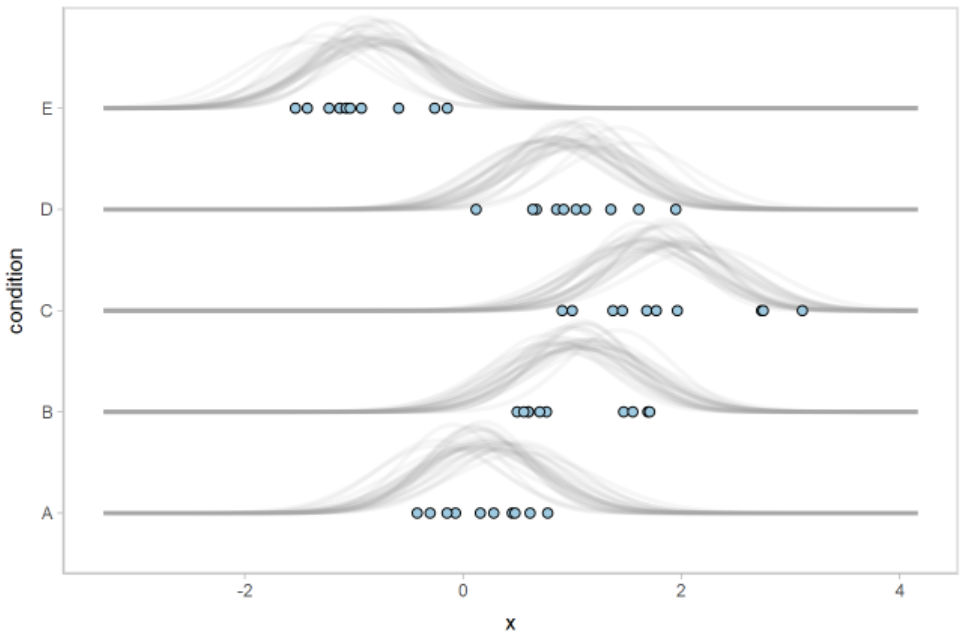
Modelling (e.g., broom)



Other packages

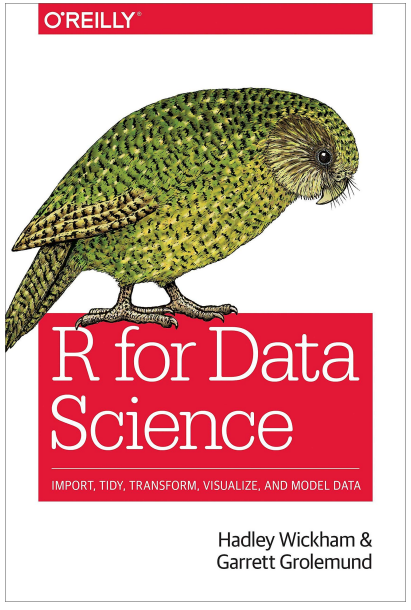
Tidybayes

- Some **geoms** for `ggplo2` and function to manage fitted **bayesian models**
- Support `rstanarm`, `brms`



Some Resources

R4DS - R for Data Science



- Best book for the **tidy** approach the **data science** in general

- Especially the many models chapter

R4DS Book

Talks

- David Robinson - Ten Tremendous Tricks in the Tidyverse
- David Robinson - Teach the Tidyverse to Beginners
- Hadley Wickham - Managing many models with R
- Hadley Wickham - Mistakes of the Tidyverse
- Hadley Wickham - Data visualization and data science
- Emily Robinson - The lesser known stars of the Tidyverse

filippo.gambarota@phd.unipd.it

Slides made with the Xaringan package by Yihui Xie