

(A bit of) Advanced R

Part 3 - a tour of the tidyverse

Julien Chiquet

<https://github.com/jchiquet/CourseAdvancedR>

Université Paris Dauphine, Juin 2018



Outline

- ① Introduction
- ② Structures and types: `tibble`, `forcats`, `stringr`
- ③ data wrangling: `readr`, `tidyr`, `dplyr`
- ④ Manipulation, vizualization: `magrittr`, `purrr`, `ggplot2`

References

Many ideas/examples inspired/stolen there:

R for data science (Wickham & Grolemund, 2016), <http://r4ds.had.co.nz>



Tidyverse website, <https://www.tidyverse.org/>



Prerequisites

Data Structures in base R

- ① Atomic vector (integer, double, logical, character)
- ② Recursive vector (list)
- ③ Factor
- ④ Matrix and array
- ⑤ Data Frame

R base programming

- ① Control Statements
- ② Functions
- ③ Functionals
- ④ Input/output
- ⑤ Rstudio API (application programming interface)

Outline

- 1 Introduction
- 2 Structures and types: `tibble`, `forcats`, `stringr`
- 3 data wrangling: `readr`, `tidyr`, `dplyr`
- 4 Manipulation, vizualization: `magrittr`, `purrr`, `ggplot2`

Tidy data: motivation

Collected data are (never) under a proper canonical format

“Happy families are all alike; every unhappy family is unhappy in its own way.” – Leo Tolstoy

“Tidy datasets are all alike, but every messy dataset is messy in its own way.” – Hadley Wickham¹

¹Rstudio's chief scientific advisor

Tidy data: motivation

Collected data are (never) under a proper canonical format

“Happy families are all alike; every unhappy family is unhappy in its own way.” – Leo Tolstoy

“Tidy datasets are all alike, but every messy dataset is messy in its own way.” – Hadley Wickham¹

¹Rstudio's chief scientific advisor

Tidy data: what?

First, a subjective question

What is the *observation/statistical unit* in your data?

Definition

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types.

In tidy data,

- ① each variable forms a column,
- ② each observation forms a row,
- ③ each type of observational unit forms a table.

Tidy data: what?

First, a subjective question

What is the *observation/statistical unit* in your data?

Definition

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types.

In tidy data,

- ① each variable forms a column,
- ② each observation forms a row,
- ③ each type of observational unit forms a table.

Tidy data: what?

First, a subjective question

What is the *observation/statistical unit* in your data?

Definition

Tidy data is a standard way of mapping the meaning of a dataset to its structure. A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types.

In tidy data,

- ① each variable forms a column,
- ② each observation forms a row,
- ③ each type of observational unit forms a table.

Tidy data: why?

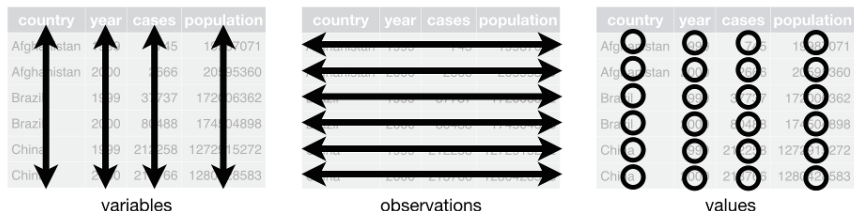


Figure 1: Tidy data

- make manipulation, visualization and modelling easier
- a common structure for all packages
- a philosophy for data representation (beyond the R framework)

Tidy or not ?

```
tidyr::table3
```

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>      <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

Tidy or not ?

```
tidyr::table2
```

```
## # A tibble: 12 x 4
##   country      year type      count
##   <chr>      <int> <chr>    <int>
## 1 Afghanistan 1999 cases      745
## 2 Afghanistan 1999 population 19987071
## 3 Afghanistan 2000 cases      2666
## 4 Afghanistan 2000 population 20595360
## 5 Brazil      1999 cases      37737
## 6 Brazil      1999 population 172006362
## 7 Brazil      2000 cases      80488
## 8 Brazil      2000 population 174504898
## 9 China       1999 cases      212258
## 10 China      1999 population 1272915272
## 11 China      2000 cases      213766
## 12 China      2000 population 1280428583
```

Tidy or not ?

```
tidyr::table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>      <int> <int>      <int>
## 1 Afghanistan 1999     745   19987071
## 2 Afghanistan 2000    2666  20595360
## 3 Brazil      1999   37737  172006362
## 4 Brazil      2000   80488  174504898
## 5 China       1999  212258 1272915272
## 6 China       2000  213766 1280428583
```

The process of data analysis

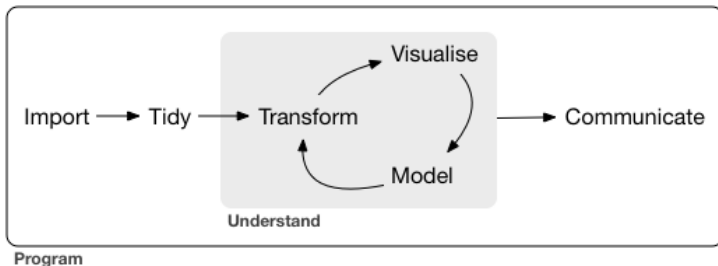


Figure 2: scheme for data analysis process

- **import:** read/load the data
- **tidy:** formating (individuals/variables data frame)
- **transform:** suppression/creation/filtering/selection
- **visualization:** representation and validation
- **model:** statistical fits
- **communication:** diffusion (web/talk/article)

The tidyverse

Definition

- contraction of 'tidy' ("well arranged) and 'universe'.
- an *opinionated collection* of R packages designed for data science.
- all packages share an underlying *design philosophy, grammar, and data structures*

Phylosophy

allows the user to focus on the important statistical questions rather than focusing on the technical aspects of data analysis

Let's have a look

The core tidyverse loads ggplot2, tibble, tidyr, readr, purrr, stringr, forcats, dplyr and others in a fancy and unconflicted way.

```
library(tidyverse)
tidyverse::tidyverse_conflicts()
```

```
## -- Conflicts -----
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
```

```
tidyverse::tidyverse_deps()
```

```
## # A tibble: 25 x 4
##   package cran  local behind
##   <chr>   <chr> <chr> <lgl>
## 1 broom   0.4.4 0.4.4 FALSE
## 2 cli     1.0.0 1.0.0 FALSE
## 3 crayon  1.3.4 1.3.4 FALSE
## 4 dbplyr  1.2.1 1.2.1 FALSE
## 5 dplyr   0.7.5 0.7.5 FALSE
## 6 forcats 0.3.0 0.3.0 FALSE
## 7 ggplot2 2.2.1 2.2.1 FALSE
## 8 haven   1.1.1 1.1.1 FALSE
## 9 hms     0.4.2 0.4.2 FALSE
## 10 httr    1.3.1 1.3.1 FALSE
## # ... with 15 more rows
```

Packages roles and overview: types



tibble

a modern re-imagining of the data frame



stringr

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



forcats

a suite of useful tools that solve common problems with factors

Packages roles and overview: wrangling



readr

a fast and friendly way to read rectangular data (like csv, tsv, and fwf)



tidyr

a set of functions that help you get to tidy data



dplyr

a consistent set of verbs that solve the most common data manipulation challenges

Packages roles and overview: manipulation



a system for declaratively creating graphics, based on The Grammar of Graphics



enhances R's functional programming (FP) toolkit



offers a set of operators which make your code more readable

Data analysis with the tidyverse

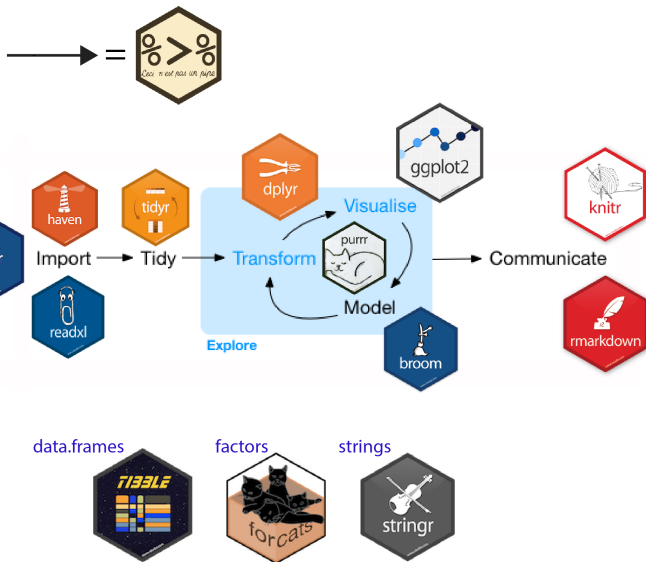


Figure 3: Updated scheme for data analysis process

Outline

- 1 Introduction
- 2 Structures and types: `tibble`, `forcats`, `stringr`
- 3 data wrangling: `readr`, `tidyr`, `dplyr`
- 4 Manipulation, vizualization: `magrittr`, `purrr`, `ggplot2`

{tibble}



Figure 4: a modern re-imagining of the data frame

tibble versus data.frame

tibbles (or `tbl_df`) are modern reimaging of the `data.frame`,

- *lazy*: do less (e.g. do not change variable names, types, no partial matching)
- *surly*: complain more (e.g. when a variable does not exist)

Conversion from a data.frame

```
head(iris)
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1           3.5           1.4           0.2   setosa
## 2           4.9           3.0           1.4           0.2   setosa
## 3           4.7           3.2           1.3           0.2   setosa
## 4           4.6           3.1           1.5           0.2   setosa
## 5           5.0           3.6           1.4           0.2   setosa
## 6           5.4           3.9           1.7           0.4   setosa
```

```
as_tibble(iris)
```

```
## # A tibble: 150 x 5
##       Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1           5.1           3.5           1.4           0.2 setosa
## 2           4.9           3           1.4           0.2 setosa
## 3           4.7           3.2           1.3           0.2 setosa
## 4           4.6           3.1           1.5           0.2 setosa
## 5           5           3.6           1.4           0.2 setosa
## 6           5.4           3.9           1.7           0.4 setosa
## 7           4.6           3.4           1.4           0.3 setosa
## 8           5           3.4           1.5           0.2 setosa
## 9           4.4           2.9           1.4           0.2 setosa
## 10          4.9           3.1           1.5           0.1 setosa
## # ... with 140 more rows
```


Conversion from a data.frame

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1         5.1         3.5         1.4         0.2   setosa
## 2         4.9         3.0         1.4         0.2   setosa
## 3         4.7         3.2         1.3         0.2   setosa
## 4         4.6         3.1         1.5         0.2   setosa
## 5         5.0         3.6         1.4         0.2   setosa
## 6         5.4         3.9         1.7         0.4   setosa
```

```
as_tibble(iris)
```

```
## # A tibble: 150 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##   <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1         5.1         3.5         1.4         0.2 setosa
## 2         4.9         3         1.4         0.2 setosa
## 3         4.7         3.2         1.3         0.2 setosa
## 4         4.6         3.1         1.5         0.2 setosa
## 5         5         3.6         1.4         0.2 setosa
## 6         5.4         3.9         1.7         0.4 setosa
## 7         4.6         3.4         1.4         0.3 setosa
## 8         5         3.4         1.5         0.2 setosa
## 9         4.4         2.9         1.4         0.2 setosa
## 10        4.9         3.1         1.5         0.1 setosa
## # ... with 140 more rows
```

Creating a tibble

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

```
## # A tibble: 5 x 3  
##       x     y     z  
##   <int> <dbl> <dbl>  
## 1     1     1     2  
## 2     2     1     5  
## 3     3     1    10  
## 4     4     1    17  
## 5     5     1    26
```

Column names of a tibble

Names can start by any character. To refer such variables, use the backticks

```
tibble(`:`) = "smile", ` ` = "space", `2000` = "number")
```

```
## # A tibble: 1 x 3  
##   `:` ` ` `2000`  
##   <chr> <chr> <chr>  
## 1 smile space number
```

Creating a tibble

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

```
## # A tibble: 5 x 3  
##       x     y     z  
##   <int> <dbl> <dbl>  
## 1     1     1     2  
## 2     2     1     5  
## 3     3     1    10  
## 4     4     1    17  
## 5     5     1    26
```

Column names of a tibble

Names can start by any character. To refer such variables, use the backticks

```
tibble(`:` = "smile", ` ` = "space", `2000` = "number")
```

```
## # A tibble: 1 x 3  
##   `:` ` ` `2000`  
##   <chr> <chr> <chr>  
## 1 smile space number
```

Row names

Row do not have names in a tibble

Solution

- one can use name by adding a specific column
- `rownames_to_column ()` can help

Example

```
as_tibble(swiss, rownames = "Province")
```

```
## # A tibble: 47 x 7
##   Province      Fertility Agriculture Examination Education Catholic
##   <chr>          <dbl>         <dbl>         <int>         <int>         <dbl>
## 1 Courtelary      80.2           17           15           12           9.96
## 2 Delemont        83.1           45.1          6            9           84.8
## 3 Franches-Mnt    92.5           39.7          5            5           93.4
## 4 Moutier         85.8           36.5          12           7           33.8
## 5 Neuveville      76.9           43.5          17           15           5.16
## 6 Porrentruy      76.1           35.3          9            7           90.6
## 7 Broye           83.8           70.2          16           7           92.8
## 8 Glane           92.4           67.8          14           8           97.2
## 9 Gruyere         82.4           53.3          12           7           97.7
## 10 Sarine         82.9           45.2          16           13           91.4
## # ... with 37 more rows, and 1 more variable: Infant.Mortality <dbl>
```

Consistency in subsetting

```
df <- data.frame(x = 1:9, y = LETTERS[1:9])  
tbl <- tibble(x = 1:9, y = LETTERS[1:9])
```

```
class(df[, 1:2])
```

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

```
class(tbl[, 1:2])
```

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

```
class(df[, 1])
```

```
## [1] "integer"
```

```
class(tbl[, 1])
```

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

Consistency in subsetting

```
df <- data.frame(x = 1:9, y = LETTERS[1:9])  
tbl <- tibble(x = 1:9, y = LETTERS[1:9])
```

```
class(df[, 1:2])
```

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

```
class(tbl[, 1:2])
```

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

```
class(df[, 1])
```

```
## [1] "integer"
```

```
class(tbl[, 1])
```

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

Consistency in subsetting

```
df <- data.frame(x = 1:9, y = LETTERS[1:9])  
tbl <- tibble(x = 1:9, y = LETTERS[1:9])
```

```
class(df[, 1:2])
```

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

```
class(tbl[, 1:2])
```

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

```
class(df[, 1])
```

```
## [1] "integer"
```

```
class(tbl[, 1])
```

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

List-column

The type `list` is available for a column in `tibble`

- a `tibble` allows cells containing lists
- a `tibble` allows cells containing data frames.

```
subset(starwars, select = c('name', 'height', 'mass', 'hair_color', 'films', 'vehicles'))
```

```
## # A tibble: 87 x 6
##   name          height mass hair_color    films    vehicles
##   <chr>         <int> <dbl> <chr>    <list>   <list>
## 1 Luke Skywalker    172    77 blond   <chr [5]> <chr [2]>
## 2 C-3PO             167    75 <NA>    <chr [6]> <chr [0]>
## 3 R2-D2              96    32 <NA>    <chr [7]> <chr [0]>
## 4 Darth Vader       202   136 none    <chr [4]> <chr [0]>
## 5 Leia Organa       150    49 brown    <chr [5]> <chr [1]>
## 6 Owen Lars         178   120 brown, grey <chr [3]> <chr [0]>
## 7 Beru Whitesun lars 165    75 brown    <chr [3]> <chr [0]>
## 8 R5-D4              97    32 <NA>    <chr [1]> <chr [0]>
## 9 Biggs Darklighter 183    84 black    <chr [1]> <chr [0]>
## 10 Obi-Wan Kenobi    182    77 auburn, white <chr [6]> <chr [1]>
## # ... with 77 more rows
```


List-column: put a vector in each case

```
head(starwars$films, 4)
```

```
## [[1]]  
## [1] "Revenge of the Sith"      "Return of the Jedi"  
## [3] "The Empire Strikes Back" "A New Hope"  
## [5] "The Force Awakens"  
##  
## [[2]]  
## [1] "Attack of the Clones"    "The Phantom Menace"  
## [3] "Revenge of the Sith"    "Return of the Jedi"  
## [5] "The Empire Strikes Back" "A New Hope"  
##  
## [[3]]  
## [1] "Attack of the Clones"    "The Phantom Menace"  
## [3] "Revenge of the Sith"    "Return of the Jedi"  
## [5] "The Empire Strikes Back" "A New Hope"  
## [7] "The Force Awakens"  
##  
## [[4]]  
## [1] "Revenge of the Sith"    "Return of the Jedi"  
## [3] "The Empire Strikes Back" "A New Hope"
```

{forcats}



Figure 5: a suite of useful tools that solve common problems with factor

forcats versus base factors

- easy use in conjunction with other tidyverse packages
- correct inconsistent behaviours of R base factors facilities

{stringr}



Figure 6: cohesive set of functions designed to make working with strings as easy as possible

stringr versus base string utilities

String manipulation is cumbersome in R base. However, string plays a big role in many data cleaning and preparation.

- easy use in conjunction with other tidyverse packages
- faster and correct implementations of common string manipulations

Outline

- 1 Introduction
- 2 Structures and types: `tibble`, `forcats`, `stringr`
- 3 data wrangling: `readr`, `tidyr`, `dplyr`
- 4 Manipulation, vizualization: `magrittr`, `purrr`, `ggplot2`

readr



Figure 7: a fast and friendly way to read rectangular data (like csv, tsv, and fwf)

{tidyr}

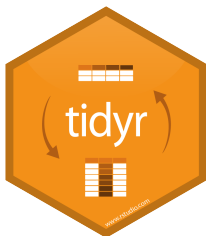


Figure 8: a set of functions that help you get to tidy data

```
library(tidyr)
```

⇒ tidyr is a package which helps you to transform messy datasets into tidy datasets.

- evolution of base function reshape
- available functions are spread, gather, unite, separate

Grades dataset

```
grades <- tibble(  
  Name = c("Tommy", "Mary", "Gary", "Cathy"),  
  Sexage = c("m.15", "f.15", "m.16", "f.14"),  
  Test1 = c(10, 15, 16, 14),  
  Test2 = c(11, 13, 10, 12),  
  Test3 = c(12, 13, 17, 10)  
)  
grades
```

```
## # A tibble: 4 x 5  
##   Name Sexage Test1 Test2 Test3  
##   <chr> <chr> <dbl> <dbl> <dbl>  
## 1 Tommy m.15      10      11      12  
## 2 Mary  f.15      15      13      13  
## 3 Gary  m.16      16      10      17  
## 4 Cathy f.14      14      12      10
```

Name	Sexage	Test1	Test2	Test3
Tommy	m.15	10	11	12
Mary	f.15	15	13	13
Gary	m.16	16	10	17
Cathy	f.14	14	12	10

separate()

Separate one column into multiple columns

```
grades <- separate(grades, Sexage, into = c("Sex", "Age"))
grades
```

```
## # A tibble: 4 x 6
##   Name Sex Age Test1 Test2 Test3
##   <chr> <chr> <chr> <dbl> <dbl> <dbl>
## 1 Tommy m    15      10      11      12
## 2 Mary f    15      15      13      13
## 3 Gary m    16      16      10      17
## 4 Cathy f    14      14      12      10
```

Name	Sex	Age	Test1	Test2	Test3
Tommy	m	15	10	11	12
Mary	f	15	15	13	13
Gary	m	16	16	10	17
Cathy	f	14	14	12	10

Remark

The inverse of `separate()` is `unite()`

separate()

Separate one column into multiple columns

```
grades <- separate(grades, Sexage, into = c("Sex", "Age"))
grades
```

```
## # A tibble: 4 x 6
##   Name Sex Age Test1 Test2 Test3
##   <chr> <chr> <chr> <dbl> <dbl> <dbl>
## 1 Tommy m    15      10      11      12
## 2 Mary f    15      15      13      13
## 3 Gary m    16      16      10      17
## 4 Cathy f    14      14      12      10
```

Name	Sex	Age	Test1	Test2	Test3
Tommy	m	15	10	11	12
Mary	f	15	15	13	13
Gary	m	16	16	10	17
Cathy	f	14	14	12	10

Remark

The inverse of `separate()` is `unite()`

gather()

Gather Columns Into Key-Value Pairs

```
grades <- gather(grades, Test1, Test2, Test3, key = Test, value = Grade)
head(grades)
```

```
## # A tibble: 6 x 5
##   Name Sex Age Test Grade
##   <chr> <chr> <chr> <chr> <dbl>
## 1 Tommy m 15 Test1 10
## 2 Mary f 15 Test1 15
## 3 Gary m 16 Test1 16
## 4 Cathy f 14 Test1 14
## 5 Tommy m 15 Test2 11
## 6 Mary f 15 Test2 13
```

Name	Sex	Age	Test	Grade
Tommy	m	15	Test1	10
Mary	f	15	Test1	15
Gary	m	16	Test1	16
Cathy	f	14	Test1	14
Tommy	m	15	Test2	11
Mary	f	15	Test2	13

Remark

The inverse of `gather()` is `spread()`

gather()

Gather Columns Into Key-Value Pairs

```
grades <- gather(grades, Test1, Test2, Test3, key = Test, value = Grade)
head(grades)
```

```
## # A tibble: 6 x 5
##   Name Sex Age Test Grade
##   <chr> <chr> <chr> <chr> <dbl>
## 1 Tommy m 15 Test1 10
## 2 Mary f 15 Test1 15
## 3 Gary m 16 Test1 16
## 4 Cathy f 14 Test1 14
## 5 Tommy m 15 Test2 11
## 6 Mary f 15 Test2 13
```

Name	Sex	Age	Test	Grade
Tommy	m	15	Test1	10
Mary	f	15	Test1	15
Gary	m	16	Test1	16
Cathy	f	14	Test1	14
Tommy	m	15	Test2	11
Mary	f	15	Test2	13

Remark

The inverse of `gather()` is `spread()`

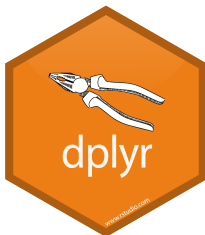


Figure 9: a consistent set of verbs (*a grammar*) that solves the most common data manipulation challenges

Typical operations

- create and pick variables
- pick and reorder observations
- create summaries
- ...

→ Functions in this package are verbs and work similarly

mtcars dataset

```
data(mtcars)
as_tibble(mtcars)
```

```
## # A tibble: 32 x 11
##   mpg   cyl  disp    hp  drat    wt  qsec    vs  am  gear  carb
## * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21     6  160   110  3.9   2.62  16.5    0    1     4     4
## 2  21     6  160   110  3.9   2.88  17.0    0    1     4     4
## 3  22.8   4  108    93  3.85  2.32  18.6    1    1     4     1
## 4  21.4   6  258   110  3.08  3.22  19.4    1    0     3     1
## 5  18.7   8  360   175  3.15  3.44  17.0    0    0     3     2
## 6  18.1   6  225   105  2.76  3.46  20.2    1    0     3     1
## 7  14.3   8  360   245  3.21  3.57  15.8    0    0     3     4
## 8  24.4   4  147.    62  3.69  3.19  20      1    0     4     2
## 9  22.8   4  141.    95  3.92  3.15  22.9    1    0     4     2
## 10 19.2   6  168.   123  3.92  3.44  18.3    1    0     4     4
## # ... with 22 more rows
```

Select rows with `filter()`

Arguments

- 1 data
- 2 filtering expressions

Output

- a tibble
- **do not modify** the original data

Example

```
filter(mtcars, cyl == 4, mpg > 30)
```

```
##      mpg cyl disp  hp drat   wt  qsec vs am gear carb
## 1 32.4   4  78.7  66 4.08 2.200 19.47  1  1   4     1
## 2 30.4   4  75.7  52 4.93 1.615 18.52  1  1   4     2
## 3 33.9   4  71.1  65 4.22 1.835 19.90  1  1   4     1
## 4 30.4   4  95.1 113 3.77 1.513 16.90  1  1   5     2
```

Reorder rows with arrange()

Principle

works like `filter()` but reorder rows according to a series of conditions

Example

```
as_tibble(arrange(mtcars, desc(carb), mpg))
```

```
## # A tibble: 32 x 11
##   mpg   cyl  disp    hp  drat    wt   qsec    vs  am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  15     8   301   335  3.54  3.57  14.6     0    1     5     8
## 2  19.7    6   145   175  3.62  2.77  15.5     0    1     5     6
## 3  10.4    8   472   205  2.93  5.25  18.0     0    0     3     4
## 4  10.4    8   460   215    3   5.42  17.8     0    0     3     4
## 5  13.3    8   350   245  3.73  3.84  15.4     0    0     3     4
## 6  14.3    8   360   245  3.21  3.57  15.8     0    0     3     4
## 7  14.7    8   440   230  3.23  5.34  17.4     0    0     3     4
## 8  15.8    8   351   264  4.22  3.17  14.5     0    1     5     4
## 9  17.8    6   168.   123  3.92  3.44  18.9     1    0     4     4
## 10 19.2    6   168.   123  3.92  3.44  18.3     1    0     4     4
## # ... with 22 more rows
```

Selecting columns with `select()` I

Similar to `base::subset(, select = c("", ""))`

With names

can be quoted or unquoted

```
as_tibble(select(mtcars, mpg, 'wt', cyl))
```

```
## # A tibble: 32 x 3
##   mpg    wt    cyl
##   * <dbl> <dbl> <dbl>
## 1  21    2.62     6
## 2  21    2.88     6
## 3  22.8  2.32     4
## 4  21.4  3.22     6
## 5  18.7  3.44     8
## 6  18.1  3.46     6
## 7  14.3  3.57     8
## 8  24.4  3.19     4
## 9  22.8  3.15     4
## 10 19.2  3.44     6
## # ... with 22 more rows
```


Selecting columns with `select()` II

With indexes

```
as_tibble(select(mtcars, 1,2,5:7))
```

```
## # A tibble: 32 x 5
##   mpg   cyl  drat   wt  qsec
##   * <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21     6  3.9   2.62  16.5
## 2  21     6  3.9   2.88  17.0
## 3  22.8   4  3.85  2.32  18.6
## 4  21.4   6  3.08  3.22  19.4
## 5  18.7   8  3.15  3.44  17.0
## 6  18.1   6  2.76  3.46  20.2
## 7  14.3   8  3.21  3.57  15.8
## 8  24.4   4  3.69  3.19   20
## 9  22.8   4  3.92  3.15  22.9
## 10 19.2   6  3.92  3.44  18.3
## # ... with 22 more rows
```

Renaming columns with `rename()`

`rename()` keeps all variables

```
as_tibble(rename(iris, petal_length = Petal.Length))
```

```
## # A tibble: 150 x 5
##   Sepal.Length Sepal.Width petal_length Petal.Width Species
##   <dbl>         <dbl>         <dbl>         <dbl> <fct>
## 1         5.1         3.5         1.4         0.2 setosa
## 2         4.9         3         1.4         0.2 setosa
## 3         4.7         3.2         1.3         0.2 setosa
## 4         4.6         3.1         1.5         0.2 setosa
## 5         5         3.6         1.4         0.2 setosa
## 6         5.4         3.9         1.7         0.4 setosa
## 7         4.6         3.4         1.4         0.3 setosa
## 8         5         3.4         1.5         0.2 setosa
## 9         4.4         2.9         1.4         0.2 setosa
## 10        4.9         3.1         1.5         0.1 setosa
## # ... with 140 more rows
```

Renaming columns with select()

Renaming can be done with select()

select() only keeps the variables specified

```
as_tibble(select(iris, petal_length = Petal.Length))
```

```
## # A tibble: 150 x 1
##   petal_length
##   <dbl>
## 1         1.4
## 2         1.4
## 3         1.3
## 4         1.5
## 5         1.4
## 6         1.7
## 7         1.4
## 8         1.5
## 9         1.4
## 10        1.5
## # ... with 140 more rows
```

Add new variables with mutate()

mutate keeps the existing variables

```
as_tibble(  
  mutate(mtcars,  
    cyl2 = 2 * cyl,  
    cyl4 = 2 * cyl2,  
    disp = disp * 0.0163871,  
    drat = NULL)  
)
```

```
## # A tibble: 32 x 12  
##       mpg   cyl  disp    hp  wt  qsec    vs  am  gear  carb  cyl2  cyl4  
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1  21     6   2.62  110  2.62  16.5    0    1     4     4    12    24  
## 2  21     6   2.62  110  2.88  17.0    0    1     4     4    12    24  
## 3 22.8     4   1.77   93  2.32  18.6    1    1     4     1     8    16  
## 4 21.4     6   4.23  110  3.22  19.4    1    0     3     1    12    24  
## 5 18.7     8   5.90  175  3.44  17.0    0    0     3     2    16    32  
## 6 18.1     6   3.69  105  3.46  20.2    1    0     3     1    12    24  
## 7 14.3     8   5.90  245  3.57  15.8    0    0     3     4    16    32  
## 8 24.4     4   2.40   62  3.19   20     1    0     4     2     8    16  
## 9 22.8     4   2.31   95  3.15  22.9    1    0     4     2     8    16  
## 10 19.2     6   2.75  123  3.44  18.3    1    0     4     4    12    24  
## # ... with 22 more rows
```

Add new variables with transmute()

transmute drops the existing variables

```
as_tibble(  
  transmute(mtcars,  
    cyl2 = 2 * cyl,  
    cyl4 = 2 * cyl2,  
    disp = disp * 0.0163871,  
    drat = NULL)  
)
```

```
## # A tibble: 32 x 3  
##   cyl2  cyl4  disp  
##   <dbl> <dbl> <dbl>  
## 1    12    24  2.62  
## 2    12    24  2.62  
## 3     8    16  1.77  
## 4    12    24  4.23  
## 5    16    32  5.90  
## 6    12    24  3.69  
## 7    16    32  5.90  
## 8     8    16  2.40  
## 9     8    16  2.31  
## 10   12    24  2.75  
## # ... with 22 more rows
```

Create summary statistics with summarise()

Reduction is done by means of statistical functions

- Center: `mean()`, `median()`
- Spread: `sd()`, `IQR()`, `mad()`
- Range: `min()`, `max()`, `quantile()`
- Position: `first()`, `last()`, `nth()`,
- Count: `n()`, `n_distinct()`
- Logical: `any()`, `all()`

Example

```
summarise(mtcars, Mean_mpg = mean(mpg), Var_disp = var(displ))
```

```
##   Mean_mpg Var_disp  
## 1  20.09062 15360.8
```

group rows according to factors with group_by()

group_by() does not do much visible expect creating a grouped data frame with type grouped_df

```
group_by(mtcars, cyl, am)
```

```
## # A tibble: 32 x 11
## # Groups:   cyl, am [6]
##      mpg   cyl  disp    hp  drat    wt   qsec    vs  am  gear  carb
## * <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21       6  160   110   3.9   2.62  16.5     0    1     4     4
## 2  21       6  160   110   3.9   2.88  17.0     0    1     4     4
## 3  22.8     4  108    93   3.85   2.32  18.6     1    1     4     1
## 4  21.4     6  258   110   3.08   3.22  19.4     1    0     3     1
## 5  18.7     8  360   175   3.15   3.44  17.0     0    0     3     2
## 6  18.1     6  225   105   2.76   3.46  20.2     1    0     3     1
## 7  14.3     8  360   245   3.21   3.57  15.8     0    0     3     4
## 8  24.4     4  147.    62   3.69   3.19  20       1    0     4     2
## 9  22.8     4  141.    95   3.92   3.15  22.9     1    0     4     2
## 10 19.2     6  168.   123   3.92   3.44  18.3     1    0     4     4
## # ... with 22 more rows
```

Combine summarise() and group_by()

Magic of group_by() comes true when used in conjunction with summarise()

```
grp_mtcars <- group_by(mtcars, cyl, carb)
summarise(grp_mtcars, Count = n(), Mean_mpg = mean(mpg), Var_disp = var(dis))
```

```
## # A tibble: 9 x 5
## # Groups:   cyl [?]
##   cyl carb Count Mean_mpg Var_disp
##   <dbl> <dbl> <int>   <dbl>   <dbl>
## 1     4     1     5    27.6    457.
## 2     4     2     6    25.9    732.
## 3     6     1     2    19.8    544.
## 4     6     4     4    19.8    19.3
## 5     6     6     1    19.7     NA
## 6     8     2     4    17.2   1886.
## 7     8     3     3    16.3     0
## 8     8     4     6    13.2   3341.
## 9     8     8     1    15      NA
```


Common remarks and extension

Remarks

Tout comme les autres primitives de `dyp1r mutate()` ne modifie pas le tableau d'origine

Outline

- 1 Introduction
- 2 Structures and types: `tibble`, `forcats`, `stringr`
- 3 data wrangling: `readr`, `tidyr`, `dplyr`
- 4 Manipulation, vizualization: `magrittr`, `purrr`, `ggplot2`

{magrittr}



Figure 10: a set of operators which make your code more readable

```
library(magrittr)
```

Provides the following operators

- Pipe %>%
- Reassignment pipe %<>%
- T-Pipe %T>%

Motivation: make Tom eat an apple

Everyday language

Tom eats an apple

Subject - Verb - Complement

Programming language

eat(Tom, apple)

Verb - Subject - Complement

Pipes

- ~> get closer to everyday language in your code
- ~> clearly expressing a sequence of multiple operations

Pipe %>%

- when you read code, %>% is pronounced “then”
- the keyboard shortcut for %>% is Ctrl + shift + M

Objective

- Helps writing R code which is easy to read (and thus, easy to understand)
- `x %>% f()` is equivalent to `f(x)`
- `x %>% f(y)` is equivalent to `f(x, y)`
- `x %>% f(y, .)` is equivalent to `f(y, x)`

Example

```
2^mean(log(seq_len(10), base = 2), na.rm = TRUE)
```

```
## [1] 4.528729
```

```
10 %>%  
  seq_len() %>%  
  log(base = 2) %>%  
  mean(na.rm = TRUE) %>%  
  {2^.}
```

```
## [1] 4.528729
```

Pipe %>%

- when you read code, %>% is pronounced “then”
- the keyboard shortcut for %>% is Ctrl + shift + M

Objective

- Helps writing R code which is easy to read (and thus, easy to understand)
- `x %>% f()` is equivalent to `f(x)`
- `x %>% f(y)` is equivalent to `f(x, y)`
- `x %>% f(y, .)` is equivalent to `f(y, x)`

Example

```
2^mean(log(seq_len(10), base = 2), na.rm = TRUE)
```

```
## [1] 4.528729
```

```
10 %>%  
  seq_len() %>%  
  log(base = 2) %>%  
  mean(na.rm = TRUE) %>%  
  {2^.}
```

```
## [1] 4.528729
```

Pipe %>%

- when you read code, %>% is pronounced “then”
- the keyboard shortcut for %>% is Ctrl + shift + M

Objective

- Helps writing R code which is easy to read (and thus, easy to understand)
- `x %>% f()` is equivalent to `f(x)`
- `x %>% f(y)` is equivalent to `f(x, y)`
- `x %>% f(y, .)` is equivalent to `f(y, x)`

Example

```
2^mean(log(seq_len(10), base = 2), na.rm = TRUE)
```

```
## [1] 4.528729
```

```
10 %>%  
  seq_len() %>%  
  log(base = 2) %>%  
  mean(na.rm = TRUE) %>%  
  {2^.}
```

```
## [1] 4.528729
```

Exercise

Consider

```
x <- c(0.109, 0.359, 0.63, 0.996, 0.515, 0.142, 0.017, 0.829, 0.907)
```

Compute the logarithm of `x`, return suitably lagged and iterated differences, compute the exponential function and round the result

- 1 In base R
- 2 Using `%>%`

(Re)assignment pipe %<>%

For affectation, `magrittr` provides the operator `%<>%` which allows to replace code like

```
mtcars <- mtcars%>% transform(cyl = cyl * 2)
```

by

```
mtcars %<>% transform(cyl = cyl * 2)
```

T-pipe %T>%

Problem with functions requiring early side effects along succession of %>%

- you might want to plot or print an object
- such function do not send back anything and break the pipe

Solution

- to overcome such an issue, use the “tee” pipe %T>%
- works like %>% except that it sends left side in place of right side of the expression
- “tee” because it looks like a pipe with a T shape

T-pipe %T>%: example without T

```
rnorm(100) %>%  
  matrix(ncol = 2) %>%  
  plot() %>%  
  str()
```

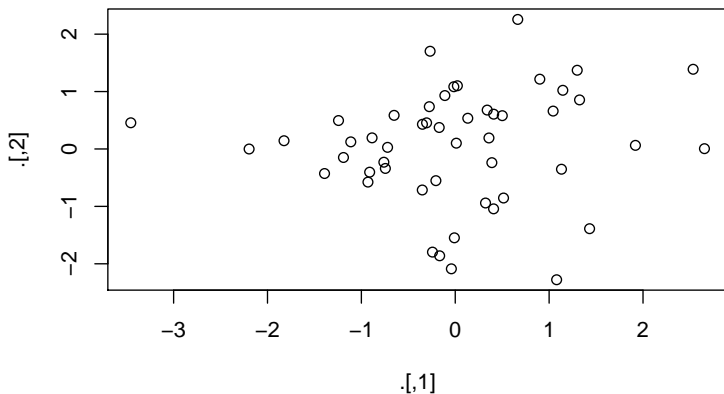


Figure 11: plot of bivariate Gaussian sample

T-pipe %T>%: example with T

```
rnorm(100) %>%  
  matrix(ncol = 2) %T>%  
  plot() %>%  
  str()
```

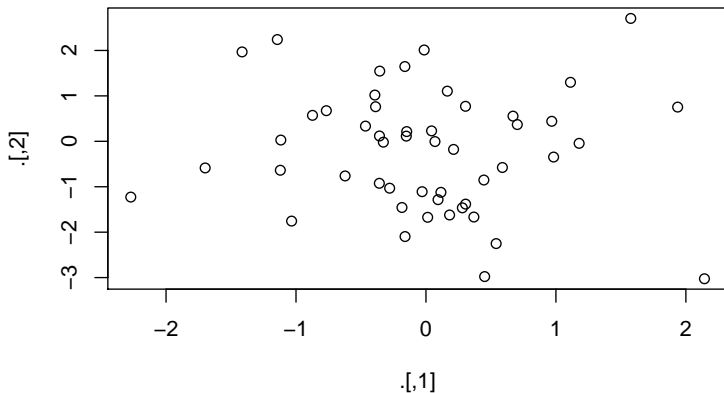


Figure 12: plot of bivariate Gaussian sample

```
## num [1:50, 1:2] -0.6224 0.3687 0.0131 0.28 -0.1483 ...
```

Exposition Operator %\$%

When working with functions that do not take data argument but still useful in a pipeline, e.g., when your data is first processed and then passed into the function.

Example

```
iris %>%  
  subset(Sepal.Length > mean(Sepal.Length)) %$%  
  cor(Sepal.Length, Sepal.Width)  
  
## [1] 0.3361992
```

When not to use the pipe

Consider other solutions when

Pipes contain too many steps

Create *intermediate* objects with meaningful names

Multiple inputs or outputs are required

E.g., when several objects need to *combine* together

Complex dependance structures exists between your entries

Pipes are fundamentally *linear*: expressing complex relationships with them yield confusing code.

{purrr}



Figure 13: enhances R's functional programming (FP) toolkit

ggplot2

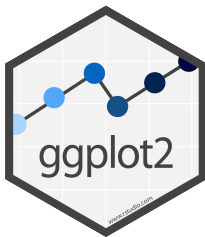


Figure 14: a system for declaratively creating graphics, based on The Grammar of Graphics

References

- R Core Team. (2017). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from <https://www.R-project.org/>
- Wickham, H. (2014). *Advanced r*. CRC Press. Retrieved from <http://adv-r.had.co.nz/>
- Wickham, H., & Golemund, G. (2016). *R for data science: Import, tidy, transform, visualize, and model data*. " O'Reilly Media, Inc." Retrieved from <http://r4ds.had.co.nz>