R for Data Science Notes

Andrea Melloncelli

2022-03-14

Contents

Chapter 1

Preface

1.1 These notes

These notes were written by Andrea Melloncelli as R for Data Science course notes.

1.2 Suggested readings

1.2.1 R Foundations

- R for Data Science
- RMarkdown (Book introduction: https://rmarkdown.rstudio.com/docs/articles/rmarkdown.html)

1.2.2 Non technical readings

- Introduction to Data Viz: https://clauswilke.com/dataviz/

1.2.3 Cookbooks

- R Cookbook
- Cheatsheets

1.2.4 Advanced R (technical readings)

- Efficient R
- Advanced R
- R Packages Manual
- Introduction to Data Science
- Bookdown

- Mastering Shiny (Wickham, 2020)

Chapter 2

Vector and matrix Operators

2.1 Create vectors

A vector is the concatenation (function `c()`) of some scalars (i.e. in this case integers, but they could be real numbers (`numeric`), Boolean values (`bool`) or even strings of characters (`character`)). Notice that each element of a vector has the same type (and class) of the other elements: we call this the type (and class) of a vector.

Let us create a couple of vectors:

```
v_1 <- c(2, 3, 5)
v_2 <- c(10, 4, 2)
```

2.2 Formalism

It is useful thinking about a pure function as a mathematical function.

The Cartesian product is useful to define function domain and co-domain. The Cartesias product is defined as:

$$A \times B = \{(a, b) \mid a \in A \text{ and } b \in B\}$$

A a *binary function* that takes a vector (of n elements of T) and a scalar and returns a vector is formalized in this way:

$$T^n \times T \to T^n$$

Where T is the time of the data structure (i.e.: `numeric`, `integer`, `character`, `bool`, etc…)

A matrix of n rows and m columns is formalized as:

$$T^{n,m}$$

2.3 Base vector operators

2.3.1 vector-scalar-to-vector operators

$$R^n \times R \to R^n$$

Every time a binary vector operator deals with a vector and a scalar (an R vector of length 1) that operation is performed between the scalar and each element of a vector, therefore it returns a vector.

```
v_1 + 3
```

```
## [1] 5 6 8
```

```
v_1 * 5
```

```
## [1] 10 15 25
```

```
v_1 - 3
```

```
## [1] -1  0  2
```

```
v_1 / 3
```

```
## [1] 0.6666667 1.0000000 1.6666667
```

```
v_1 ^ 3
```

```
## [1]   8  27 125
```

2.3.2 vector-vector-to-vector operators

$$R^n \times R^n \to R^n$$

```
# R^n X R^n -> R^n
v_1 + v_2
```

```
## [1] 12  7  7
```

```
v_1 * v_2
```

```
## [1] 20 12 10
```

```
v_1 - v_2
```

```
## [1] -8 -1  3
```

```
v_1 / v_2
```

```
## [1] 0.20 0.75 2.50
```

```
v_1 ^ v_2
```

```
## [1] 1024   81   25
```

Every time a binary vector operator deals with a vector and another vector of the same length that operation is performed element-wise: between an element of the first vector and the element of the other vector in the corresponding position.

2.4 Vector and Matrix operators

2.4.1 Scalar product

$$R^n \times R^n \to R$$

```
v_1 %*% v_2
```

```
##      [,1]
## [1,]   42
```

This is the sum of the element wise product.

2.4.2 Matrix product

$$R^{n,m} \times R^{m,o} \to R^{n,o}$$

```
mat <- matrix(1:12, ncol = 4)
mat
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

```
mat_2  <- matrix(1:20, nrow = 4)
mat_2
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9   13   17
## [2,]    2    6   10   14   18
## [3,]    3    7   11   15   19
## [4,]    4    8   12   16   20
```

```
mat %*% mat_2
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   70  158  246  334  422
## [2,]   80  184  288  392  496
## [3,]   90  210  330  450  570
```

2.5 Vectors

2.5.1 Subset by indices

Using the `[]` operator with a integer scalar or a vector it is possible to subset a vector. The length of the output is the same length of the vector of indexes.

```
# create a example vector: vector product to scalar (see above)
v_1 <- 1:20 * 10

# subset and take the 15th element only
v_1[15]
```

```
## [1] 150
```

```
# subset and take the 5th and 13th elements (a vector)
v_1[c(5,13)]
```

```
## [1]  50 130
```

```
# subset and take all elements from the 6th one to the 10th one inclusive (a vector)
v_1[6:10]
```

```
## [1]  60  70  80  90 100
```

2.5.2 Subset by logical

Using the `[]` operator with a Boolean scalar or a vector it is possible to subset a vector and take only the position where the value is true. This vector can be the result of a logical condition (element-wise condition). The length of the logical vector must be the same of the vector to be subset.

```
# create a example vector: vector product to scalar (see above)
v_2 <- 1:3 * 10

# logical subset: take true positions (`T`)
v_2[c(T,T,F)]
```

```
## [1] 10 20
```

```
# condition subset
v_2[v_2 < 25]
```

```
## [1] 10 20
```

2.6 Lists

2.6.1 create a list

A list is an ordered sequence of objects. Those objects can have a different type. In this example we create a list of three elements with names (giving names is possible also for vectors).

```
ls_1 <- list(a = 1, b = 2, c = 3)
```

2.6.2 Subset by indices

```
# subset
ls_1[1:2]
```

```
## $a
## [1] 1
##
## $b
## [1] 2
```

```
ls_1[2]
```

```
## $b
## [1] 2
```

2.6.3 Subset by logical

```
# logical subset
ls_1[c(T,T,F)]
```

```
## $a
## [1] 1
##
## $b
## [1] 2
```

Notice that a subset of a list is a list even if a single element is extracted.

2.6.4 Extract a single element

Using the `[[` operator extracting an element out of the list is possible: it returns the element itself not wrapped in a list.

```
# Extract the single element
ls_1[[2]]
```

```
## [1] 2
```

```
# Extract and use the element
mean(ls_1[[1]])
```

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

2.7 Statements

2.7.1 if

```
a <- 2

if (a < 3) {
  message("'a' is less than 3")
}
```

```
## 'a' is less than 3
```

2.7.2 else

```
a <- 4
if (a < 3) {
  message("'a' is less than 3")
} else {
  message("'a' is not less than 3")
}
```

```
## 'a' is not less than 3
```

2.7.3 Else-if chain

```
a <- 4
if (a < 3) {
  message("'a' is less than 3")
} else if (a > 5) {
  message("'a' is greater than 5")
} else if (a >= 3 & a <= 5) {
  message("'a' is between 3 and 4 inclusive")
} else {
  message("'a' is not a real number")
}
```

```
## 'a' is between 3 and 4 inclusive
```

2.8 For loop

Loop on a vector or a list of elements:

```
for (element in c(1, 3, 6)) {
  message("element: ", element)
}
```

```
## element: 1
```

```
## element: 3
```

```
## element: 6
```

```
for (element in list(1, 3, 6)) {
  message("element: ", element)
}
```

```
## element: 1
```

```
## element: 3
```

```
## element: 6
```

Loop over a vector indexes:

```
vec <- 1:10 * 10

for (i in seq_along(vec)) {
  message("step index i: ", i, ",   vec[i]: ", vec[i])
}
```

```
## step index i: 1,   vec[i]: 10
```

```
## step index i: 2,   vec[i]: 20
```

```
## step index i: 3,   vec[i]: 30
```

```
## step index i: 4,   vec[i]: 40
```

```
## step index i: 5,   vec[i]: 50
```

```
## step index i: 6,   vec[i]: 60
```

```
## step index i: 7,   vec[i]: 70
```

```
## step index i: 8,   vec[i]: 80
```

```
## step index i: 9,   vec[i]: 90
```

```
## step index i: 10,   vec[i]: 100
```

Example: count how many numbers divisible by 3 in `vec`:

```
count_even <- 0
for (element in vec) {
  if (element %% 3 == 0) {           # if the element is even
    count_even <- count_even + 1     # add one to the counter
  }
}
count_even
```

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

Useful functions:

```
seq_along(vec)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
1:length(vec)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
seq_len(length(vec))
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

2.9 While loop

```
i <- 10
while (i > 0) {
  print(i)
  i <- i - 1
}
```

```
## [1] 10
## [1] 9
## [1] 8
## [1] 7
## [1] 6
## [1] 5
## [1] 4
## [1] 3
## [1] 2
## [1] 1
```

Chapter 3

Dplyr

```
library(tidyverse)
```

3.1 Data Frame

```
tb_df <- data.frame(
  id = 1:5,
  height = c(1.7, 1.7, 1.8, 1.9, 2.0),
  weight = c(70 , 73 , 80 , 100, 95)
)
tb_df
```

```
##   id height weight
## 1  1    1.7     70
## 2  2    1.7     73
## 3  3    1.8     80
## 4  4    1.9    100
## 5  5    2.0     95
```

```
# Derived variable: Body Mass Index
tb_df$bmi <- tb_df$weight / (tb_df$height)^2
tb_df
```

```
##   id height weight      bmi
## 1  1    1.7     70 24.22145
## 2  2    1.7     73 25.25952
## 3  3    1.8     80 24.69136
## 4  4    1.9    100 27.70083
## 5  5    2.0     95 23.75000
```

3.2 Tibble

```
## NSE: Non Standard Evaluation
tb <- tibble(
  id = 1:5,
  height = c(1.7, 1.7, 1.8, 1.9, 2.0),
  weight = c(70 , 73 , 80 , 100, 95),
  # Derived variable: Body Mass Index
  bmi    = weight / height^2
)

tb
```

```
## # A tibble: 5 x 4
##      id height weight   bmi
##   <int>  <dbl>  <dbl> <dbl>
## 1     1    1.7     70  24.2
## 2     2    1.7     73  25.3
## 3     3    1.8     80  24.7
## 4     4    1.9    100  27.7
## 5     5    2       95  23.8
```

3.2.1 Data Frame to Tibble conversion

```
## cast to tbl
iris_tbl <- as_tibble(iris)
iris_tbl
```

```
## # 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
```

Save column names:

```
## rownames as a column
rownames_to_column(mtcars, var = 'car')
```

```
##                    car  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1            Mazda RX4 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## 2        Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## 3           Datsun 710 22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 4       Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## 5    Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## 6              Valiant 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## 7           Duster 360 14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## 8            Merc 240D 24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## 9             Merc 230 22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## 10            Merc 280 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## 11           Merc 280C 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## 12          Merc 450SE 16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## 13          Merc 450SL 17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## 14         Merc 450SLC 15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## 15  Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## 16 Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## 17   Chrysler Imperial 14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## 18            Fiat 128 32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## 19         Honda Civic 30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## 20      Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## 21       Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## 22    Dodge Challenger 15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## 23         AMC Javelin 15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## 24          Camaro Z28 13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## 25    Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## 26           Fiat X1-9 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## 27       Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## 28        Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## 29      Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## 30        Ferrari Dino 19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## 31       Maserati Bora 15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## 32          Volvo 142E 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
mtcars_tbl <- as_tibble(rownames_to_column(mtcars, var = 'car'))
mtcars_tbl
```

```
## # A tibble: 32 x 12
##    car           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 ~  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 D~  21.4     6  258    110  3.08  3.22  19.4     1     0     3     1
##  5 Hornet Spo~  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
```

3.3 Data import

3.3.1 Read a CSV file

CSV: comma separated values file. The first row contains the column names:

```
head data/dc-wikia-data.csv
```

```
## page_id,name,urlslug,ID,ALIGN,EYE,HAIR,SEX,GSM,ALIVE,APPEARANCES,FIRST APPEARANCE,YE
## 1422,Batman (Bruce Wayne),\/wiki\/Batman_(Bruce_Wayne),Secret Identity,Good Characte
## 23387,Superman (Clark Kent),\/wiki\/Superman_(Clark_Kent),Secret Identity,Good Chara
## 1458,Green Lantern (Hal Jordan),\/wiki\/Green_Lantern_(Hal_Jordan),Secret Identity,G
## 1659,James Gordon (New Earth),\/wiki\/James_Gordon_(New_Earth),Public Identity,Good
## 1576,Richard Grayson (New Earth),\/wiki\/Richard_Grayson_(New_Earth),Secret Identity
## 1448,Wonder Woman (Diana Prince),\/wiki\/Wonder_Woman_(Diana_Prince),Public Identity
## 1486,Aquaman (Arthur Curry),\/wiki\/Aquaman_(Arthur_Curry),Public Identity,Good Char
## 1451,Timothy Drake (New Earth),\/wiki\/Timothy_Drake_(New_Earth),Secret Identity,Goo
## 71760,Dinah Laurel Lance (New Earth),\/wiki\/Dinah_Laurel_Lance_(New_Earth),Public I
```

```
# with a specific delimiter
dc <- read_delim("data/dc-wikia-data.csv", delim = ",")

# with a standard delimiter
dc <- read_csv("data/dc-wikia-data.csv")

# with a predefined column type
dc <- read_csv('data/dc-wikia-data.csv',
               col_types = cols(
                 page_id = col_integer(),
                 name = col_character(),
                 urlslug = col_character(),
                 ID = col_factor(),
                 ALIGN = col_factor(),
                 EYE = col_factor(),
                 HAIR = col_factor(),
                 SEX = col_factor(),
                 GSM = col_character(),
                 ALIVE = col_character(),
                 APPEARANCES = col_double(),
                 `FIRST APPEARANCE` = col_character(),
                 YEAR = col_double()
```

```
                ))
```

3.3.2 Column names

Column names standardization: all lower case.

```
# Extraction
old_colnames <- colnames(dc)
old_colnames
```

```
##  [1] "page_id"          "name"             "urlslug"          "ID"
##  [5] "ALIGN"            "EYE"              "HAIR"             "SEX"
##  [9] "GSM"              "ALIVE"            "APPEARANCES"      "FIRST APPEARANCE"
## [13] "YEAR"
```

```
# manipulation
new_colnames <- tolower(old_colnames)
new_colnames
```

```
##  [1] "page_id"          "name"             "urlslug"          "id"
##  [5] "align"            "eye"              "hair"             "sex"
##  [9] "gsm"              "alive"            "appearances"      "first appearance"
## [13] "year"
```

```
# substitution
colnames(dc) <- new_colnames

# result
colnames(dc)
```

```
##  [1] "page_id"          "name"             "urlslug"          "id"
##  [5] "align"            "eye"              "hair"             "sex"
##  [9] "gsm"              "alive"            "appearances"      "first appearance"
## [13] "year"
```

or a short version for the whole process:

```
colnames(dc) <- tolower(colnames(dc))
```

3.3.3 Save to Rds format

```
# write to Rds
saveRDS(dc, "data/dc-wikia-data.Rds")
# read from Rds
dc <- readRDS("data/dc-wikia-data.Rds")
```

3.4 Base operations

3.4.1 Simple selection

```
# select columns: name, appearances, sex
dc_2 <- select(dc, name, appearances, sex)

dc_2
```

```
## # A tibble: 6,896 x 3
##    name                              appearances sex
##    <chr>                                   <dbl> <fct>
##  1 Batman (Bruce Wayne)                     3093 Male Characters
##  2 Superman (Clark Kent)                    2496 Male Characters
##  3 Green Lantern (Hal Jordan)               1565 Male Characters
##  4 James Gordon (New Earth)                 1316 Male Characters
##  5 Richard Grayson (New Earth)              1237 Male Characters
##  6 Wonder Woman (Diana Prince)              1231 Female Characters
##  7 Aquaman (Arthur Curry)                   1121 Male Characters
##  8 Timothy Drake (New Earth)                1095 Male Characters
##  9 Dinah Laurel Lance (New Earth)           1075 Female Characters
## 10 Flash (Barry Allen)                      1028 Male Characters
## # ... with 6,886 more rows
```

3.4.2 Simple filter

```
# get the "Female Characters"
dc_2_female <- filter(dc_2, sex == "Female Characters")
```

3.4.3 Pipe

Two operations in one:

```
filter(select(dc, name, appearances, sex),
       sex == "Female Characters")
```

```
## # A tibble: 1,967 x 3
##    name                               appearances sex
##    <chr>                                    <dbl> <fct>
##  1 Wonder Woman (Diana Prince)               1231 Female Characters
##  2 Dinah Laurel Lance (New Earth)            1075 Female Characters
##  3 GenderTest                                1028 Female Characters
##  4 Barbara Gordon (New Earth)                 951 Female Characters
##  5 Lois Lane (New Earth)                      934 Female Characters
##  6 Kara Zor-L (Earth-Two)                     635 Female Characters
##  7 Zatanna Zatara (New Earth)                 439 Female Characters
```

```
##  8 Cassandra Sandsmark (New Earth)         423 Female Characters
##  9 Rachel Roth (New Earth)                 399 Female Characters
## 10 Helena Bertinelli (New Earth)           393 Female Characters
## # ... with 1,957 more rows
```

Two operations in one pipeline:

```
dc %>%
  select(name, appearances, sex) %>%
  filter(sex == "Female Characters")
```

```
## # A tibble: 1,967 x 3
##    name                              appearances sex
##    <chr>                                   <dbl> <fct>
##  1 Wonder Woman (Diana Prince)              1231 Female Characters
##  2 Dinah Laurel Lance (New Earth)           1075 Female Characters
##  3 GenderTest                               1028 Female Characters
##  4 Barbara Gordon (New Earth)                951 Female Characters
##  5 Lois Lane (New Earth)                     934 Female Characters
##  6 Kara Zor-L (Earth-Two)                    635 Female Characters
##  7 Zatanna Zatara (New Earth)                439 Female Characters
##  8 Cassandra Sandsmark (New Earth)           423 Female Characters
##  9 Rachel Roth (New Earth)                   399 Female Characters
## 10 Helena Bertinelli (New Earth)             393 Female Characters
## # ... with 1,957 more rows
```

RStudio shortcut to write the *pipe* %>%: CTRL + SHIFT + m

3.4.4 Column Selection (select)

3.4.4.1 Basic selections

```
# select a range
dc %>%
  select(name, id:year)
```

```
## # A tibble: 6,896 x 11
##    name   id    align eye   hair  sex   gsm   alive appearances `first appearan~
##    <chr>  <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl> <chr>
##  1 Batma~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        3093 1939, May
##  2 Super~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        2496 1986, October
##  3 Green~ Secr~ Good~ Brow~ Brow~ Male~ <NA>  Livi~        1565 1959, October
##  4 James~ Publ~ Good~ Brow~ Whit~ Male~ <NA>  Livi~        1316 1987, February
##  5 Richa~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1237 1940, April
##  6 Wonde~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~        1231 1941, December
##  7 Aquam~ Publ~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1121 1941, November
##  8 Timot~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1095 1989, August
##  9 Dinah~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1075 1969, November
```

```
## 10 Flash~ Secr~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1028 1956, October
## # ... with 6,886 more rows, and 1 more variable: year <dbl>
```

```
# unselect
dc %>%
  select(-page_id, -urlslug)
```

```
## # A tibble: 6,896 x 11
##    name   id    align eye   hair  sex   gsm   alive appearances `first appearan~
##    <chr>  <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl> <chr>
##  1 Batma~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        3093 1939, May
##  2 Super~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        2496 1986, October
##  3 Green~ Secr~ Good~ Brow~ Brow~ Male~ <NA>  Livi~        1565 1959, October
##  4 James~ Publ~ Good~ Brow~ Whit~ Male~ <NA>  Livi~        1316 1987, February
##  5 Richa~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1237 1940, April
##  6 Wonde~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~        1231 1941, December
##  7 Aquam~ Publ~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1121 1941, November
##  8 Timot~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1095 1989, August
##  9 Dinah~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1075 1969, November
## 10 Flash~ Secr~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1028 1956, October
## # ... with 6,886 more rows, and 1 more variable: year <dbl>
```

```
# rename column 'id' to 'secret_id'
dc %>%
  rename(secret_id = id)
```

```
## # A tibble: 6,896 x 13
##    page_id name     urlslug     secret_id  align eye   hair  sex    gsm   alive
##      <int> <chr>    <chr>       <fct>      <fct> <fct> <fct> <fct>  <chr> <chr>
##  1    1422 Batman ~ "\\/wiki\\~ Secret Id~ Good~ Blue~ Blac~ Male ~ <NA>  Livin~
##  2   23387 Superma~ "\\/wiki\\~ Secret Id~ Good~ Blue~ Blac~ Male ~ <NA>  Livin~
##  3    1458 Green L~ "\\/wiki\\~ Secret Id~ Good~ Brow~ Brow~ Male ~ <NA>  Livin~
##  4    1659 James G~ "\\/wiki\\~ Public Id~ Good~ Brow~ Whit~ Male ~ <NA>  Livin~
##  5    1576 Richard~ "\\/wiki\\~ Secret Id~ Good~ Blue~ Blac~ Male ~ <NA>  Livin~
##  6    1448 Wonder ~ "\\/wiki\\~ Public Id~ Good~ Blue~ Blac~ Femal~ <NA>  Livin~
##  7    1486 Aquaman~ "\\/wiki\\~ Public Id~ Good~ Blue~ Blon~ Male ~ <NA>  Livin~
##  8    1451 Timothy~ "\\/wiki\\~ Secret Id~ Good~ Blue~ Blac~ Male ~ <NA>  Livin~
##  9   71760 Dinah L~ "\\/wiki\\~ Public Id~ Good~ Blue~ Blon~ Femal~ <NA>  Livin~
## 10    1380 Flash (~ "\\/wiki\\~ Secret Id~ Good~ Blue~ Blon~ Male ~ <NA>  Livin~
## # ... with 6,886 more rows, and 3 more variables: appearances <dbl>,
## #   first appearance <chr>, year <dbl>
```

3.4.4.2 Name matching

```
colnames(dc)
```

```
##  [1] "page_id"          "name"             "urlslug"          "id"
```

```
##  [5] "align"            "eye"              "hair"             "sex"
##  [9] "gsm"              "alive"            "appearances"      "first appearance"
## [13] "year"
```

```
# match the name
dc %>%
  select(name, contains("appearance"))
```

```
## # A tibble: 6,896 x 3
##    name                           appearances `first appearance`
##    <chr>                                <dbl> <chr>
##  1 Batman (Bruce Wayne)                  3093 1939, May
##  2 Superman (Clark Kent)                 2496 1986, October
##  3 Green Lantern (Hal Jordan)            1565 1959, October
##  4 James Gordon (New Earth)              1316 1987, February
##  5 Richard Grayson (New Earth)           1237 1940, April
##  6 Wonder Woman (Diana Prince)           1231 1941, December
##  7 Aquaman (Arthur Curry)                1121 1941, November
##  8 Timothy Drake (New Earth)             1095 1989, August
##  9 Dinah Laurel Lance (New Earth)        1075 1969, November
## 10 Flash (Barry Allen)                   1028 1956, October
## # ... with 6,886 more rows
```

```
dc %>%
  select(ends_with("id"))
```

```
## # A tibble: 6,896 x 2
##    page_id id
##      <int> <fct>
##  1    1422 Secret Identity
##  2   23387 Secret Identity
##  3    1458 Secret Identity
##  4    1659 Public Identity
##  5    1576 Secret Identity
##  6    1448 Public Identity
##  7    1486 Public Identity
##  8    1451 Secret Identity
##  9   71760 Public Identity
## 10    1380 Secret Identity
## # ... with 6,886 more rows
```

```
dc %>%
  select(starts_with("id"))
```

```
## # A tibble: 6,896 x 1
##    id
##    <fct>
##  1 Secret Identity
```

```
##  2 Secret Identity
##  3 Secret Identity
##  4 Public Identity
##  5 Secret Identity
##  6 Public Identity
##  7 Public Identity
##  8 Secret Identity
##  9 Public Identity
## 10 Secret Identity
## # ... with 6,886 more rows
```

```
dc %>% select(matches("appearance.*"))
```

```
## # A tibble: 6,896 x 2
##    appearances `first appearance`
##          <dbl> <chr>
##  1        3093 1939, May
##  2        2496 1986, October
##  3        1565 1959, October
##  4        1316 1987, February
##  5        1237 1940, April
##  6        1231 1941, December
##  7        1121 1941, November
##  8        1095 1989, August
##  9        1075 1969, November
## 10        1028 1956, October
## # ... with 6,886 more rows
```

3.4.4.3 Select by type (where)

```
dc %>% select(where(is.factor))
```

```
## # A tibble: 6,896 x 5
##    id              align           eye        hair       sex
##    <fct>           <fct>           <fct>      <fct>      <fct>
##  1 Secret Identity Good Characters Blue Eyes  Black Hair Male Characters
##  2 Secret Identity Good Characters Blue Eyes  Black Hair Male Characters
##  3 Secret Identity Good Characters Brown Eyes Brown Hair Male Characters
##  4 Public Identity Good Characters Brown Eyes White Hair Male Characters
##  5 Secret Identity Good Characters Blue Eyes  Black Hair Male Characters
##  6 Public Identity Good Characters Blue Eyes  Black Hair Female Characters
##  7 Public Identity Good Characters Blue Eyes  Blond Hair Male Characters
##  8 Secret Identity Good Characters Blue Eyes  Black Hair Male Characters
##  9 Public Identity Good Characters Blue Eyes  Blond Hair Female Characters
## 10 Secret Identity Good Characters Blue Eyes  Blond Hair Male Characters
## # ... with 6,886 more rows
```

```
dc %>% select(where(is.character))
```

```
## # A tibble: 6,896 x 5
##    name              urlslug                  gsm   alive       `first appearan~
##    <chr>             <chr>                    <chr> <chr>       <chr>
##  1 Batman (Bruce Wa~ "\\/wiki\\/Batman_(Bruc~ <NA>  Living Cha~ 1939, May
##  2 Superman (Clark ~ "\\/wiki\\/Superman_(Cl~ <NA>  Living Cha~ 1986, October
##  3 Green Lantern (H~ "\\/wiki\\/Green_Lanter~ <NA>  Living Cha~ 1959, October
##  4 James Gordon (Ne~ "\\/wiki\\/James_Gordon~ <NA>  Living Cha~ 1987, February
##  5 Richard Grayson ~ "\\/wiki\\/Richard_Gray~ <NA>  Living Cha~ 1940, April
##  6 Wonder Woman (Di~ "\\/wiki\\/Wonder_Woman~ <NA>  Living Cha~ 1941, December
##  7 Aquaman (Arthur ~ "\\/wiki\\/Aquaman_(Art~ <NA>  Living Cha~ 1941, November
##  8 Timothy Drake (N~ "\\/wiki\\/Timothy_Drak~ <NA>  Living Cha~ 1989, August
##  9 Dinah Laurel Lan~ "\\/wiki\\/Dinah_Laurel~ <NA>  Living Cha~ 1969, November
## 10 Flash (Barry All~ "\\/wiki\\/Flash_(Barry~ <NA>  Living Cha~ 1956, October
## # ... with 6,886 more rows
```

3.4.5 Unique values (distinct)

```
dc %>%
  distinct(id)
```

```
## # A tibble: 4 x 1
##   id
##   <fct>
## 1 Secret Identity
## 2 Public Identity
## 3 <NA>
## 4 Identity Unknown
```

```
dc %>%
  select(id, sex) %>%
  distinct()
```

```
## # A tibble: 14 x 2
##    id               sex
##    <fct>            <fct>
##  1 Secret Identity  Male Characters
##  2 Public Identity  Male Characters
##  3 Public Identity  Female Characters
##  4 Secret Identity  Female Characters
##  5 <NA>             Male Characters
##  6 Secret Identity  <NA>
##  7 <NA>             Female Characters
##  8 Public Identity  <NA>
##  9 <NA>             <NA>
```

```
## 10 Public Identity  Genderless Characters
## 11 Secret Identity  Genderless Characters
## 12 Identity Unknown Male Characters
## 13 <NA>             Genderless Characters
## 14 <NA>             Transgender Characters
```

`distinct` is similar to the base R function `unique`, but the latter works on vectors, not on data frames.

3.4.6 Row Filter (filter)

```
# Single condition
# Example:
# get "Female Characters" only
dc %>% filter(sex == "Female Characters")
```

```
## # A tibble: 1,967 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1    1448 Wonde~ "\\/wik~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~        1231
##  2   71760 Dinah~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1075
##  3  403631 Gende~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1028
##  4    1905 Barba~ "\\/wik~ Secr~ Good~ Blue~ Red ~ Fema~ <NA>  Livi~         951
##  5   23383 Lois ~ "\\/wik~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~         934
##  6    1580 Kara ~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         635
##  7    1577 Zatan~ "\\/wik~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~         439
##  8    1607 Cassa~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         423
##  9    1864 Rache~ "\\/wik~ Secr~ Good~ Purp~ Blac~ Fema~ <NA>  Livi~         399
## 10   37696 Helen~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~         393
## # ... with 1,957 more rows, and 2 more variables: first appearance <chr>,
## #   year <dbl>
```

```
# Match multiple conditions (all of them)
# Example:
# get the "Female Characters" AND with 'Blond Hair'
dc %>% filter(sex == "Female Characters", hair == "Blond Hair")
```

```
## # A tibble: 314 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1   71760 Dinah~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1075
##  2  403631 Gende~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1028
##  3    1580 Kara ~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         635
##  4    1607 Cassa~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         423
##  5    1696 Court~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         305
##  6    1870 Kara ~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         297
##  7   14002 Jesse~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         228
```

```
##  8    5384 Matri~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         206
##  9    1453 Steph~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         198
## 10   13102 Gabri~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         175
## # ... with 304 more rows, and 2 more variables: first appearance <chr>,
## #   year <dbl>
```

```
# or:
dc %>% filter(sex == "Female Characters" & hair == "Blond Hair")
```

```
## # A tibble: 314 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1   71760 Dinah~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1075
##  2  403631 Gende~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1028
##  3    1580 Kara ~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         635
##  4    1607 Cassa~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         423
##  5    1696 Court~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         305
##  6    1870 Kara ~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         297
##  7   14002 Jesse~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         228
##  8    5384 Matri~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         206
##  9    1453 Steph~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         198
## 10   13102 Gabri~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         175
## # ... with 304 more rows, and 2 more variables: first appearance <chr>,
## #   year <dbl>
```

```
# Match one of the alternatives (any of them)
# Example:
# get the "Female Characters" OR with 'Blond Hair'
dc %>% filter(sex == "Female Characters"  | hair == "Blond Hair")
```

```
## # A tibble: 2,397 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1    1448 Wonde~ "\\/wik~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~        1231
##  2    1486 Aquam~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1121
##  3   71760 Dinah~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1075
##  4    1380 Flash~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1028
##  5  403631 Gende~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1028
##  6    1459 Alan ~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Male~ <NA>  Dece~         969
##  7    1905 Barba~ "\\/wik~ Secr~ Good~ Blue~ Red ~ Fema~ <NA>  Livi~         951
##  8   23383 Lois ~ "\\/wik~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~         934
##  9    1580 Kara ~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~         635
## 10    1577 Zatan~ "\\/wik~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~         439
## # ... with 2,387 more rows, and 2 more variables: first appearance <chr>,
## #   year <dbl>
```

```
# Conditions on numbers
# Example:
```

```
# get characters that appeared a greater or equal number of times than 1000
dc %>% filter(appearances >= 1000)
```

```
## # A tibble: 11 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1    1422 Batma~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        3093
##  2   23387 Super~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        2496
##  3    1458 Green~ "\\/wik~ Secr~ Good~ Brow~ Brow~ Male~ <NA>  Livi~        1565
##  4    1659 James~ "\\/wik~ Publ~ Good~ Brow~ Whit~ Male~ <NA>  Livi~        1316
##  5    1576 Richa~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1237
##  6    1448 Wonde~ "\\/wik~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~        1231
##  7    1486 Aquam~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1121
##  8    1451 Timot~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1095
##  9   71760 Dinah~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1075
## 10    1380 Flash~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1028
## 11  403631 Gende~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1028
## # ... with 2 more variables: first appearance <chr>, year <dbl>
```

```
# Example:
# within a given interval (inclusive)
dc %>% filter(900 <= appearances, appearances <= 1000)
```

```
## # A tibble: 5 x 13
##   page_id name   urlslug  id     align eye   hair  sex   gsm   alive appearances
##     <int> <chr>  <chr>    <fct>  <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
## 1    1459 Alan ~ "\\/wik~ Secre~ Good~ Blue~ Blon~ Male~ <NA>  Dece~         969
## 2    1905 Barba~ "\\/wik~ Secre~ Good~ Blue~ Red ~ Fema~ <NA>  Livi~         951
## 3    1386 Jason~ "\\/wik~ Publi~ Good~ Blue~ Brow~ Male~ <NA>  Livi~         951
## 4   23383 Lois ~ "\\/wik~ Publi~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~         934
## 5    1456 Alfre~ "\\/wik~ Publi~ Good~ Blue~ Blac~ Male~ <NA>  Livi~         930
## # ... with 2 more variables: first appearance <chr>, year <dbl>
```

```
# or
dc %>% filter(between(appearances, 900, 1000))
```

```
## # A tibble: 5 x 13
##   page_id name   urlslug  id     align eye   hair  sex   gsm   alive appearances
##     <int> <chr>  <chr>    <fct>  <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
## 1    1459 Alan ~ "\\/wik~ Secre~ Good~ Blue~ Blon~ Male~ <NA>  Dece~         969
## 2    1905 Barba~ "\\/wik~ Secre~ Good~ Blue~ Red ~ Fema~ <NA>  Livi~         951
## 3    1386 Jason~ "\\/wik~ Publi~ Good~ Blue~ Brow~ Male~ <NA>  Livi~         951
## 4   23383 Lois ~ "\\/wik~ Publi~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~         934
## 5    1456 Alfre~ "\\/wik~ Publi~ Good~ Blue~ Blac~ Male~ <NA>  Livi~         930
## # ... with 2 more variables: first appearance <chr>, year <dbl>
```

```
# Example:
# outside a given interval
dc %>% filter(appearances < 900 | 1000 < appearances)
```

```
## # A tibble: 6,536 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1    1422 Batma~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        3093
##  2   23387 Super~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        2496
##  3    1458 Green~ "\\/wik~ Secr~ Good~ Brow~ Brow~ Male~ <NA>  Livi~        1565
##  4    1659 James~ "\\/wik~ Publ~ Good~ Brow~ Whit~ Male~ <NA>  Livi~        1316
##  5    1576 Richa~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1237
##  6    1448 Wonde~ "\\/wik~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~        1231
##  7    1486 Aquam~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1121
##  8    1451 Timot~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1095
##  9   71760 Dinah~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1075
## 10    1380 Flash~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1028
## # ... with 6,526 more rows, and 2 more variables: first appearance <chr>,
## #   year <dbl>
```

```
# or
dc %>% filter( ! between(appearances, 900, 1000))
```

```
## # A tibble: 6,536 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1    1422 Batma~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        3093
##  2   23387 Super~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        2496
##  3    1458 Green~ "\\/wik~ Secr~ Good~ Brow~ Brow~ Male~ <NA>  Livi~        1565
##  4    1659 James~ "\\/wik~ Publ~ Good~ Brow~ Whit~ Male~ <NA>  Livi~        1316
##  5    1576 Richa~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1237
##  6    1448 Wonde~ "\\/wik~ Publ~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~        1231
##  7    1486 Aquam~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1121
##  8    1451 Timot~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1095
##  9   71760 Dinah~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1075
## 10    1380 Flash~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1028
## # ... with 6,526 more rows, and 2 more variables: first appearance <chr>,
## #   year <dbl>
```

```
# set-in operator

# Example:
# 'appearances' has a value in a vector of possible real numbers
dc %>% filter(appearances %in% c(900, 1000))
```

```
## # A tibble: 0 x 13
## # ... with 13 variables: page_id <int>, name <chr>, urlslug <chr>, id <fct>,
```

```
## #   align <fct>, eye <fct>, hair <fct>, sex <fct>, gsm <chr>, alive <chr>,
## #   appearances <dbl>, first appearance <chr>, year <dbl>
```

```
# Example:
# 'hair' has a value in a vector of possible strings
dc %>% filter(hair %in% c('Violet Hair', 'Pink Hair'))
```

```
## # A tibble: 15 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1    1790 Brain~ "\\/wik~ Secr~ Good~ Red ~ Pink~ Fema~ <NA>  Dece~          38
##  2   11948 Susan~ "\\/wik~ Secr~ Good~ Viol~ Viol~ Fema~ <NA>  Dece~          28
##  3   11949 Flora~ "\\/wik~ Secr~ Neut~ Viol~ Viol~ Fema~ <NA>  Dece~          27
##  4   11950 Susan~ "\\/wik~ Secr~ Good~ Viol~ Viol~ Fema~ <NA>  Livi~          14
##  5  296697 Silic~ "\\/wik~ Publ~ Neut~ Whit~ Pink~ Fema~ <NA>  Dece~          12
##  6    1670 Fay M~ "\\/wik~ <NA>  Bad ~ Blue~ Pink~ Fema~ <NA>  Dece~          12
##  7  119936 Vanes~ "\\/wik~ <NA>  Bad ~ <NA>  Pink~ Fema~ <NA>  Dece~          10
##  8  132735 Grett~ "\\/wik~ Publ~ Bad ~ Blac~ Viol~ Male~ <NA>  Dece~           9
##  9   94578 Popro~ "\\/wik~ Secr~ <NA>  Blue~ Pink~ Fema~ <NA>  Livi~           7
## 10  132760 Veniz~ "\\/wik~ Publ~ Good~ <NA>  Pink~ Fema~ <NA>  Livi~           7
## 11  152150 Eduar~ "\\/wik~ <NA>  Bad ~ <NA>  Pink~ Male~ <NA>  Livi~           2
## 12  154648 Vera ~ "\\/wik~ Publ~ Bad ~ <NA>  Pink~ Fema~ <NA>  Livi~           2
## 13  273686 Hoppy~ "\\/wik~ Secr~ Good~ Blue~ Pink~ Male~ <NA>  Livi~           1
## 14  273695 Milli~ "\\/wik~ Publ~ Good~ Blue~ Pink~ Fema~ <NA>  Livi~           1
## 15  206817 B'aad~ "\\/wik~ Publ~ Bad ~ Blue~ Pink~ Male~ <NA>  Livi~          NA
## # ... with 2 more variables: first appearance <chr>, year <dbl>
```

```
# * FILTER EXTRAS

# take rows by their position index
dc %>% slice(5:10)
```

```
## # A tibble: 6 x 13
##   page_id name   urlslug  id     align eye   hair  sex   gsm   alive appearances
##     <int> <chr>  <chr>    <fct>  <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
## 1    1576 Richa~ "\\/wik~ Secre~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1237
## 2    1448 Wonde~ "\\/wik~ Publi~ Good~ Blue~ Blac~ Fema~ <NA>  Livi~        1231
## 3    1486 Aquam~ "\\/wik~ Publi~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1121
## 4    1451 Timot~ "\\/wik~ Secre~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1095
## 5   71760 Dinah~ "\\/wik~ Publi~ Good~ Blue~ Blon~ Fema~ <NA>  Livi~        1075
## 6    1380 Flash~ "\\/wik~ Secre~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1028
## # ... with 2 more variables: first appearance <chr>, year <dbl>
```

```
# Extract a sample of 'n' lines
dc %>% sample_n(3)
```

```
## # A tibble: 3 x 13
##   page_id name   urlslug  id     align eye   hair  sex   gsm   alive appearances
```

```
##     <int> <chr>  <chr>    <fct>  <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
## 1   80996 Geoff~ "\\/wik~ <NA>   Bad ~ Blac~ <NA>  Male~ <NA>  Livi~          22
## 2  159025 Phili~ "\\/wik~ <NA>   <NA>  <NA>  Brow~ Male~ <NA>  Dece~           2
## 3  265509 Pom P~ "\\/wik~ Secre~ Bad ~ <NA>  Blon~ Fema~ <NA>  Livi~           1
## # ... with 2 more variables: first appearance <chr>, year <dbl>
```

```
# Extract a fraction of all lines
dc %>% sample_frac(0.1)
```

```
## # A tibble: 690 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1  161253 Armle~ "\\/wik~ <NA>  Neut~ <NA>  <NA>  Male~ <NA>  Dece~           2
##  2  291990 Ulgo ~ "\\/wik~ Publ~ Good~ <NA>  Blac~ Male~ <NA>  Livi~           3
##  3   18084 Charl~ "\\/wik~ Publ~ Good~ Ambe~ <NA>  Male~ <NA>  Livi~          33
##  4  131080 Chama~ "\\/wik~ <NA>  Bad ~ <NA>  <NA>  Male~ <NA>  Livi~           3
##  5   32183 Glomm~ "\\/wik~ <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  Livi~           1
##  6   11644 Eliza~ "\\/wik~ Publ~ Good~ Brow~ Brow~ Fema~ <NA>  Dece~           3
##  7  213796 Goode~ "\\/wik~ Publ~ Bad ~ <NA>  Red ~ Male~ <NA>  Livi~           3
##  8  213598 McCaf~ "\\/wik~ <NA>  Good~ Brow~ Whit~ Male~ <NA>  Dece~           8
##  9   17640 Jake ~ "\\/wik~ Publ~ Good~ Brow~ Blac~ Male~ <NA>  Livi~          14
## 10    7585 Obero~ "\\/wik~ Publ~ <NA>  Blue~ Whit~ Male~ <NA>  Livi~         167
## # ... with 680 more rows, and 2 more variables: first appearance <chr>,
## #   year <dbl>
```

```
# Re-sample repeating lines
dc %>% sample_frac(1.5, replace = TRUE)
```

```
## # A tibble: 10,344 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1  183681 Jack ~ "\\/wik~ <NA>  Bad ~ <NA>  <NA>  Male~ <NA>  Livi~           1
##  2   16376 Harol~ "\\/wik~ Secr~ <NA>  Blue~ Blon~ Male~ <NA>  Livi~          86
##  3    1923 James~ "\\/wik~ Publ~ Bad ~ Blue~ Brow~ Male~ <NA>  Dece~          52
##  4  263718 Conco~ "\\/wik~ Publ~ Bad ~ <NA>  <NA>  Male~ <NA>  Livi~           3
##  5  116949 Edwar~ "\\/wik~ <NA>  Good~ Blue~ Blon~ Male~ <NA>  Livi~          20
##  6    8812 Artem~ "\\/wik~ Secr~ Bad ~ Brow~ Blac~ Fema~ <NA>  Livi~          11
##  7  239032 Uglyh~ "\\/wik~ Secr~ Bad ~ Blac~ Blac~ Male~ <NA>  Livi~           1
##  8  178464 Garri~ "\\/wik~ Publ~ Neut~ Brow~ Blac~ Male~ <NA>  Livi~          12
##  9  183682 Jack ~ "\\/wik~ <NA>  Bad ~ <NA>  <NA>  Male~ <NA>  Livi~           1
## 10  127053 Big G~ "\\/wik~ Secr~ Bad ~ <NA>  Red ~ Male~ <NA>  Dece~           2
## # ... with 10,334 more rows, and 2 more variables: first appearance <chr>,
## #   year <dbl>
```

```
# if you want to fix the random state of the random sampling
set.seed(123)
dc %>% sample_n(3)
```

```
## # A tibble: 3 x 13
##   page_id name   urlslug  id     align eye   hair  sex   gsm   alive appearances
##     <int> <chr>  <chr>    <fct>  <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
## 1  238752 Freew~ "\\/wik~ Secre~ Bad ~ <NA>  <NA>  Male~ <NA>  Livi~           9
## 2   51183 Danie~ "\\/wik~ <NA>   Bad ~ Blue~ Brow~ Male~ <NA>  Dece~           9
## 3   70010 Mars ~ "\\/wik~ Publi~ Neut~ <NA>  <NA>  Male~ <NA>  Livi~          11
## # ... with 2 more variables: first appearance <chr>, year <dbl>
```

3.4.7 Row sorting (arrange)

Sort using a column as a criteria:

```
dc %>%
  select(name, appearances, sex) %>%
  filter(sex == "Female Characters") %>%
  arrange(appearances)
```

```
## # A tibble: 1,967 x 3
##    name                           appearances sex
##    <chr>                                <dbl> <fct>
##  1 Adellca (New Earth)                      1 Female Characters
##  2 Big Mummy (New Earth)                    1 Female Characters
##  3 Bizarro Power Girl (New Earth)           1 Female Characters
##  4 Cockroach (New Earth)                    1 Female Characters
##  5 Jennifer Myers (New Earth)               1 Female Characters
##  6 Jinny Greenteeth (New Earth)             1 Female Characters
##  7 Joan Ng (New Earth)                      1 Female Characters
##  8 Lattea (New Earth)                       1 Female Characters
##  9 Naomi Lord (New Earth)                   1 Female Characters
## 10 Yo-Yo (New Earth)                        1 Female Characters
## # ... with 1,957 more rows
```

```
# sort by one column
dc_2 %>% arrange(appearances)
```

```
## # A tibble: 6,896 x 3
##    name                            appearances sex
##    <chr>                                 <dbl> <fct>
##  1 Springheeled Jack (Prime Earth)           1 Male Characters
##  2 Napalm (Prime Earth)                      1 Male Characters
##  3 Adellca (New Earth)                       1 Female Characters
##  4 Armory (New Earth)                        1 Male Characters
##  5 Artois (New Earth)                        1 Male Characters
##  6 Aya (New Earth)                           1 <NA>
##  7 Backslash (New Earth)                     1 Male Characters
##  8 Bayfrentos (New Earth)                    1 Male Characters
##  9 Big Mummy (New Earth)                     1 Female Characters
```

```
## 10 Bizarro Blue Beetle (New Earth)           1 Male Characters
## # ... with 6,886 more rows
```

```
# sort by one column in descending order
dc_2 %>% arrange(desc(appearances))
```

```
## # A tibble: 6,896 x 3
##    name                            appearances sex
##    <chr>                                 <dbl> <fct>
##  1 Batman (Bruce Wayne)                   3093 Male Characters
##  2 Superman (Clark Kent)                  2496 Male Characters
##  3 Green Lantern (Hal Jordan)             1565 Male Characters
##  4 James Gordon (New Earth)               1316 Male Characters
##  5 Richard Grayson (New Earth)            1237 Male Characters
##  6 Wonder Woman (Diana Prince)            1231 Female Characters
##  7 Aquaman (Arthur Curry)                 1121 Male Characters
##  8 Timothy Drake (New Earth)              1095 Male Characters
##  9 Dinah Laurel Lance (New Earth)         1075 Female Characters
## 10 Flash (Barry Allen)                    1028 Male Characters
## # ... with 6,886 more rows
```

```
# sort by two criteria (the first has the precedence)
dc %>% arrange(sex, desc(appearances))
```

```
## # A tibble: 6,896 x 13
##    page_id name   urlslug  id    align eye   hair  sex   gsm   alive appearances
##      <int> <chr>  <chr>    <fct> <fct> <fct> <fct> <fct> <chr> <chr>       <dbl>
##  1    1422 Batma~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        3093
##  2   23387 Super~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        2496
##  3    1458 Green~ "\\/wik~ Secr~ Good~ Brow~ Brow~ Male~ <NA>  Livi~        1565
##  4    1659 James~ "\\/wik~ Publ~ Good~ Brow~ Whit~ Male~ <NA>  Livi~        1316
##  5    1576 Richa~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1237
##  6    1486 Aquam~ "\\/wik~ Publ~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1121
##  7    1451 Timot~ "\\/wik~ Secr~ Good~ Blue~ Blac~ Male~ <NA>  Livi~        1095
##  8    1380 Flash~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Male~ <NA>  Livi~        1028
##  9    1459 Alan ~ "\\/wik~ Secr~ Good~ Blue~ Blon~ Male~ <NA>  Dece~         969
## 10    1386 Jason~ "\\/wik~ Publ~ Good~ Blue~ Brow~ Male~ <NA>  Livi~         951
## # ... with 6,886 more rows, and 2 more variables: first appearance <chr>,
## #   year <dbl>
```

3.5 Transform columns (mutate)

3.5.1 Create a new column

```
dc %>%
  mutate(age = 2021 - year) %>%
```

```
  arrange(desc(age)) %>%
  select(name, age)
```

```
## # A tibble: 6,896 x 2
##    name                                  age
##    <chr>                               <dbl>
##  1 Richard Occult (New Earth)             86
##  2 Merlin (New Earth)                     85
##  3 Franklin Delano Roosevelt (New Earth)  85
##  4 Arthur Pendragon (New Earth)           85
##  5 Lancelot (New Earth)                   85
##  6 Guinevere (New Earth)                  85
##  7 Lady of the Lake (New Earth)           85
##  8 Gawain (New Earth)                     85
##  9 Gareth (New Earth)                     85
## 10 Bedivere (New Earth)                   85
## # ... with 6,886 more rows
```

```
# install.packages("nycflights13")
library(nycflights13)

flights %>%
  select(dep_delay, arr_delay) %>%
  mutate(time_gain = dep_delay - arr_delay)
```

```
## # A tibble: 336,776 x 3
##    dep_delay arr_delay time_gain
##        <dbl>     <dbl>     <dbl>
##  1         2        11        -9
##  2         4        20       -16
##  3         2        33       -31
##  4        -1       -18        17
##  5        -6       -25        19
##  6        -4        12       -16
##  7        -5        19       -24
##  8        -3       -14        11
##  9        -3        -8         5
## 10        -2         8       -10
## # ... with 336,766 more rows
```

3.5.2 Segment data values into bins

In order to create a categorical variable from a continuous variable you need to segment and sort data values into bins. You can do that with the `case_when` or the `cut` functions.

3.5.2.1 'cut' Example

Here a `cut` example. `cut` splits a variable in classes, each of them correspond to an interval of values.

You can automatically split the range of `year` in a number of classes:

```
# You can automatically split the range of `year` in a number of classes:
dc_2 <- dc %>%
  mutate(classe = cut(year, breaks = 10)) %>%
  select(name, year, classe)
dc_2
```

```
## # A tibble: 6,896 x 3
##    name                           year classe
##    <chr>                         <dbl> <fct>
##  1 Batman (Bruce Wayne)           1939 (1935,1943]
##  2 Superman (Clark Kent)          1986 (1982,1990]
##  3 Green Lantern (Hal Jordan)     1959 (1958,1966]
##  4 James Gordon (New Earth)       1987 (1982,1990]
##  5 Richard Grayson (New Earth)    1940 (1935,1943]
##  6 Wonder Woman (Diana Prince)    1941 (1935,1943]
##  7 Aquaman (Arthur Curry)         1941 (1935,1943]
##  8 Timothy Drake (New Earth)      1989 (1982,1990]
##  9 Dinah Laurel Lance (New Earth) 1969 (1966,1974]
## 10 Flash (Barry Allen)            1956 (1951,1958]
## # ... with 6,886 more rows
```

Or you can provide a set of separators (`breaks`) and labels for the intervals. Mind the number of breaks should be one more than the labels.

```
# separa le classi sulla variabile separatori e attribuisci la corretta etichetta.
breaks <- 0:10 * 10L + 1900L
labels <- paste('years', as.character(0:9 * 10L))
dc_2 <- dc %>%
  mutate(classe = cut(year,
                      breaks = breaks,
                      labels = labels)) %>%
  select(name, year, classe)
dc_2
```

```
## # A tibble: 6,896 x 3
##    name                           year classe
##    <chr>                         <dbl> <fct>
##  1 Batman (Bruce Wayne)           1939 years 30
##  2 Superman (Clark Kent)          1986 years 80
##  3 Green Lantern (Hal Jordan)     1959 years 50
##  4 James Gordon (New Earth)       1987 years 80
##  5 Richard Grayson (New Earth)    1940 years 30
```

```
##  6 Wonder Woman (Diana Prince)     1941 years 40
##  7 Aquaman (Arthur Curry)          1941 years 40
##  8 Timothy Drake (New Earth)       1989 years 80
##  9 Dinah Laurel Lance (New Earth)  1969 years 60
## 10 Flash (Barry Allen)             1956 years 50
## # ... with 6,886 more rows
```

Detail. Here an extract of the procedure, the cut function applied on a single number:

```
cut(1939,
    breaks = breaks,
    labels = labels)
```

```
## [1] years 30
## 10 Levels: years 0 years 10 years 20 years 30 years 40 years 50 ... years 90
```

```
# 1939 in in the interval between the 4th and 5th breaks
between(1939, breaks[4], breaks[5])
```

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

```
# the selected labels will be the 4th
labels[4]
```

```
## [1] "years 30"
```

Documentation:

```
help(cut)
```

3.5.2.2 case_when example

```
flights %>%
  select(arr_delay) %>%
  mutate(delay_class = case_when(
    arr_delay > 1000 ~ "big-delay",
    arr_delay < 1000 & arr_delay > 0 ~ "delay",
    arr_delay <= 0 ~ "no-delay",
    TRUE ~ NA_character_
  ))
```

```
## # A tibble: 336,776 x 2
##    arr_delay delay_class
##        <dbl> <chr>
##  1        11 delay
##  2        20 delay
##  3        33 delay
##  4       -18 no-delay
##  5       -25 no-delay
```

```
##  6        12 delay
##  7        19 delay
##  8       -14 no-delay
##  9        -8 no-delay
## 10         8 delay
## # ... with 336,766 more rows
```

3.6 Aggregate rows (summarise)

3.6.1 Scalar-returning aggregations

`summarise` works with functions that return a scalar:

$$R^n \to R$$

For example, the `mean` function take a vector and returns a single value

```
mean(flights$arr_delay, na.rm = TRUE)
```

```
## [1] 6.895377
```

3.6.2 Aggregate to a scalar

Prepare the dataset:

```
# install.packages('nycflights13')
library(nycflights13)

flights_tiny <-
  flights %>%
  select(dep_delay, arr_delay, carrier, origin, dest)
flights_tiny
```

```
## # A tibble: 336,776 x 5
##    dep_delay arr_delay carrier origin dest
##        <dbl>     <dbl> <chr>   <chr>  <chr>
##  1         2        11 UA      EWR    IAH
##  2         4        20 UA      LGA    IAH
##  3         2        33 AA      JFK    MIA
##  4        -1       -18 B6      JFK    BQN
##  5        -6       -25 DL      LGA    ATL
##  6        -4        12 UA      EWR    ORD
##  7        -5        19 B6      EWR    FLL
##  8        -3       -14 EV      LGA    IAD
##  9        -3        -8 B6      JFK    MCO
## 10        -2         8 AA      LGA    ORD
## # ... with 336,766 more rows
```

Calculate a single aggregation:

```
flights_tiny %>%
  summarise(mean_arr_delay = mean(arr_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   mean_arr_delay
##            <dbl>
## 1           6.90
```

Calculate multiple aggregations:

```
flights_tiny %>%
  summarise(mean_arr_delay   = mean(arr_delay, na.rm = TRUE),
            sd_arr_delay     = sd(arr_delay, na.rm = TRUE),
            mean_dep_delay   = mean(dep_delay, na.rm = TRUE),
            median_arr_delay = median(arr_delay, na.rm = TRUE),
            first_quartile_arr_delay  = quantile(arr_delay, probs = 0.25, na.rm = TRUE
            second_quartile_arr_delay = quantile(arr_delay, probs = 0.5, na.rm = TRUE)
```

```
## # A tibble: 1 x 6
##   mean_arr_delay sd_arr_delay mean_dep_delay median_arr_delay first_quartile_ar~
##            <dbl>        <dbl>          <dbl>            <dbl>               <dbl>
## 1           6.90         44.6           12.6               -5                 -17
## # ... with 1 more variable: second_quartile_arr_delay <dbl>
```

3.6.3 Vector-returning aggregations

An example is the `quantile` function that return a vector of length equals to `probs`:

```
vec <- 1:11
vec
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11
```

```
quantile(vec, probs = c(0, 0.25, 0.5, 0.75, 1), na.rm = TRUE)
```

```
##   0%  25%  50%  75% 100%
##  1.0  3.5  6.0  8.5 11.0
```

Here the application to `arr_delay`:

```
arr_delay <- flights_tiny$arr_delay

# R^n -> R^m
quantile(arr_delay, probs = c(0, 0.25, 0.5, 0.75, 1), na.rm = TRUE)
```

```
##   0%  25%  50%  75% 100%
##  -86  -17   -5   14 1272
```

3.6.4 Aggregate to a vector

`summarise` works also with functions that return a vector of a length different from the starting length. (If the function returns a vector of the same length you will probably want to use `mutate`)

```
# summarise restituisce questa volta un tibble di 3 righe
flights_tiny %>%
  summarise(quartile_value = quantile(arr_delay, probs = c(0.25, 0.5, 0.75), na.rm = TRUE),
            quartile_key   = names(quartile_value)) %>%
  select(quartile_key, quartile_value)
```

```
## # A tibble: 3 x 2
##   quartile_key quartile_value
##   <chr>                 <dbl>
## 1 25%                     -17
## 2 50%                      -5
## 3 75%                      14
```

3.7 Compute per groups

3.7.1 Aggregation per groups

Aggregate per a single variable group:

```
flights %>%
  group_by(carrier) %>%
  summarise(mean_arr_delay = mean(arr_delay, na.rm = TRUE))
```

```
## # A tibble: 16 x 2
##    carrier mean_arr_delay
##    <chr>            <dbl>
##  1 9E               7.38
##  2 AA               0.364
##  3 AS              -9.93
##  4 B6               9.46
##  5 DL               1.64
##  6 EV              15.8
##  7 F9              21.9
##  8 FL              20.1
##  9 HA              -6.92
## 10 MQ              10.8
## 11 OO              11.9
## 12 UA               3.56
## 13 US               2.13
## 14 VX               1.76
## 15 WN               9.65
```

```
## 16 YV              15.6
```

That aggregation has been made once per each of these groups:

```
flights %>%
  distinct(carrier)
```

```
## # A tibble: 16 x 1
##    carrier
##    <chr>
##  1 UA
##  2 AA
##  3 B6
##  4 DL
##  5 EV
##  6 MQ
##  7 US
##  8 WN
##  9 VX
## 10 FL
## 11 AS
## 12 9E
## 13 F9
## 14 HA
## 15 YV
## 16 OO
```

Aggregate per groups generated by multiple variables:

```
flights %>%
  group_by(carrier, origin, dest) %>%
  summarise(mean_arr_delay = mean(arr_delay, na.rm = TRUE)) %>%
  ungroup()
```

```
## # A tibble: 439 x 4
##    carrier origin dest  mean_arr_delay
##    <chr>   <chr>  <chr>          <dbl>
##  1 9E      EWR    ATL            -6.25
##  2 9E      EWR    CVG             1.40
##  3 9E      EWR    DTW             2.54
##  4 9E      EWR    MSP             1.60
##  5 9E      JFK    ATL             1.40
##  6 9E      JFK    AUS            -3.5
##  7 9E      JFK    BNA            10.2
##  8 9E      JFK    BOS             5.66
##  9 9E      JFK    BUF             6.67
## 10 9E      JFK    BWI             8.73
## # ... with 429 more rows
```

That aggregation has been made once per each of these groups:

```
flights %>%
  distinct(carrier, origin, dest)
```

```
## # A tibble: 439 x 3
##    carrier origin dest
##    <chr>   <chr>  <chr>
##  1 UA      EWR    IAH
##  2 UA      LGA    IAH
##  3 AA      JFK    MIA
##  4 B6      JFK    BQN
##  5 DL      LGA    ATL
##  6 UA      EWR    ORD
##  7 B6      EWR    FLL
##  8 EV      LGA    IAD
##  9 B6      JFK    MCO
## 10 AA      LGA    ORD
## # ... with 429 more rows
```

3.7.2 Grouping attribute

Group is an attribute into the data frame, not a real split. It can be removed with the `ungroup()` function.

```
# summarise remove only one grouping variable by default
# you can read the 'groups' attribute in the tibble
tbl <- flights %>%
  group_by(carrier, origin, dest) %>%
  summarise(mean_arr_delay = mean(arr_delay, na.rm = TRUE))

# before doing another aggregation you may want to 'ungroup()'
tbl %>%
  ungroup() %>%
  summarise(mean_arr_delay = mean(mean_arr_delay, na.rm = T))
```

```
## # A tibble: 1 x 1
##   mean_arr_delay
##            <dbl>
## 1           7.31
```

```
# here you see that the grouping attribute can be stored with the tibble variable itself
grouped_flights <- flights %>%
  group_by(carrier)

grouped_flights %>%
  summarise(mean_arr_delay = mean(arr_delay, na.rm = TRUE))
```

```
## # A tibble: 16 x 2
##    carrier mean_arr_delay
##    <chr>            <dbl>
##  1 9E               7.38
##  2 AA               0.364
##  3 AS              -9.93
##  4 B6               9.46
##  5 DL               1.64
##  6 EV              15.8
##  7 F9              21.9
##  8 FL              20.1
##  9 HA              -6.92
## 10 MQ              10.8
## 11 OO              11.9
## 12 UA               3.56
## 13 US               2.13
## 14 VX               1.76
## 15 WN               9.65
## 16 YV              15.6
```

3.7.3 Transformation per group

```
df <- tibble(
  warehouse = rep(LETTERS[1:5], times = 5),
  time = rep(1:5, each = 5),
  total_quantity = as.integer(runif(5 * 5, min = 1, max = 100))
) %>%
  arrange(warehouse, time)
df
```

```
## # A tibble: 25 x 3
##    warehouse  time total_quantity
##    <chr>     <int>          <int>
##  1 A             1             88
##  2 A             2             55
##  3 A             3             57
##  4 A             4             33
##  5 A             5             99
##  6 B             1             94
##  7 B             2             46
##  8 B             3             11
##  9 B             4             95
## 10 B             5             65
## # ... with 15 more rows
```

- `warehouse` id of the facility where quantity is stored.

- `time` the number of the day for example
- `total_quantity` the number of items stored in the warehouse at that time.

The task now, is to find the `variation` of the number of items per day:

```
df %>%
  group_by(warehouse) %>%
  mutate(
    previous_total_quantity = lag(total_quantity),
    variation = total_quantity - previous_total_quantity)
```

```
## # A tibble: 25 x 5
## # Groups:   warehouse [5]
##    warehouse  time total_quantity previous_total_quantity variation
##    <chr>     <int>          <int>                   <int>     <int>
##  1 A             1             88                      NA        NA
##  2 A             2             55                      88       -33
##  3 A             3             57                      55         2
##  4 A             4             33                      57       -24
##  5 A             5             99                      33        66
##  6 B             1             94                      NA        NA
##  7 B             2             46                      94       -48
##  8 B             3             11                      46       -35
##  9 B             4             95                      11        84
## 10 B             5             65                      95       -30
## # ... with 15 more rows
```

Remarks:

- every day 1 the previous_total_quantity is `NA`, and therefore the `variation` (because it is the difference of a number and a `NA`)
- `group_by` is mandatory to keep different `warehouse`s separated each others.

3.8 Combine multiple data frames

3.8.1 Joins

3.8.1.1 Basics

Joins are a way to merge table based on the correspondence on a key.

```
library(tidyverse)
```

Let us a couple of example tables. Let us define a foreign-key a column whose values have a correspondence in another table. This creates a relationship among two tables.

```
# Main table, where
# 'id' id number
# 'lower' a foreign key
main_tbl <- tibble(
  id = c(1:2, 3, 4:6), #1:5,
  lower = c(letters[1:2], 'non-letter', letters[4:6])
)
main_tbl
```

```
## # A tibble: 6 x 2
##      id lower
##   <dbl> <chr>
## 1     1 a
## 2     2 b
## 3     3 non-letter
## 4     4 d
## 5     5 e
## 6     6 f
```

And another that contains (usually all) the occurrences of the foreign-key:

```
# The table of letters
# 'lower' list all the lower case letters
# 'upper' an attribute to the letters, for the example case a upper case copy of the l
letter_tbl <- tibble(
  lower = letters,
  upper = LETTERS
)
letter_tbl
```

```
## # A tibble: 26 x 2
##    lower upper
##    <chr> <chr>
##  1 a     A
##  2 b     B
##  3 c     C
##  4 d     D
##  5 e     E
##  6 f     F
##  7 g     G
##  8 h     H
##  9 i     I
## 10 j     J
## # ... with 16 more rows
```

```
# full-join
```

```
main_tbl %>%
  full_join(letter_tbl)
```

```
## # A tibble: 27 x 3
##       id lower      upper
##    <dbl> <chr>      <chr>
##  1     1 a          A
##  2     2 b          B
##  3     3 non-letter <NA>
##  4     4 d          D
##  5     5 e          E
##  6     6 f          F
##  7    NA c          C
##  8    NA g          G
##  9    NA h          H
## 10    NA i          I
## # ... with 17 more rows
```

```
# inner-join

main_tbl %>%
  inner_join(letter_tbl)
```

```
## # A tibble: 5 x 3
##      id lower upper
##   <dbl> <chr> <chr>
## 1     1 a     A
## 2     2 b     B
## 3     4 d     D
## 4     5 e     E
## 5     6 f     F
```

```
# left-join

main_tbl %>%
  left_join(letter_tbl)
```

```
## # A tibble: 6 x 3
##      id lower      upper
##   <dbl> <chr>      <chr>
## 1     1 a          A
## 2     2 b          B
## 3     3 non-letter <NA>
## 4     4 d          D
## 5     5 e          E
## 6     6 f          F
```

```
# right-join

main_tbl %>%
  right_join(letter_tbl)
```

```
## # A tibble: 26 x 3
##       id lower upper
##    <dbl> <chr> <chr>
##  1     1 a     A
##  2     2 b     B
##  3     4 d     D
##  4     5 e     E
##  5     6 f     F
##  6    NA c     C
##  7    NA g     G
##  8    NA h     H
##  9    NA i     I
## 10    NA j     J
## # ... with 16 more rows
```

```
# inverting the order a `right_join` return the same as a `left_join`
letter_tbl %>%
  right_join(main_tbl)
```

```
## # A tibble: 6 x 3
##   lower      upper    id
##   <chr>      <chr> <dbl>
## 1 a          A         1
## 2 b          B         2
## 3 d          D         4
## 4 e          E         5
## 5 f          F         6
## 6 non-letter <NA>      3
```

3.8.1.2 Example

```
library(nycflights13)
# data(package = "nycflights13")
```

`tailnum` is plate number of a single plane:

```
# interesting columns
flights_small <- flights %>%
  select(origin, dest, tailnum)
flights_small
```

```
## # A tibble: 336,776 x 3
```

```
##    origin dest  tailnum
##    <chr>  <chr> <chr>
##  1 EWR    IAH   N14228
##  2 LGA    IAH   N24211
##  3 JFK    MIA   N619AA
##  4 JFK    BQN   N804JB
##  5 LGA    ATL   N668DN
##  6 EWR    ORD   N39463
##  7 EWR    FLL   N516JB
##  8 LGA    IAD   N829AS
##  9 JFK    MCO   N593JB
## 10 LGA    ORD   N3ALAA
## # ... with 336,766 more rows
```

Information of a single plane are stored in `planes` table (loaded with the package `nycflights13`):

```
planes
```

```
## # A tibble: 3,322 x 9
##    tailnum  year type          manufacturer   model  engines seats speed engine
##    <chr>   <int> <chr>         <chr>          <chr>    <int> <int> <int> <chr>
##  1 N10156   2004 Fixed wing m~ EMBRAER        EMB-1~       2    55    NA Turbo-~
##  2 N102UW   1998 Fixed wing m~ AIRBUS INDUST~ A320-~       2   182    NA Turbo-~
##  3 N103US   1999 Fixed wing m~ AIRBUS INDUST~ A320-~       2   182    NA Turbo-~
##  4 N104UW   1999 Fixed wing m~ AIRBUS INDUST~ A320-~       2   182    NA Turbo-~
##  5 N10575   2002 Fixed wing m~ EMBRAER        EMB-1~       2    55    NA Turbo-~
##  6 N105UW   1999 Fixed wing m~ AIRBUS INDUST~ A320-~       2   182    NA Turbo-~
##  7 N107US   1999 Fixed wing m~ AIRBUS INDUST~ A320-~       2   182    NA Turbo-~
##  8 N108UW   1999 Fixed wing m~ AIRBUS INDUST~ A320-~       2   182    NA Turbo-~
##  9 N109UW   1999 Fixed wing m~ AIRBUS INDUST~ A320-~       2   182    NA Turbo-~
## 10 N110UW   1999 Fixed wing m~ AIRBUS INDUST~ A320-~       2   182    NA Turbo-~
## # ... with 3,312 more rows
```

For each flight (row) we want to associate the corresponding plane information, by matching the `tailnum` of the flying airplane.

```
flights_small %>%
  left_join(planes, by = "tailnum")
```

```
## # A tibble: 336,776 x 11
##    origin dest  tailnum  year type      manufacturer  model  engines seats speed
##    <chr>  <chr> <chr>   <int> <chr>     <chr>         <chr>    <int> <int> <int>
##  1 EWR    IAH   N14228   1999 Fixed wi~ BOEING        737-8~       2   149    NA
##  2 LGA    IAH   N24211   1998 Fixed wi~ BOEING        737-8~       2   149    NA
##  3 JFK    MIA   N619AA   1990 Fixed wi~ BOEING        757-2~       2   178    NA
##  4 JFK    BQN   N804JB   2012 Fixed wi~ AIRBUS        A320-~       2   200    NA
##  5 LGA    ATL   N668DN   1991 Fixed wi~ BOEING        757-2~       2   178    NA
```

```
##  6 EWR    ORD   N39463   2012 Fixed wi~ BOEING        737-9~       2   191    NA
##  7 EWR    FLL   N516JB   2000 Fixed wi~ AIRBUS INDUS~ A320-~       2   200    NA
##  8 LGA    IAD   N829AS   1998 Fixed wi~ CANADAIR      CL-60~       2    55    NA
##  9 JFK    MCO   N593JB   2004 Fixed wi~ AIRBUS        A320-~       2   200    NA
## 10 LGA    ORD   N3ALAA     NA <NA>      <NA>          <NA>        NA    NA    NA
## # ... with 336,766 more rows, and 1 more variable: engine <chr>
```

In case the two dataframes have the matching columns with a different name (we rename one of the two tables to create a simple exercise) you can specify the matching names:

```
# rename a column
flights_2 <- flights_small %>%
  rename(my_tailnum = "tailnum")

# join specifying the joining columns if they have different names on the two tables
flights_2 %>%
  left_join(
    planes,
    by = c("my_tailnum" = "tailnum")
  )
```

```
## # A tibble: 336,776 x 11
##    origin dest  my_tailnum  year type    manufacturer  model engines seats speed
##    <chr>  <chr> <chr>      <int> <chr>   <chr>         <chr>   <int> <int> <int>
##  1 EWR    IAH   N14228      1999 Fixed ~ BOEING        737-~       2   149    NA
##  2 LGA    IAH   N24211      1998 Fixed ~ BOEING        737-~       2   149    NA
##  3 JFK    MIA   N619AA      1990 Fixed ~ BOEING        757-~       2   178    NA
##  4 JFK    BQN   N804JB      2012 Fixed ~ AIRBUS        A320~       2   200    NA
##  5 LGA    ATL   N668DN      1991 Fixed ~ BOEING        757-~       2   178    NA
##  6 EWR    ORD   N39463      2012 Fixed ~ BOEING        737-~       2   191    NA
##  7 EWR    FLL   N516JB      2000 Fixed ~ AIRBUS INDUS~ A320~       2   200    NA
##  8 LGA    IAD   N829AS      1998 Fixed ~ CANADAIR      CL-6~       2    55    NA
##  9 JFK    MCO   N593JB      2004 Fixed ~ AIRBUS        A320~       2   200    NA
## 10 LGA    ORD   N3ALAA        NA <NA>    <NA>          <NA>       NA    NA    NA
## # ... with 336,766 more rows, and 1 more variable: engine <chr>
```

```
# join matching by more than one column on both tables
flights %>%
  left_join(weather, by = c("year", "month", "day", "origin", "hour"))
```

```
## # A tibble: 336,776 x 29
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
```

```
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 21 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour.x <dttm>, temp <dbl>, dewp <dbl>, humid <dbl>, wind_dir <dbl>,
## #   wind_speed <dbl>, wind_gust <dbl>, precip <dbl>, pressure <dbl>,
## #   visib <dbl>, time_hour.y <dttm>
```

3.8.1.3 Joins can replicate rows

In case of a `left_join` (but it works in a similar way with all the joins) if the left table has replication in its foreign-key (`lower`) is normal and everything will work as expected.

```
# create a lower column with repetitions (the classic foreing-key can contain repeted values)
main_tbl <- tibble(
  id = c(1:2, 3, 4:6), #1:5,
  lower = c(letters[1:2], 'non-letter', letters[1:3])
)
# the "lower" column of the right table does not contain repetitions: there is
# no row duplication in the join.
main_tbl %>%
  left_join(letter_tbl, by = "lower")
```

```
## # A tibble: 6 x 3
##      id lower      upper
##   <dbl> <chr>      <chr>
## 1     1 a          A
## 2     2 b          B
## 3     3 non-letter <NA>
## 4     4 a          A
## 5     5 b          B
## 6     6 c          C
```

in case also the right-table contains repetition, left rows will be duplicated for any row matching in the right table.

```
# the "lower" column of the right table NOW DOES CONTAIN repetitions: there is
# row duplication in the join!

letter_tbl <- tibble(
  lower = c(letters, letters),
  upper = c(LETTERS, LETTERS)
```

```
) %>%
  arrange(lower)
letter_tbl
```

```
## # A tibble: 52 x 2
##    lower upper
##    <chr> <chr>
##  1 a     A
##  2 a     A
##  3 b     B
##  4 b     B
##  5 c     C
##  6 c     C
##  7 d     D
##  8 d     D
##  9 e     E
## 10 e     E
## # ... with 42 more rows
```

```
main_tbl %>%
  left_join(letter_tbl, by = "lower")
```

```
## # A tibble: 11 x 3
##       id lower      upper
##    <dbl> <chr>      <chr>
##  1     1 a          A
##  2     1 a          A
##  3     2 b          B
##  4     2 b          B
##  5     3 non-letter <NA>
##  6     4 a          A
##  7     4 a          A
##  8     5 b          B
##  9     5 b          B
## 10     6 c          C
## 11     6 c          C
```

3.8.2 Split and Unite data frames

3.8.2.1 Bind rows

```
mtcar_list <- split(mtcars, as.character(mtcars$cyl))
mtcars_cyl_6 <- mtcar_list[["6"]]
mtcars_cyl_8 <- mtcar_list[["8"]]
```

These two data frames have the same schema, and their rows can be concate-

nated:

```
bind_rows(mtcars_cyl_6, mtcars_cyl_8)
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C         17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 450SE        16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL        17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC       15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

3.8.2.2 Bind columns

```
mtcars_mpg_cyl <- select(mtcars, mpg, cyl)
mtcars_disp_hp_drat <- select(mtcars, hp, drat, wt)
```

These two dataframes have the same number of rows, therefore their columns can be bound:

```
bind_cols(mtcars_disp_hp_drat, mtcars_mpg_cyl)
```

```
##                    hp drat    wt  mpg cyl
## Mazda RX4         110 3.90 2.620 21.0   6
## Mazda RX4 Wag     110 3.90 2.875 21.0   6
## Datsun 710         93 3.85 2.320 22.8   4
## Hornet 4 Drive    110 3.08 3.215 21.4   6
## Hornet Sportabout 175 3.15 3.440 18.7   8
## Valiant           105 2.76 3.460 18.1   6
## Duster 360        245 3.21 3.570 14.3   8
## Merc 240D          62 3.69 3.190 24.4   4
## Merc 230           95 3.92 3.150 22.8   4
```

```
## Merc 280            123 3.92 3.440 19.2   6
## Merc 280C           123 3.92 3.440 17.8   6
## Merc 450SE          180 3.07 4.070 16.4   8
## Merc 450SL          180 3.07 3.730 17.3   8
## Merc 450SLC         180 3.07 3.780 15.2   8
## Cadillac Fleetwood  205 2.93 5.250 10.4   8
## Lincoln Continental 215 3.00 5.424 10.4   8
## Chrysler Imperial   230 3.23 5.345 14.7   8
## Fiat 128             66 4.08 2.200 32.4   4
## Honda Civic          52 4.93 1.615 30.4   4
## Toyota Corolla       65 4.22 1.835 33.9   4
## Toyota Corona        97 3.70 2.465 21.5   4
## Dodge Challenger    150 2.76 3.520 15.5   8
## AMC Javelin         150 3.15 3.435 15.2   8
## Camaro Z28          245 3.73 3.840 13.3   8
## Pontiac Firebird    175 3.08 3.845 19.2   8
## Fiat X1-9            66 4.08 1.935 27.3   4
## Porsche 914-2        91 4.43 2.140 26.0   4
## Lotus Europa        113 3.77 1.513 30.4   4
## Ford Pantera L      264 4.22 3.170 15.8   8
## Ferrari Dino        175 3.62 2.770 19.7   6
## Maserati Bora       335 3.54 3.570 15.0   8
## Volvo 142E          109 4.11 2.780 21.4   4
```

Chapter 4

Tidyr

```
library(tidyverse)
library(lubridate)
library(ggplot2)
```

```
# DATA and FIRST EXPLORATIONS
covid_raw <- readRDS("data/covid-ita-regions.Rds")

# Several Covid-19 indicators by date and region - Italy
# Quick data check
glimpse(covid_raw)
```

```
## Rows: 7,812
## Columns: 14
## $ tests       <int> 58, 89, 114, 141, 169, 189, 243, 269, 354, 376, 397, 577, ~
## $ confirmed   <int> 0, 0, 0, 0, 0, 0, 6, 9, 13, 18, 21, 31, 42, 57, 93, 116, 1~
## $ recovered   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 10, 11, 11~
## $ deaths      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 3, 6, 8, 10, ~
## $ hosp        <int> NA, NA, NA, NA, NA, NA, NA, NA, 1, 3, 4, 4, 8, 8, 19, 33, ~
## $ vent        <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ icu         <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 1, 1, 1, 6~
## $ population  <int> 1215538, 1215538, 1215538, 1215538, 1215538, 1215538, 1215~
## $ country     <chr> "ITA", "ITA", "ITA", "ITA", "ITA", "ITA", "ITA", "ITA", "I~
## $ region_name <chr> "Friuli Venezia Giulia", "Friuli Venezia Giulia", "Friuli ~
## $ lat         <dbl> 45.64944, 45.64944, 45.64944, 45.64944, 45.64944, 45.64944~
## $ long        <dbl> 13.76814, 13.76814, 13.76814, 13.76814, 13.76814, 13.76814~
## $ region_code <chr> "Friuli-Venezia Giulia", "Friuli-Venezia Giulia", "Friuli-~
## $ time        <chr> "2020-02-24 08:00:00", "2020-02-25 08:00:00", "2020-02-26 ~
```

4.1 Pivoting

4.1.1 Pivot Longer

```
# Dataset in long form: better for analysis and plotting with ggplot2
# In covid_raw we have several stats in different columns,
# Goal: have a row for each statistics

covid_tbl <- read_csv("data/covid-ita-regions.csv")
glimpse(covid_tbl)
```

```
## Rows: 7,812
## Columns: 14
## $ tests       <dbl> 58, 89, 114, 141, 169, 189, 243, 269, 354, 376, 397, 577, ~
## $ confirmed   <dbl> 0, 0, 0, 0, 0, 0, 6, 9, 13, 18, 21, 31, 42, 57, 93, 116, 1~
## $ recovered   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 3, 3, 3, 10, 11, 11~
## $ deaths      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 3, 6, 8, 10, ~
## $ hosp        <dbl> NA, NA, NA, NA, NA, NA, NA, NA, 1, 3, 4, 4, 8, 8, 19, 33, ~
## $ vent        <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA~
## $ icu         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 1, 1, 1, 6~
## $ population  <dbl> 1215538, 1215538, 1215538, 1215538, 1215538, 1215538, 1215~
## $ country     <chr> "ITA", "ITA", "ITA", "ITA", "ITA", "ITA", "ITA", "ITA", "I~
## $ region_name <chr> "Friuli Venezia Giulia", "Friuli Venezia Giulia", "Friuli ~
## $ lat         <dbl> 45.64944, 45.64944, 45.64944, 45.64944, 45.64944, 45.64944~
## $ long        <dbl> 13.76814, 13.76814, 13.76814, 13.76814, 13.76814, 13.76814~
## $ region_code <chr> "Friuli-Venezia Giulia", "Friuli-Venezia Giulia", "Friuli-~
## $ time        <dttm> 2020-02-24 08:00:00, 2020-02-25 08:00:00, 2020-02-26 08:0~
```

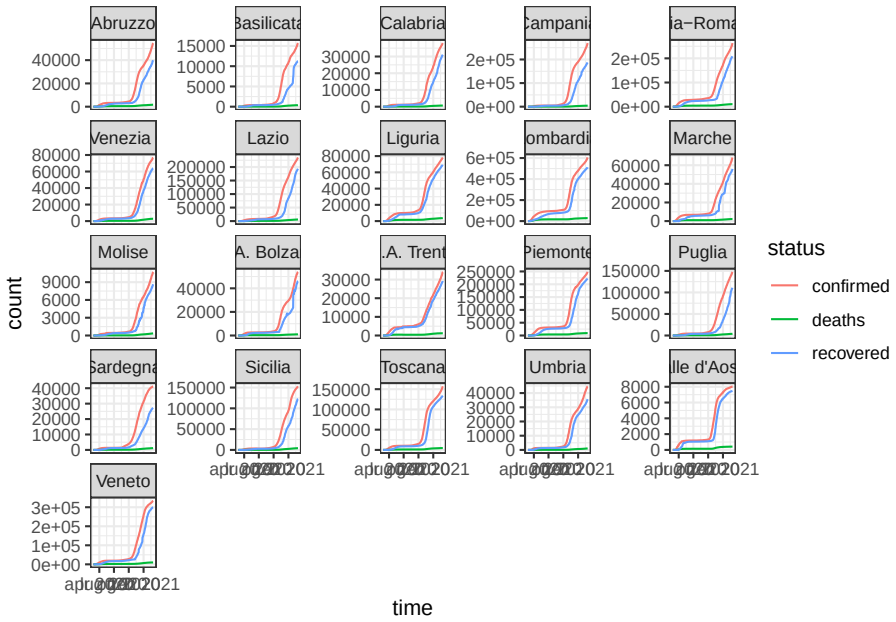
```
# Focus on covid-19 status
covid_status <- covid_tbl %>%
  select(time, region = region_name, confirmed, recovered, deaths) %>%
  arrange(time, region)

# Long version
status_long <- covid_status %>%
  pivot_longer(-c(time, region), names_to = "status", values_to = "count")

status_long
```

```
## # A tibble: 23,436 x 4
##    time                region     status    count
##    <dttm>              <chr>      <chr>     <dbl>
##  1 2020-02-24 08:00:00 Abruzzo    confirmed     0
##  2 2020-02-24 08:00:00 Abruzzo    recovered     0
##  3 2020-02-24 08:00:00 Abruzzo    deaths        0
##  4 2020-02-24 08:00:00 Basilicata confirmed     0
##  5 2020-02-24 08:00:00 Basilicata recovered     0
```

```
##  6 2020-02-24 08:00:00 Basilicata deaths        0
##  7 2020-02-24 08:00:00 Calabria   confirmed     0
##  8 2020-02-24 08:00:00 Calabria   recovered     0
##  9 2020-02-24 08:00:00 Calabria   deaths        0
## 10 2020-02-24 08:00:00 Campania   confirmed     0
## # ... with 23,426 more rows
```

The long version is sutable to be used with `ggplot` or `dplyr`:

```
# Long version is easier to handle - Plot demo
status_long %>%
  ggplot(aes(x = time, y = count, col = status)) +
  geom_line() +
  facet_wrap(. ~ region, scales = "free_y") +
  theme_bw()
```



4.1.2 Pivot Wider

```
# Dataset in wide form: better for data presentation
status_wide <- status_long %>%
  pivot_wider(names_from = status, values_from = count)

status_wide
```

```
## # A tibble: 7,812 x 5
##    time                region                confirmed recovered deaths
```

```
##    <dttm>              <chr>                    <dbl>     <dbl>  <dbl>
##  1 2020-02-24 08:00:00 Abruzzo                      0         0      0
##  2 2020-02-24 08:00:00 Basilicata                   0         0      0
##  3 2020-02-24 08:00:00 Calabria                     0         0      0
##  4 2020-02-24 08:00:00 Campania                     0         0      0
##  5 2020-02-24 08:00:00 Emilia-Romagna              18         0      0
##  6 2020-02-24 08:00:00 Friuli Venezia Giulia        0         0      0
##  7 2020-02-24 08:00:00 Lazio                        3         1      0
##  8 2020-02-24 08:00:00 Liguria                      0         0      0
##  9 2020-02-24 08:00:00 Lombardia                  172         0      6
## 10 2020-02-24 08:00:00 Marche                       0         0      0
## # ... with 7,802 more rows
```

```
# Demo
covid_tbl %>%
  mutate(month = month(time, label = T)) %>%
  group_by(month, region_name) %>%
  summarise(hosp = sum(hosp)) %>%
  pivot_wider(names_from = month, values_from = hosp)
```

```
## # A tibble: 21 x 13
##    region_name     gen    feb    mar    apr    mag   giu   lug   ago   set   ott
##    <chr>         <dbl>  <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Abruzzo       14948     NA   5626  11226   6580  2118   605   855  1394  5849
##  2 Basilicata     2738     NA     NA   1978    960    NA    NA    NA   235  1422
##  3 Calabria       9115     NA     NA   4757   1888   453   118   335   904  2292
##  4 Campania      46979     NA   8489  19226  11582  2399  1065  1783  9271 28514
##  5 Emilia-Roma~  84974  61589  61017 107977  36731  6462  2879  2674  5299 17385
##  6 Friuli Vene~  22514     NA     NA   5594   2772   608   215   359   663  2410
##  7 Lazio         93590     NA  16244  45161  38341 10780  6164  7262 14260 38217
##  8 Liguria       22410     NA  16355  32046  13126  2900   953   711  3726 14679
##  9 Lombardia    122225 115113 217884 345238 158514 56993  6287  5385  8934 44426
## 10 Marche        19202     NA  19399  28352   7057   814   139   382   608  3745
## # ... with 11 more rows, and 2 more variables: nov <dbl>, dic <dbl>
```

4.2 Separate and Unite

4.2.1 Separate

Let us introduce an example dataframe where the column `sex_and_age` contains two distinct variables `sex` and `age`. In this format it is not possible, for example, to group by sex or calculate an average age.

```
# SEPARATE different values stored in a sigle column
people <- tibble(
  id = 1:3,
  sex_and_age = c("F/23", "M/24", "F/22")
```

```
)
people
```

```
## # A tibble: 3 x 2
##      id sex_and_age
##   <int> <chr>
## 1     1 F/23
## 2     2 M/24
## 3     3 F/22
```

With `separate` it is possible to split a character variable in to its components:

```
people %>%
  separate(
    sex_and_age,
    into = c("sex", "age"),
    sep = "/"
  )
```

```
## # A tibble: 3 x 3
##      id sex   age
##   <int> <chr> <chr>
## 1     1 F     23
## 2     2 M     24
## 3     3 F     22
```

4.2.2 Unite

The opposite operation is achieved with `unite` that gather multiple vars into a single character.

```
# select useful column only
small_flights <- flights %>%
  select(year, month, day, origin, dest)

# unite a date into a single string (suitable to be used by `lubridate`)
small_flights %>%
  unite(year, month, day, col = "date", sep = "-")
```

```
## # A tibble: 336,776 x 3
##    date     origin dest
##    <chr>    <chr>  <chr>
##  1 2013-1-1 EWR    IAH
##  2 2013-1-1 LGA    IAH
##  3 2013-1-1 JFK    MIA
##  4 2013-1-1 JFK    BQN
##  5 2013-1-1 LGA    ATL
##  6 2013-1-1 EWR    ORD
```

```
##  7 2013-1-1 EWR    FLL
##  8 2013-1-1 LGA    IAD
##  9 2013-1-1 JFK    MCO
## 10 2013-1-1 LGA    ORD
## # ... with 336,766 more rows
```

Chapter 5

Lubridate

```
library(tidyverse)
library(lubridate)
library(ggplot2)
```

5.1 Overview

Date-time data can be frustrating to work with in R. R commands for date-times are generally unintuitive and change depending on the type of date-time object being used. Moreover, the methods we use with date-times must be robust to time zones, leap days, daylight savings times, and other time related quirks, and R lacks these capabilities in some situations. Lubridate makes it easier to do the things R does with date-times and possible to do the things R does not.

If you are new to lubridate, the best place to start is the date and times chapter in R for data science.

Source: https://lubridate.tidyverse.org/

```
library(tidyverse)
library(lubridate)
library(ggplot2)




# DATA and FIRST EXPLORATIONS ---------------------------------------------

# covid_raw <- readRDS("data/covid-ita-regioni.csv")
covid_raw <- readRDS("data/covid-ita-regions.Rds")
```

```
# Several Covid-19 indicators by date and region - Italy

# Quick data check
# glimpse(covid_raw)
# summary(covid_raw)
```

5.2 Parsing date and time

Usually dates and timestamps (date + day-time) are found in data in a wide variety of string formats, lubridate can convert very different date formats into date/timestamp format:

```
# Extreme example from lubridate package help
x <- c(20100101120101,
       "2009-01-02 12-01-02",
       "2009.01.03 12:01:03",
       "2009-1-4 12-1-4",
       "2009-1, 5 12:1, 5",
       "200901-08 1201-08",
       "2009 arbitrary 1 non-decimal 6 chars 12 in between 1 !!! 6",
       "OR collapsed formats: 20090107 120107 (as long as prefixed with zeros)",
       "Automatic wday, Thu, detection, 10-01-10 10:01:10 and p format: AM",
       "Created on 10-01-11 at 10:01:11 PM")
ymd_hms(x)
```

```
##  [1] "2010-01-01 12:01:01 UTC" "2009-01-02 12:01:02 UTC"
##  [3] "2009-01-03 12:01:03 UTC" "2009-01-04 12:01:04 UTC"
##  [5] "2009-01-05 12:01:05 UTC" "2009-01-08 12:01:08 UTC"
##  [7] "2009-01-06 12:01:06 UTC" "2009-01-07 12:01:07 UTC"
##  [9] "2010-01-10 10:01:10 UTC" "2010-01-11 22:01:11 UTC"
```

`ymd_hms` it a function name acronim of Year, Month, Day, Hour, Minute, Second and it uses digits (from the input string) in this order to output a time object.

An example of conversion to a date format:

```
# Parsing mechanism
x <- c("23/12/2013", "07/10/2014")
dmy(x)
```

```
## [1] "2013-12-23" "2014-10-07"
```

If the information has different granularity the `truncated` param must be specified:

```
# Handling truncation
x <- c("2011-12-31 12:59:59", "2010-01-01 12:11", "2010-01-01 12", "2010-01-01")
```

```
ymd_hms(x, truncated = 3) #up to 3 elements may be missing
```

```
## [1] "2011-12-31 12:59:59 UTC" "2010-01-01 12:11:00 UTC"
## [3] "2010-01-01 12:00:00 UTC" "2010-01-01 00:00:00 UTC"
```

```
# Application on Covid data
covid_time <- covid_raw %>%
  select(region = region_name, time) %>%
  mutate(datetime = ymd_hms(time))

# Remark: If we don't specify the time zone, UTC is assumed
covid_time
```

```
## # A tibble: 7,812 x 3
##    region                time                datetime
##    <chr>                 <chr>               <dttm>
##  1 Friuli Venezia Giulia 2020-02-24 08:00:00 2020-02-24 08:00:00
##  2 Friuli Venezia Giulia 2020-02-25 08:00:00 2020-02-25 08:00:00
##  3 Friuli Venezia Giulia 2020-02-26 08:00:00 2020-02-26 08:00:00
##  4 Friuli Venezia Giulia 2020-02-27 08:00:00 2020-02-27 08:00:00
##  5 Friuli Venezia Giulia 2020-02-28 08:00:00 2020-02-28 08:00:00
##  6 Friuli Venezia Giulia 2020-02-29 08:00:00 2020-02-29 08:00:00
##  7 Friuli Venezia Giulia 2020-03-01 08:00:00 2020-03-01 08:00:00
##  8 Friuli Venezia Giulia 2020-03-02 08:00:00 2020-03-02 08:00:00
##  9 Friuli Venezia Giulia 2020-03-03 08:00:00 2020-03-03 08:00:00
## 10 Friuli Venezia Giulia 2020-03-04 08:00:00 2020-03-04 08:00:00
## # ... with 7,802 more rows
```

5.3 Working with time-zones

5.3.1 Introduction

Consider timestamps in the Covid data set to be in the "CET" timezone (Central European Time). To interpret the `time` string in to the correct time zone the param `tz` must be specified.

```
# CONVERT TIME ZONES
covid_time %>%
  mutate(datetime_tz = ymd_hms(time, tz = "CET"),
         chicago_time = with_tz(datetime_tz, "America/Chicago"),
         chicago_time_wrong = with_tz(datetime, "America/Chicago"))
```

```
## # A tibble: 7,812 x 6
##    region    time    datetime            datetime_tz         chicago_time
##    <chr>     <chr>   <dttm>              <dttm>              <dttm>
##  1 Friuli V~ 2020-0~ 2020-02-24 08:00:00 2020-02-24 08:00:00 2020-02-24 01:00:00
##  2 Friuli V~ 2020-0~ 2020-02-25 08:00:00 2020-02-25 08:00:00 2020-02-25 01:00:00
```

```
##  3 Friuli V~ 2020-0~ 2020-02-26 08:00:00 2020-02-26 08:00:00 2020-02-26 01:00:00
##  4 Friuli V~ 2020-0~ 2020-02-27 08:00:00 2020-02-27 08:00:00 2020-02-27 01:00:00
##  5 Friuli V~ 2020-0~ 2020-02-28 08:00:00 2020-02-28 08:00:00 2020-02-28 01:00:00
##  6 Friuli V~ 2020-0~ 2020-02-29 08:00:00 2020-02-29 08:00:00 2020-02-29 01:00:00
##  7 Friuli V~ 2020-0~ 2020-03-01 08:00:00 2020-03-01 08:00:00 2020-03-01 01:00:00
##  8 Friuli V~ 2020-0~ 2020-03-02 08:00:00 2020-03-02 08:00:00 2020-03-02 01:00:00
##  9 Friuli V~ 2020-0~ 2020-03-03 08:00:00 2020-03-03 08:00:00 2020-03-03 01:00:00
## 10 Friuli V~ 2020-0~ 2020-03-04 08:00:00 2020-03-04 08:00:00 2020-03-04 01:00:00
## # ... with 7,802 more rows, and 1 more variable: chicago_time_wrong <dttm>
```

```
# `chicago_time_wrong` is wrong, since it assumes we are in UTC timezone
```

5.3.2 Accepted time-zones names

The list of accepted time-zones is returned by this function:

```
# OlsonNames() # see help for more
head(OlsonNames(), n = 10)
```

```
##  [1] "Africa/Abidjan"     "Africa/Accra"       "Africa/Addis_Ababa"
##  [4] "Africa/Algiers"     "Africa/Asmara"      "Africa/Asmera"
##  [7] "Africa/Bamako"      "Africa/Bangui"      "Africa/Banjul"
## [10] "Africa/Bissau"
```

5.3.3 Fix time-zone of a timestamp

The function `force_tz` set a different timezone using the same time, it does not do the conversion, it only sets a different timezone.

```
# FIX TIME-ZONES
# We can fix the initial mistake forcing the time-zone
covid_time <- covid_time %>%
  mutate(datetime = force_tz(datetime, tzone = "Europe/Rome"))
```

5.4 Setting and extracting info from a date and time objects

```
covid_time %>%
  mutate(date = date(datetime),
         hour = hour(datetime),
         weekday = wday(date, label = T),
         month = month(date, label = T))
```

```
## # A tibble: 7,812 x 7
##    region         time        datetime            date        hour weekday month
##    <chr>          <chr>       <dttm>              <date>     <int> <ord>   <ord>
```

```
##  1 Friuli Venezi~ 2020-02-24~ 2020-02-24 08:00:00 2020-02-24     8 lun     feb
##  2 Friuli Venezi~ 2020-02-25~ 2020-02-25 08:00:00 2020-02-25     8 mar     feb
##  3 Friuli Venezi~ 2020-02-26~ 2020-02-26 08:00:00 2020-02-26     8 mer     feb
##  4 Friuli Venezi~ 2020-02-27~ 2020-02-27 08:00:00 2020-02-27     8 gio     feb
##  5 Friuli Venezi~ 2020-02-28~ 2020-02-28 08:00:00 2020-02-28     8 ven     feb
##  6 Friuli Venezi~ 2020-02-29~ 2020-02-29 08:00:00 2020-02-29     8 sab     feb
##  7 Friuli Venezi~ 2020-03-01~ 2020-03-01 08:00:00 2020-03-01     8 dom     mar
##  8 Friuli Venezi~ 2020-03-02~ 2020-03-02 08:00:00 2020-03-02     8 lun     mar
##  9 Friuli Venezi~ 2020-03-03~ 2020-03-03 08:00:00 2020-03-03     8 mar     mar
## 10 Friuli Venezi~ 2020-03-04~ 2020-03-04 08:00:00 2020-03-04     8 mer     mar
## # ... with 7,802 more rows
```

5.5 Time span and arithmetic

5.5.1 Time span

```
# INTERVAL: time span between specific dates
covid_phase <- covid_time %>%
  mutate(today = today(tzone = "CET"),
         start_phase2 = ymd("2020-05-04", tz = "CET"),
         phase2 = interval(start_phase2, today),
         in_phase2 = datetime %within% phase2)

# Interval object have several helpers: int_start(), int_end(), int_shift() ...
# See interval() help for more
```

5.5.2 Duration

```
# DURATIONS: time duration in seconds
# Duration helper functions are named after the units of time, but start with a d
dminutes(3) # length of 2 minutes
```

```
## [1] "180s (~3 minutes)"
```

```
class(dminutes(3))
```

```
## [1] "Duration"
## attr(,"package")
## [1] "lubridate"
```

5.5.3 Period

```
# PERIODS: time duration, adjusted to be more readable
# Period helper functions are named after the units of time, but plural
minutes(3)
```

```
## [1] "3M 0S"
```

```
class(minutes(3))
```

```
## [1] "Period"
## attr(,"package")
## [1] "lubridate"
```

5.5.4 Difference between Period and Duration

```
# You can use duration and periods to do date arithmetic
ymd(20110101) + years(1) # More intuitive
```

```
## [1] "2012-01-01"
```

```
ymd(20110101) + dyears(1) # More precise
```

```
## [1] "2012-01-01 06:00:00 UTC"
```

```
# Operation with dates
covid_phase <- covid_phase %>%
  mutate(yesterday = today - days(1),
         two_weeks_later = datetime + weeks(2),
         length_phase2_d = phase2/days(1), # length in days
         length_phase2_w = phase2/weeks(1) # length in weeks
  )

# Months arithmetic is not so obvious.
# Try to be sure about what you want

# If adding or subtracting a month or a year creates an invalid date, lubridate will r
mar31 <- ymd(20190331)
mar31 + months(1)
```

```
## [1] NA
```

```
# %m+% AND %m-% OPERATORS
# this avoid to outputs NAs
# If necessary, they roll back dates to the last day of the month
mar31 %m+% months(1)
```

```
## [1] "2019-04-30"
```

```
mar31 %m-% months(1)
```

```
## [1] "2019-02-28"
```

```
# If you want to add 31 days, then it's still different:
mar31 + days(31)
```

```
## [1] "2019-05-01"
```

5.6 Rounding dates

```
covid_phase <- covid_phase %>%
  mutate(date_month = floor_date(datetime, unit = "month"))

# Useful for summary stats

# grouping variable with lubridate ----------------------------------------

# In order to use a grouping variable of the month, you can create a string
# (concatenation of year and month), but in a more useful way you can use the
# computed last day of the month.

covid_time %>%
  select(region, datetime) %>%
  # mutate(mese = paste(year(datetime), month(datetime), sep = "-"))
  mutate(date_month = ceiling_date(datetime, unit = "month") - days(1))
```

```
## # A tibble: 7,812 x 3
##    region                 datetime            date_month
##    <chr>                  <dttm>              <dttm>
##  1 Friuli Venezia Giulia 2020-02-24 08:00:00 2020-02-29 00:00:00
##  2 Friuli Venezia Giulia 2020-02-25 08:00:00 2020-02-29 00:00:00
##  3 Friuli Venezia Giulia 2020-02-26 08:00:00 2020-02-29 00:00:00
##  4 Friuli Venezia Giulia 2020-02-27 08:00:00 2020-02-29 00:00:00
##  5 Friuli Venezia Giulia 2020-02-28 08:00:00 2020-02-29 00:00:00
##  6 Friuli Venezia Giulia 2020-02-29 08:00:00 2020-02-29 00:00:00
##  7 Friuli Venezia Giulia 2020-03-01 08:00:00 2020-03-31 00:00:00
##  8 Friuli Venezia Giulia 2020-03-02 08:00:00 2020-03-31 00:00:00
##  9 Friuli Venezia Giulia 2020-03-03 08:00:00 2020-03-31 00:00:00
## 10 Friuli Venezia Giulia 2020-03-04 08:00:00 2020-03-31 00:00:00
## # ... with 7,802 more rows
```

5.7 Print time objects (convert to strings)

5.7.1 Example print a nice label for a unique week of the year

```
datetime <- ymd_hms("2011-12-31 12:59:59")

format(datetime, "%Y-W%U")  # 2020-W01 Year-Week
```

```
## [1] "2011-W52"
```

```
# see help(strptime) for the %codes
# dates help(format.Date),
# date-times help(format.POSIXct)
```

5.7.2 Using stamp

`stamp` help to create a format starting from an example string:

```
# 1. Derive a template, create a function
# Tip: use a date with day > 12
example_date <- "Created Sunday, Jan 17, 1999 3:34"
sf <- stamp(example_date)

# 2. sf is a function to apply the template to dates
sf(ymd("2010-04-05"))
```

```
## [1] "Created Sunday, 04 05, 2010 00:00"
```

```
#> [1] "Created Monday, Apr 05, 2010 00:00"
```

Chapter 6

Git and Github

6.1 Git setup

Set up your commit signature globally (for all repositories):

```
# Customize your git signature,
# the copy, paste and execute on Terminal

git config --global user.name "Andrea Melloncelli"
git config --global user.email andrea@vanlog.it
```

See also: https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

6.2 Add a remote to your locally created repository

Prerequisite: create a new RStudio project with git, with at least one commit.

In any empty github repository web page you can find these instructions.

Advice: select HTTPS protocol.

```
# or push an existing repository from the command line

# just in case, remove 'origin' if it already exists
git remote rm origin

git remote add origin https://github.com/andreamelloncelli/R-for-data-science-2022-collaboration.
git branch -M main
git push -u origin main
```

You can verify the correct remote's configuration:

```
git remote -v
#> origin  https://github.com/andreamelloncelli/R-for-data-science-2022-collaboration.
#> origin  https://github.com/andreamelloncelli/R-for-data-science-2022-collaboration.
```

6.3 Other CLI Git Operations

CLI means Command Line Interface, which is the "Command Prompt" on Windows, the Bash on Linux and MacOS.

```
# get the status
git status
# upload the latest modifications
git push
# download the latest modifications
git pull
# history
git log
# go to the branch main
git checkout main
# create a new branch
git branch work-in-progress
# move on that branch
git checkout work-in-progress
```

6.4 Troubleshooting via CLI

6.4.1 Pull

Remark: `pull` is the equivalent of the sequence of fetch and merge operations:

```
# download the latest modifications, without using them
git fetch
# merge you "main" branch with the remote version ("origin/main")
git merge origin/main
```

Therefore if a conflict happens during a `git pull`, it is a merging conflict. You can abort the merge falling back to your version of main with:

```
git merge --abort
```

or decide to use others version of a file with:

```
git checkout --theirs path/to/file
git add path/to/file
git merge --continue
```

or manually fix the conflict and then commit.

See this post for more information.

If you want to lose your latest commits (not only the working directory changes) and use the upstream version (origin/main version) you can revert to the HEAD version:

```
# backup your main branch
git branch main_bak
# update to 'origin/main'
git fetch
# BEWARE THIS IS A DESTRUCTIVE OPERATION IF YOU DID NOT MAKE
# THE 'main_bak' branch correctly:
git reset --hard origin/main
```

To find your old version:

```
git checkout main_bak
```

6.5 References

- Configure RStudio and Git guide: https://www.geo.uzh.ch/microsite/reproducible_research/post/rr-rstudio-git/
- Learn git commands used by RStudio: https://rviews.rstudio.com/2020/04/23/10-commands-to-get-started-with-git/
- Git cheatsheet
- Learn git visually: http://git-school.github.io/visualizing-git/
- Other resources: https://try.github.io/
- Git Flow: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Chapter 7

Create a Package

7.1 Create a project package

In RStudio:

- "New Project…"
- "New folder"
- "R Package"
- then, finish the guided procedure.

7.2 Setup Script

Create a new buffer file, copy and paste this script, and execute it step by step.

7.9.1

7.3 Roxygen

This guide assumes that you use Roxygen for the package. Even it is not strictly necessary it helps in keeping a tidy project with its automated procedures.

The main purpose of Roxygen is to manage:

- the `NAMESPACE` file. This file is responsible of what external packages and functions are visible by your package ("import") and what functions of your package are visible to the end user ("export").
- the `man` folder. It contains the documentation of your package (".md" files) and make them available as the RStudio guide to the end user.

See more at https://r-pkgs.org/man.html

7.4 Create Library Functions

7.4.1 Library

Library functions are the core of your package. They are on the ".R" files in your "R/" folder. How to organize them in files is up to you, it will not affect the R behaviour.

7.4.2 First exported function

```
#' my_division
#'
#' This function is an example of pure function. it is exported: the user of the
#' package will be able to use it. It does not need to import packages: it does
#' not require any other packages to operates.
#'
#' @param x [numeric] numerator
#' @param y [numeric] denominator
#'
#' @return [numeric] result of the division
#' @export
#'
my_division <- function(x, y) {
  x / y
}
```

7.4.3 Functions that depends on some libraries

This example shows a function that needs other packages to work:

```
#' average_mpg
#'
#' This function is an example of pure function, that requires a package to
#' operates: dplyr in this case
#'
#' @param tbl
#'
#' @export
#' @import dplyr
average_mpg <- function(tbl) {
  tbl %>%
    group_by(cyl) %>%
    summarise(mean_mpg = mean(mpg))
}
```

In this case all functions of `dplyr` have been imported. We could import only the necessary functions with the `@importFrom` statement to import from `dplyr`

only the functions `select` and `%>%`:

```
#' @importFrom dplyr select `%>%`
```

7.5 Use Unit Tests with 'testthat'

7.5.1 Unit Tests

A Unit Test is meant to be a small procedure to check a single function. You can run unit tests all together with "CTRL + SHIFT + t" keyboard shortcut. In these procedures you are supposed to call a function, providing some suitable inputs, and expect something in the result.

`testthat` is the package that allows you to run the tests. It provide the `test_that` function as well as several `expect_*` functions.

7.5.2 Create a test file

The `usethis` library provide a shortcut to create a test file in a standard way. For example in order to test the functions in the `R/my-file.R` you can call this function:

```
usethis::use_test('my-file')
```

that creates a file in `tests/testthat/test-my-file.R` with an example test. You can delete the example and start implementing your test.

7.5.3 Implemente a new test

`test_that` is the function that contains a unit test. It has two argoments: (1) the test label and (2) a procedure to test the function. The test ends with an `expect_equal` statement to check the result.

Trivial test checking itself:

```
test_that("division works", {
  expect_equal(10 / 3,  # it returns 3.333333333333333
               3.3,
               tolerance = 0.1)
})
```

Real test:

```
test_that("division works", {
  my_result <- my_division(10, 3)

  expect_equal(my_result,
               3.3,
```

```
                tolerance = 0.1)
})
```

where `my_division` could be:

```
my_division <- function(x, y) {
  x / y
}
```

Note that the `testthat` package provides a number of `expect_*` function to check the result in a different way.

7.5.4 Implement a test with saved result

This test will compare the result of `average_mpg(mtcars)` with the object saved in `average_mpg_works.Rds`. Use `update = TRUE` to overwrite the saved value (see `help(expect_known_value)`)

```
test_that("average_mpg works", {

  expect_known_value(average_mpg(mtcars),
                     file = '../../tests/testthat/average_mpg_works.Rds',
                     update = FALSE)

})
```

7.6 Use S3

S3 is a framework to implement a Object Oriented Programming logic in R.

7.6.1 Create a factory

```
#' cool_obj
#'
#' @param a numeric
#' @param b numeric
#'
#' @return cool_obj
#' @export
#'
#' @examples
cool_obj <- function(a, b) {
  obj <- list(a = a, b = b)
  class(obj) <- 'cool_obj'
  obj
}
```

7.6.2 Implement an already existing method

```
#' cool_obj
#'
#' @param co cool_obj
#'
#' @return NULL
#' @export
#'
#' @examples
print.cool_obj <- function(co) {
  cat('A: ', co$a)
  cat('\n')
  cat('B: ', co$b)
}
```

7.6.3 create a brand new method for your object

Create the new method:

```
#' cool_obj
#'
#' @param co cool_obj
#' @param ... other non implemented parameters
#'
#' @return cool_obj
#' @export
#'
cool_switch <- function(co, ...) {
  UseMethod('cool_switch')
}
```

Implement that method for your `cool_obj`:

```
#' cool_obj
#'
#' @param co cool_obj
#' @param ... other non implemented parameters
#'
#' @return cool_obj
#' @export
#'
cool_switch.cool_obj <- function(co, ...) {
  cool_obj(
    a = co$b,
    b = co$a
  )
```

```
}
```

7.7 Provide example dataset

7.7.1 Create the creation procedure

Use a template:

```
usethis::use_data_raw( name = "my_dataset", open = FALSE )
```

This function will create the `data-raw/my_dataset.R` file. This file is meant to be the procedure to create your dataset. You must modify this file in function of your needs. For example:

```
## code to prepare `my_dataset` dataset goes here
library(tibble)

my_dataset <- as_tibble(rownames_to_column(mtcars, var = 'car'))

usethis::use_data(my_dataset)
```

```
Setting active project to '/home/daco/dev/myPackage'
Adding 'R' to Depends field in DESCRIPTION
Creating 'data/'
Saving 'my_dataset' to 'data/my_dataset.rda'
Document your data (see 'https://r-pkgs.org/data.html')
```

Once you are done, running this script your data file `data/my_dataset.rda` will be created.

7.7.2 LazyData

This is an option in the DESCRIPTION file:

```
LazyData: true
```

7.7.3 Use the dataset

Now building and installing your package this dataset will be available:

```
library(myPackage)
my_dataset
```

7.7.4 Document your datasets

Create `R/data.R` file (or choose a different name), then create a documentation for the name of the dataset. Example:

```
#' Tibble mtcars
#'
#' A dataset containing mtcars data as a tibble.
#'
#' @format A tibble with 32 rows and 12 variables:
#' \describe{
#'   \item{car}{car model name}
#'   \item{mpg}{miles per gallon}
#'   \item{cyl}{number of cylinders}
#'   ...
#' }
#' @source \url{http://www.xxxxxxxxx.info/}
"my_dataset"
```

Once you have built and installed your package, `help(my_dataset)` will call a document with this information.

See https://r-pkgs.org/data.html#documenting-data.

7.8 Provide a vignette

7.8.1 Create a vignette

Use a template:

```
## Vignette
usethis::use_vignette("ThisTidyPackage")
```

This will create the `vignette` folder and inside the `ThisTidyPackage.Rmd` file. This file is meant to be a usage example case of your package.

Now you can build and install your package with:

```
devtools::install(build_vignettes = TRUE)
```

And look at your vignette with this command:

```
vignette("ThisTidyPackage")
```

7.8.2 Provide package with vignette as a file

In case you want to share the package you can provide it with vignette:

```
# create docs and vignettes
devtools::document(roclets = c('rd', 'collate', 'namespace', 'vignette'))
# create a source file package: '../myShapes_0.0.0.9000.tar.gz'
devtools::build()
```

```
# then you can install it from a file
install.packages('../myShapes_0.0.0.9000.tar.gz', repos = NULL)
```

7.8.3 Install the package with vignettes from a Github repository:

```
remotes::install_github('andreamelloncelli/myShapes',
                        force = T,
                        build_vignettes = T)
```

If you are asked to update some package you can refuse.

7.9 Appendix

7.9.1 Setup Script

Create a new buffer file, copy and paste this script, and execute it step by step.

```
## Create a new package with RStudio

# Package setup ---------------------------------------------------------------

## Use version control
# install.packages("usethis")
usethis::use_git_config(
  scope = "project",
  user.name = "John Doe",
  user.email = "john@example.org"
)
usethis::use_git()

# avoid problem with the dev scripts: dev/package-utility.R (this file)
dir.create("dev")
# save this file in `dev` as `setup.R`
usethis::use_build_ignore("dev")
# now you can save or move this file in "dev"

# Fill in the DESCRIPTION file
# rstudioapi::navigateToFile( "DESCRIPTION" )
usethis::use_description(
  list(
    Title = "App title",
    `Authors@R` = "person('Andrea', 'Melloncelli', email = 'andrea@vanlog.it', role = c
    Description = "A sentence describing the package.",
    URL = "https://github.com/vanlog/R-for-data-science-2020"
```

```
  )
)
usethis::use_lgpl_license()       # You can set another license here
usethis::use_tidy_description()   # sort fields and packages

## Common tasks
usethis::use_readme_md( open = FALSE )
# usethis::use_code_of_conduct()
# usethis::use_lifecycle_badge( "Experimental" )
# usethis::use_news_md( open = FALSE )


## Use tests: if you want to use tests
# usethis::use_testthat()
# installed.packages("devtools")


# Develop ----------------------------------------------------------------


## Add a package
usethis::use_package( "dplyr" )
# remember to add it to ROXYGEN or NAMESPACE:
#' @import dplyr  # ROXYGEN
#' import(dplyr)  # NAMESPACE

## If you want to use roxygen, enable ROXYGEN in the project.
# Menu: tools > Project options > build tools > generate the documentation with roxygen
usethis::use_namespace(roxygen = TRUE)
devtools::document() # to fill NAMESPACE and documentation with ROXYGEN comments
# or roxygen2::roxygenise() # converts roxygen comments to .Rd files.
# or [Ctrl + Shift + D] in RStudio

## Build or load
# Load the package [CTRL + SHIFT + L] or install-and-reload [CTRL + SHIFT + B]

## Check the package for Cran or [CTRL + SHIFT + E]
devtools::check(document = FALSE) # check the package

## Add internal datasets
## If you want to provide data along with your package
usethis::use_data_raw( name = "my_dataset", open = FALSE )

## Tests
## Add one line by test you want to create
```

```
usethis::use_test( "hello" )

## Vignette
usethis::use_vignette("ThisTidyPackage")
devtools::build_vignettes()
# Install the package and see it with `vignette("ThisTidyPackage")`
# List your vignettes: vignette(package = 'cancRFDS')
# Install your package manually: devtools::install(build_vignettes = TRUE)

# Deploy ------------------------------------------------------------------

# devtools::missing_s3()
#
# devtools::check()
# rhub::check_for_cran()
```

7.9.2 Shortcuts

- `CTRL + SHIFT + b`: build, install the package, restart R, load the package
- `CTRL + SHIFT + l`: quickly load the package (skip installation and restart)
- `CTRL + SHIFT + t`: run tests

7.9.3 Referencies

- Cheatsheet

Chapter 8

Parallel Programming

8.1 Premise

This manual is compiled with little data. In order to see the time gain of using parallel programming rise the number of `points`

8.2 Introduction

`rnd_couple` is a function that generates a point in the square: a couple (x,y) where each coordinates is randomly extracted from a uniform distribution between (-1, 1). Applying the `rnd_couple` n times you get n random points.

```
n <- 10^3 #10^5
```

```
library(purrr)

rnd_couple <- function(x) {
  c(x = runif(1, min = -1, max = 1),
    y = runif(1, min = -1, max = 1))
}

# generates 'n' random points.
points <- lapply(seq_len(n), rnd_couple)
```

Now to the number of points inside the circle. Applying the `in_cicle` function to a point it returns `TRUE` (which counts 1 in a sum) if the point is inside the circle, `FALSE` (which counts 0 in a sum) otherwise. Given the total number of points inside the circle (`count`) and the total number of points (`total`), the function `counter_to_pi` returns the value of Pi.

```
in_circle <- function(point) {
  point[['x']]^2 + point[['y']]^2 < 1
}

counter_to_pi <- function(count, total) {
  count/total * 4
}
```

8.3 The algorithm

8.3.1 Sequential

This for loop counts the points inside the circle in variable `counter`. But the second operation in each step is non parallelizable, because each step depends on the previous one in the value of `counter`.

```
counter <- 0
for (point_idx in seq_along(points)) {
  point <- points[[point_idx]]            # it would be parallelizable
  counter <- counter + in_circle(point)   # non parallelizable: each steps depend on t
}
counter_to_pi(counter, n)
```

8.3.2 Split the algorithm

In order to parallelize the algorithm, it is necessary to split the for loop in two. The first loop in the following code is parallelizable, but still is running sequentially because the `for` statement is always sequential. The second has other requirements to be parallelizable (see below the specific section).

```
## ideally a map or lapply
are_points_in_circle <- numeric(n)
for (point_idx in seq_along(points)) {
  point <- points[[point_idx]]
  are_points_in_circle[[point_idx]] <- in_circle(point)
}

## ideally a reduce
counter <- 0
for (is_point_in_circle in are_points_in_circle) {
  counter <- counter + is_point_in_circle
}
counter_to_pi(counter, n)
```

8.3.3 For to map

Here the two loops are re-written as map-reduce operations. `map` is very similar to `lapply`, both of them are sequential, but they have a multi-core counterpart.

```
are_points_in_circle <- map(points, in_circle)
counter <- reduce(are_points_in_circle, `+`)
counter_to_pi(counter, n)

# in a pipeline
map(points, in_circle) %>%
  reduce(`+`) %>%
  counter_to_pi( n)
```

8.4 Parallelize the map

8.4.1 mclapply

Here `lapply` is substituted by `mclapply` which does the same work but it runs in parallel.

```
# This is a simple case, you can appreciate more the effect of the parallelization
# with a longer function to be parallelized. In order to do that you can create
# a more time consuming function:
#
# heavier in_circle
in_circle_heavy <- function(point) {
  # just to make this function taking longer
  for (i in seq_len(300)) {
    point[['x']]^2 + point[['y']]^2 < 1
  }
  # the actual result
  point[['x']]^2 + point[['y']]^2 < 1
}
# substitute `in_circle` with the heavier one
in_circle <- in_circle_heavy
```

```
tic()
lapply(points, in_circle) %>%
  reduce(`+`) %>%
  counter_to_pi(n)
toc()

tic()
parallel::mclapply(points, in_circle, mc.cores = 4, mc.preschedule = T) %>%
  reduce(`+`) %>%
  counter_to_pi(n)
```

```
toc()
```

```
# install.packages("bench")
bench::mark(
  iterations = 3, memory = FALSE, check = FALSE, filter_gc = FALSE,
  lapply = lapply(points, in_circle),
  mclapply = parallel::mclapply(points, in_circle, mc.preschedule = T, mc.cores = 4)
)
```

8.5 doSNOW Montecarlo

8.5.1 doSNOW Library

This is a library which is more complex than parallel and is fully compatible with Linux, Windows and MacOS systems.

First of all, we need to create and register a cluster. `cluster` is the object that holds the information about what type of infrastructure will execute the code. When you are done you need to shut down this cluster with `stopCluster`.

`parLapply` is the `mcalpply` equivalent for doSNOW. It takes the `cluster` object as argument.

8.5.2 Single core example

```
library(doSNOW)
```

Create the cluster with `1` core:

```
cluster <- makeCluster(1, type = "SOCK")
registerDoSNOW(cluster)

# Some useful information about the cluster
getDoParWorkers()
getDoParRegistered()
getDoParName()
getDoParVersion()

tic()
# like parallel::mclapply(). mc.preshedule=T seems to be the default here.
snow::parLapply(cl = cluster,
                x = points,
                fun = in_circle) %>%
  reduce(`+`) %>%
  counter_to_pi(n)
toc()
```

```
stopCluster(cluster)
```

8.5.3 Multi core example

We are comparing the time of `lapply` (sequential) with `parLapply` (multi-core). The function `parallel::detectCores()` return the number of CPUs available on this hardware.

```
# library(doSNOW)

n_cpus <- parallel::detectCores()
cluster <-  makeCluster(n_cpus, type = "SOCK")
registerDoSNOW(cluster)


# sequential
tic()
lapply(points, in_circle) %>%
  reduce(`+`) %>%
  counter_to_pi( n)
toc()

tic()
# like mclapply(). mc.preshedule=T seems to be the default here.
snow::parLapply(cl = cluster,
          x = points,
          fun = in_circle) %>%
  reduce(`+`) %>%
  counter_to_pi( n)
toc()


stopCluster(cluster)
```

8.6 Parallelize the reduce operation

It is possible parallelize the reduce operation in case the reduce operation is associative (see slides).

In order to do that two nested map-reduce cicles will be used.

Let us define some useful functions:

```
# apply in_circle and reduce
# inner map-reduce level: sequential
in_circle_and_reduce <- function(points) {
```

```
  lapply(points, in_circle) %>%
    reduce(`+`)
}
```

8.6.1 Split the dataset

With the function `snow::splitList`, you can split the long `points` list in to a number of pieces equals to the number or cluster CPUs you want to use. Each one of these groups contains an almost equal part of the total and it will feed a single sequential process. These groups will be run in parallel.

```
point_groups <- snow::splitList(points, 3)
str(point_groups, max.level = 1)
```

8.6.2 2-level map-reduce

Let us now create a single computer cluster with all the CPUs available:

```
# library(doSNOW)

n_cpus <- parallel::detectCores()
cluster <- makeCluster(n_cpus, type = "SOCK")
registerDoSNOW(cluster)
```

Now the cluster is running `n_cpus` new R processes. You needs to export to them the definition of the function you have in your environment. You can use the `clusterExport` function providing a list of strings with function names. NB: in case you re-define a function you will need to re-export it.

```
# export dependencies in cluster
clusterExport(cluster, list("in_circle", "reduce", "%>%"))
```

```
tic()
point_groups <- splitList(points, n_cpus)
# outer map-reduce level: parallel
snow::parLapply(cl = cluster,
                x = point_groups,
                fun = in_circle_and_reduce) %>%
  reduce(`+`) %>%
  counter_to_pi(n)
toc()
```

```
stopCluster(cluster)
```

8.7 Vector function to optimize the reduce operation

Some binary functions have the correspondent n-ary function or vector function (that takes a vector):

```
# binary function
`+`(1,2)
# this will NOT work
# `+`(1,2,3)

# n-ary function
sum(1,2,3)
# vector function
sum(c(1,2,3))
```

Therefore the reduce operation can be done more efficiently:

```
tic()
lapply(points, in_circle) %>%
  unlist() %>%      # list to vector
  sum() %>%         # vector reducer: usually faster than reduce(...)
  counter_to_pi(n)
toc()
```

8.8 Confilcts between `parallel` and `doSNOW`

`parallel` and `doSNOW` provide a set of functions with the same name, but different specifications. It is better to load only one of the two packages at a time.

Here an example:

```
help('parLapply', package = 'parallel') # see argument `x`
help('parLapply', package = 'snow') # see argument `x` and `X`
```

Chapter 9

Code Optimization

9.1 Libraries

9.1.1 Libraries needed in this section

```
library(dplyr)
library(purrr)
library(readr)
library(readxl)
library(ggplot2)
```

9.1.2 Useful functions

Let us create some useful functions to plot the result of the benchmarks we are going to do.

```
## Useful functions
# Plot a benchmark
show_bm <- function(bm) {
  print(print_bench(bm))
  autoplot(bm)
}
# printable bench (for RMarkdown)
print_bench <- function(bm) {
  bm %>%
    mutate(expression = as.character(expression))
}
```

9.1.3 Function call timing

Calling a function takes time. Here we compare the execution of a calculation against `1+1`, against the execution of the same calculation by a function `g` and against the call of a empty function `f`.

```
# empty function
f <- function() {
  NULL
}

# sum function
g <- function(x) {
  x + 1
}

bench::mark(
  check = F,
  bare = NULL,
  add = 1+1,
  f = f(),
  g = g(1)
) %>%
  show_bm()
```

Remarks:

1. comparing the timing of `f` and of `g` you see that the sum itself takes a small amount of time compared to the time due of an empty function call.

9.1.4 Vectorization

Vectorization allows you to write code simpler to read, but also more effective. The reason of the effectiveness is that the vector function is a "primitive' function written in a lower level language (i.e.: C, C++ or Fortran) more optimized than any R code you can write.

```
size <- 10^6 #10^2 # 10^5

v1 <- runif(size)
v2 <- runif(size)

bench::mark(
  loop_for = {
    v3 <- NULL
    for (i in seq_len(length(v1))) {
      v3[i] <-v1[i] + v2[i]
    }
```

```
    v3
  },
  loop_for_size = {
    v3 <- numeric(size)
    for (i in seq_len(length(v1))) {
      v3[i] <- v1[i] + v2[i]
    }
    v3
  },
  vector = v3 <- v1 + v2
) %>% show_bm()

# Explanation examples
# `[`(v3, 1) == v3[1]
# c(1,2) + 1
# c(1,2) + c(2,3,2)
```

Remarks:

1. Pre allocating a vector (see in `loop_for_size`) is a good strategy to reduce the overhead due to the increasing of the dimension of the vector. In other words, every time you add an element to a vector, R allocates a new vector and copy all its content in the new container, this time loss is avoided if you pre-allocate a vector of the final size.

9.1.5 Vectorization counterexample

```
bench::mark(
  mean_1 = mean(v1),
  mean_2 = sum(v1) / length(v1)
)
```

There are some counterexample in which a vectorized operation is slower. In this case, the difference is due to the implementation of `mean` that probably to be numerically more precise does a double run.

9.1.6 vector or lists

Here a comparison of similar operations performed on vector, matrices and list.

```
point_df <- tibble(
  x = runif(1000),
  y = runif(1000)
)

point_list <- unname(split(point_df, 1:1000))
```

```
point_matrix   <- as.matrix(point_df)
point_t_matrix <- t(point_matrix)

bench::mark(
  vec = with(point_df, x^2 + y^2 < 1),
  tbl = pull(transmute(point_df, x^2 + y^2 < 1)),
  list = sapply(point_list,
                USE.NAMES = FALSE,
                function(point) point$x^2 + point$y^2 < 1 ),
  matrix_1 = apply(point_matrix, 1, function(point) sum(point^2) < 1),
  matrix_2 = apply(point_t_matrix, 2, function(point) sum(point^2) < 1),
  # check = FALSE
) %>%
  show_bm()
```

Remarks:

1. vector operation on vectors are faster than on lists.
2. the other timings present similar, but different operations so that you can see the order of magnitude of the times.

9.2 Input and output

9.2.1 Why

Also I/O operations take different times and can be optimized. In particular, reading/writing a file takes time depending on the format of the file, and on if it compress the data. Also when you read a file that does not contain information about the type of the variables, like a csv, the operation of inferring that type takes time. Therefore if you specify the type of the columns you will get a more efficient reading.

9.2.2 benchmark setup

```
# I/O files
# Try with different sizes

rds <- tempfile("ds", fileext = ".Rds")
plain_rds <- tempfile("plain_ds", fileext = ".Rds")
rdata <- tempfile("ds", fileext = ".Rdata")
csv <- tempfile("ds", fileext = ".csv")
# xlsx<- tempfile("ds", fileext = ".xlsx")

size <- 10^2 # 10^5
```

```
ds <- tibble(
  x = rnorm(size),
  y = rnorm(size),
  z = rnorm(size)
)
```

9.2.3 Writing benchmark

```
bench::mark(
  check = F,
  rds = saveRDS(ds, rds),
  plain_rds = saveRDS(ds, plain_rds, compress = F),
  rdata = save(ds, file = rdata),
  csv   = write_csv(ds, csv)
  # xlsx::write.xlsx(ds, xlsx)
) %>%
  show_bm()
```

Remarks:

1. Writing a csv is slower than writing a compressed rds.
2. Writing a compressed rds is slower than a plain rds.

9.2.4 Reading benchmark

```
bench::mark(
  check = F,
  rds = readRDS(rds),
  plain_rds = readRDS(plain_rds),
  rdata = load(rdata),
  # csv_old = read.csv(csv),
  csv = read_csv(csv),
  csv_schema = read_csv(csv, col_types = cols(
    x = col_double(),
    y = col_double(),
    z = col_double()
  ))
  # xlsx = read_xlsx(xlsx)
) %>%
  show_bm()

file.remove(rds, plain_rds, rdata, csv)
```

Remarks:

1. Writing remarks stand also for the reading

2. Reading a csv with a given schema is faster than reading inferring the types

9.3 Existing solutions

9.3.1 Sort partial

It is often a good practice to search if somebody has already solved an issue. Here `sort` has a parameter to sort only a number of elements removing the necessity to sort the whole vector. In this example it performs ten times better.

```
# install.packages("microbenchmark")
# install.packages("ggbeeswarm")
library(ggplot2)
x <- rnorm(100000)
bench::mark(
  sort(x, partial=1:10)[1:10],
  sort(x)[1:10]
) %>%
  show_bm()
```

9.4 OOP

9.4.1 Benchmark setup

```
# OOP methods overhead ----------------------------------------------------

f <- function(x) NULL

s3 <- function(x) UseMethod("s3")
s3.integer <- function(x) NULL

A <- setClass("A", representation(a = "list"))
setGeneric("s4", function(x) standardGeneric("s4"))
setMethod(s4, "A", function(x) NULL)

B <- setRefClass("B")
B$methods(r5 = function(x) NULL)

C <- R6::R6Class("B", public = list(
  r6 = function(x) NULL
)
)

a <- A()
```

```
b <- B$new()
c <- C$new()
```

9.4.2 OOP Benchmark

Object Oriented Programming framework do a great job allowing to write tidy, more mantainable code, but this involve runtime tasks, like dispatch, that have a cost.

```
library(microbenchmark)
options(digits = 3)
microbenchmark(
  bare = NULL,
  fun = f(),
  s3 = s3(1L),
  s4 = s4(a),
  r5 = b$r5(),
  r6 = c$r6(),
  # rcpp =      call_r_fun(2L, 3, somma)
  times = 10000
)
# Unit: nanoseconds
#  expr  min    lq    mean median    uq     max neval
#  bare    9    33    44.4     37    52    8597 10000
#   fun  169   231   311.0    268   316   18768 10000
#    s3  887  1209  1823.8   1411  1610 1615297 10000
#    s4 8739 10354 11857.7  11012 11882 1706266 10000
#    r5 6778  7746  9252.5   8491  9159 1675410 10000
#    r6 1272  1650  2071.9   1960  2223   21858 10000
```

9.5 Copy-on-modify

9.5.1 Definition

Copy a data structure is a task expansive as a matter of time ad used memory. R avoid useless copies with copy-on-modify paradigm. This means that when you copy a variable it is not doing the actual copy: it simply uses two labels to address a single value. When you modify one of the two variables the actual copy is performed.

9.5.2 Example on vectors

```
# big size
size <- 10^8
```

```
# big vector
v1 <- runif(size)

# copy of the vector: the used memory is unchanged.
v2 <- v1

# modifying a copy of the vector the actual copy begins and more memory is used.
v1[2] <- 10

# the value is changed
v1[2]
# the copy has the original value
v2[2]

# Nothing happens removing
rm(v2)
# Memory is actually freed calling the Garbage Collector
gc()

# Same for v1
rm(v1)
gc()
```

Chapter 10

Montecarlo Optimization

10.1 Libraries

```
library(tidyverse)
library(tictoc)
# library(parallel)
library(dplyr)
library(tictoc)
```

Useful functions:

```
# Plot a benchmark
show_bm <- function(bm) {
  print(print_bench(bm))
  autoplot(bm)
}
# printable bench (for RMarkdown)
print_bench <- function(bm) {
  bm %>%
    mutate(expression = as.character(expression))
}
```

Functions for the Montecarlo:

```
# pi / 4 = count / total
counter_to_pi <- function(count, total) {
  count/total * 4
}

in_circle <- function(unused) {
  x <- runif(1, min = -1, max = 1)
```

```
  y <- runif(1, min = -1, max = 1)
  x^2 + y^2 < 1
}

n <- 10^4
```

Sequential Montecarlo Map-reduce:

```
pi_map_reduce <- function(n) {
  map(seq_len(n), in_circle) %>%
    reduce(`+`) %>%
    counter_to_pi(n)
}
```

```
# vectorial in_circle version
vec_in_circle <- function(n) {
  x <- runif(n, min = -1, max = 1)
  y <- runif(n, min = -1, max = 1)
  x^2 + y^2 < 1
}

# vectorial Montecarlo
pi_vec <- function(n) {
  vec_in_circle(n) %>%
    sum() %>%
    counter_to_pi(n)
}
```

10.2 Vectorized Vs sequential

```
bench::mark(
  check = F,
  pi_map_reduce(n),
  pi_vec(n)
) %>%
  show_bm()
```

As you can see, a vectorized function can run a couple of order of magnitude faster the for-loop version.

10.3 Vectorized VS parallel

10.3.1 Functions

```
# parallel in_circle version
par_in_circle <- function(n) {
  parLapply(cl = cluster,
            x = seq_len(n),
            fun = in_circle)
}

# parallel Montecarlo
pi_parall <- function(n) {
  par_in_circle(n) %>%
    unlist() %>%
    sum() %>%
    counter_to_pi(n)
}
```

Two step map-reduce (parallel outer, sequential inner):

```
# Two step map-reduce (outer parallel, and vectorized the inner one)
pi_super_parall <- function(n) {

  workers <- getDoParWorkers()
  block_indexes <- seq_len(workers)
  block_n <- n/workers

  # Parall pi_vec(n/4) * 4 cores
  parLapply(cl = cluster,
            x = block_indexes,
            fun = function(indexes) sum({
              x <- runif(block_n, min = -1, max = 1)
              y <- runif(block_n, min = -1, max = 1)
              x^2 + y^2 < 1
            })) %>%
    unlist() %>%
    sum() %>%
    counter_to_pi(n)
}
```

the two-level map-reduce is slightly faster than sequential version. This is due to the overhead time due to the initialization of the parallelization.

NB: you see that the parallelization implementation is inside `pi_super_parall`, this is an example where the implementation of a function is in parallel and you do not see it from this external point of view.

```
library(doSNOW)
cluster <- makeCluster(parallel::detectCores(), type = "SOCK")
registerDoSNOW(cluster)

n <- 10^7

bench::mark(
  check = F, iterations = 10, filter_gc = F,
  pi_vec(n),
  pi_super_parall(n)
) %>%
  show_bm()

stopCluster(cluster)
```

```
# map-reduce explanation
#
# map:
# runif -> in_circle
#
# reduce:
# sum(is_point_in_circle)
#
# We did:
# map -> vectorization
# reduce -> parallel_reduce by the associative property
#
# Parallel reduce
# a + b + c + d + ...
# (a + b + ...) + (g + h + ...) + (l + m + ... ) + (p + q + ...)
```

10.3.2 Overhead due to the parallelization in detail

```
double <- function(x) {
  x*2
}

cluster <- makeCluster(parallel::detectCores(), type = "SOCK")
registerDoSNOW(cluster)

bench::mark(
  check = FALSE,
  par = parLapply(cl = cluster,
                  x = seq_len(4),
                  fun = double),
```

```
  vec = double(seq_len(4))
) %>% show_bm()

stopCluster(cluster)
```

10.4 Move to a lower level language to optimize to the maximum level

`Rcpp` is a package that helps you to write CPP code and use it with R. You can create function inside R or create CPP code files, compile them and use them in R. You can also create a package with a mixed code base.

Here you see a simple example where the Montecarlo has been implemented in CPP. You see that you need to fix the type for each variable (i.e.: `double sum = 0;`).

```
# https://adv-r.hadley.nz/rcpp.html

library(Rcpp)
library(microbenchmark)
library(bench)
library(tictoc)

# You'll also need a working C++ compiler. To get it:
#
# On Windows, install `Rtools`.
# On Mac, install `Xcode` from the app store.
# On Linux, `sudo apt-get install r-base-dev` or similar.

# simple ------------------------------------------------------------------

cppFunction('double pi_cpp(int n) {
  // srand(time(NULL));   // Initialization, should only be called once.
  double sum = 0;
  for (int i = 0; i< n; i++) {
    double x = double(rand())/RAND_MAX * 2. - 1.;      // Returns a pseudo-random integer betwee
    double y = double(rand())/double(RAND_MAX) * 2. - 1.;
    bool in_circle = x*x + y*y < 1;
    sum = sum + in_circle;
  }
  return sum/n * 4;
}')

n <- 10^5 #10^6
# tic()
```

```
pi_cpp(10000)
# toc()
```

```
bench::mark(
  check = F,
  pi_vec(n),
  pi_cpp(n)
) %>%
  show_bm()
```

You see that the CPP version is faster also of the vectorial one. Notice that is implemented with a for loop, we did the same implementation with R and it was terribly slower. This is an example of how much the speed can be different if same algorithm in implemented in R or in CPP.

Bibliography

Wickham, H. (2020). *Mastering Shiny*. O'Reilly, 1st edition.