

Basic data manipulation

Lecture 3

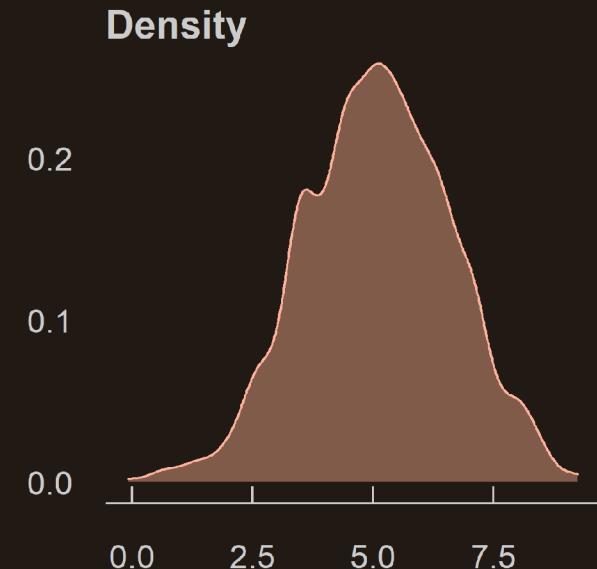
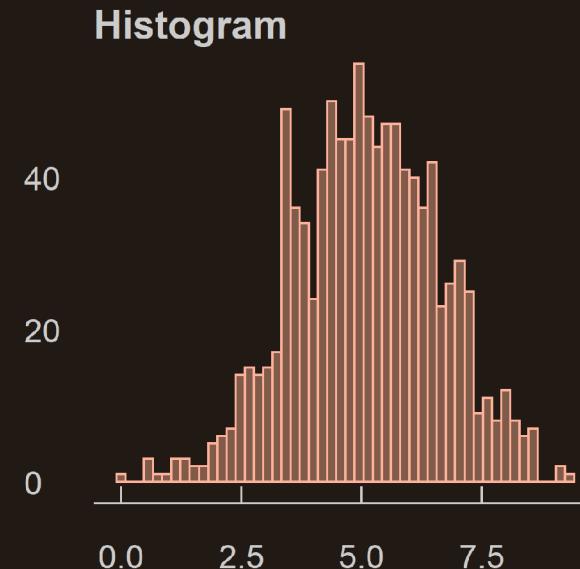
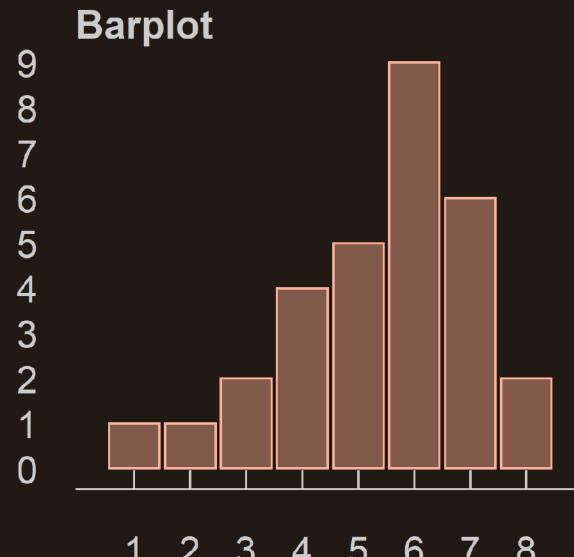
Louis SIRUGUE

CPES 2 - Fall 2022

Quick reminder

1. Distributions

- The **distribution** of a variable documents all its possible values and how frequent they are

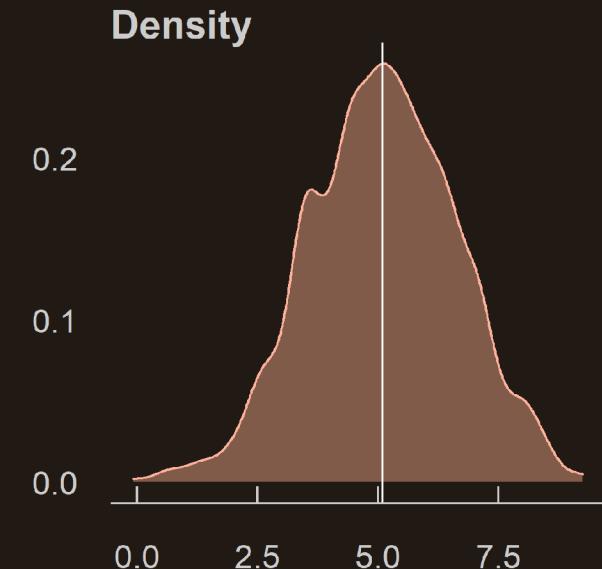
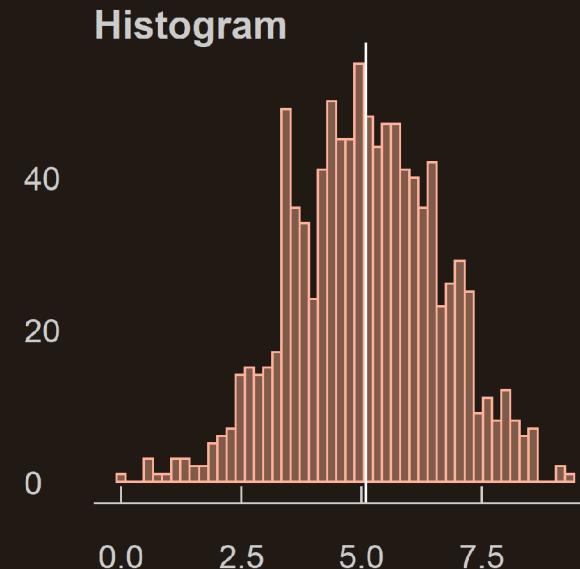
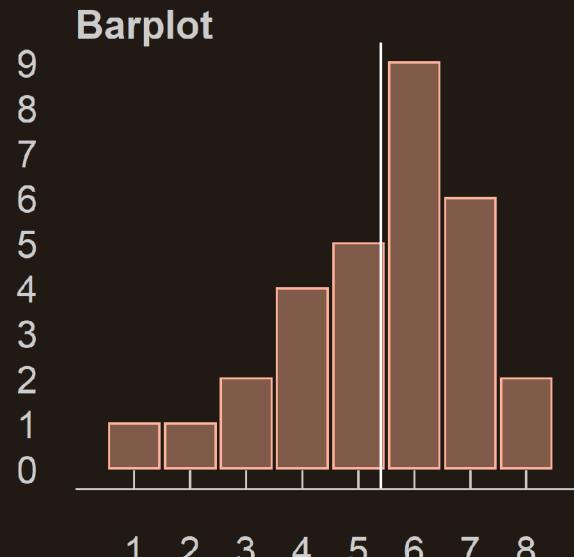


- We can describe a distribution with:

Quick reminder

1. Distributions

- The **distribution** of a variable documents all its possible values and how frequent they are

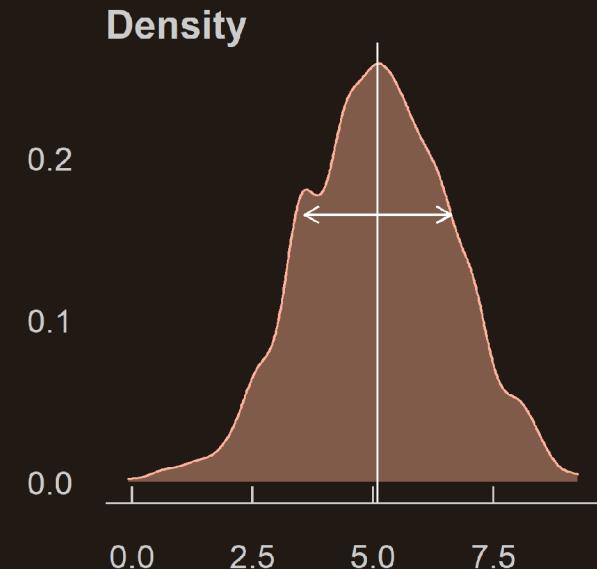
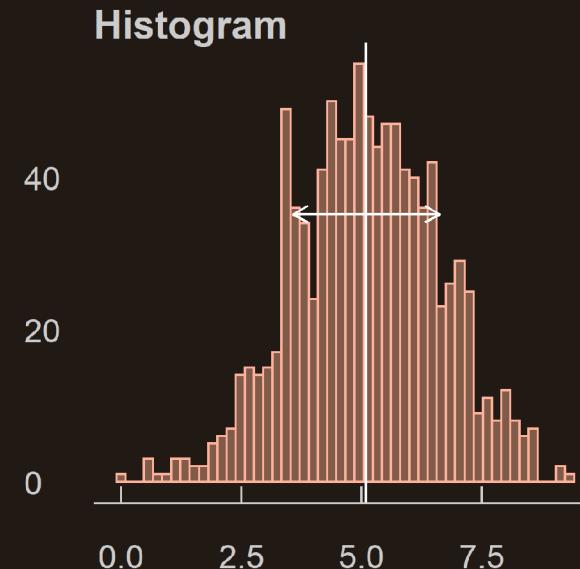
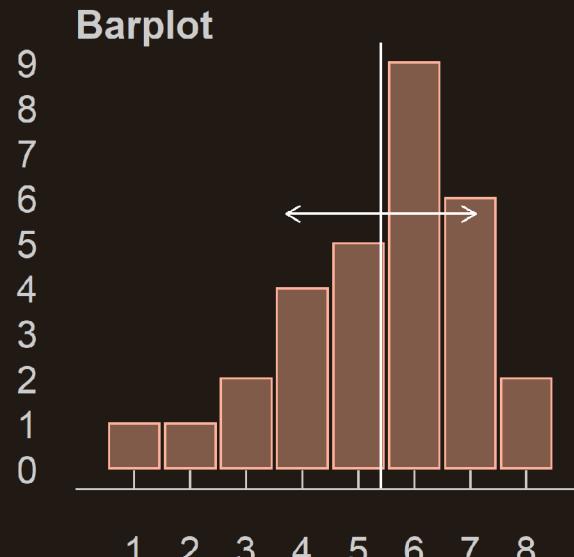


- We can describe a distribution with:
 - Its **central tendency**

Quick reminder

1. Distributions

- The **distribution** of a variable documents all its possible values and how frequent they are



- We can describe a distribution with:
 - Its **central tendency**
 - And its **spread**



Quick reminder

2. Central tendency

- The **mean** is the sum of all values divided by the number of observations

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$$

- The **median** is the value that divides the (sorted) distribution into two groups of equal size

$$\text{Med}(x) = \begin{cases} x\left[\frac{N+1}{2}\right] & \text{if } N \text{ is odd} \\ \frac{x\left[\frac{N}{2}\right] + x\left[\frac{N}{2} + 1\right]}{2} & \text{if } N \text{ is even} \end{cases}$$

3. Spread

- The **standard deviation** is square root of the average squared deviation from the mean

$$\text{SD}(x) = \sqrt{\text{Var}(x)} = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \bar{x})^2}$$

- The **interquartile range** is the difference between the maximum and the minimum value from the middle half of the distribution

$$\text{IQR} = Q_3 - Q_1$$

Quick reminder

4. Inference

- In Statistics, we view variables as a given realization of a **data generating process**
 - Hence, the **mean** is what we call an **empirical moment**, which is an **estimation**...
 - ... of the **expected value**, the **theoretical moment** of the DGP we're interested in
- To know how confident we can be in this estimation, we need to compute a **confidence interval**

$$\left[\bar{x} - t_{n-1, 97.5\%} \times \frac{\text{SD}(x)}{\sqrt{n}}; \bar{x} + t_{n-1, 97.5\%} \times \frac{\text{SD}(x)}{\sqrt{n}} \right]$$

- It gets **larger** as the **variance** of the distribution of x increases
- And gets **smaller** as the **sample size** n increases



Warm up practice

05 : 00

- 1) Import the `ligue1.csv` dataset and store it in an object called `fb`
- 2) Create a subset of this dataset containing only matches that took place at 13h
- 3) Print the number of matches in this subset and compute the average attendance
- 4) Redo the same exercise on matches that took place at 20h45

You've got 5 minutes!

Solution

1) Import the `ligue1.csv` dataset and store it in an object called `fb`

```
fb <- read.csv("C:/User/Documents/ligue1.csv", encoding = "UTF-8")
```

2) Create a subset of this dataset containing only matches that took place at 13h

```
sub13 <- fb[fb$Time == "13:00", ]
```

3) Print the number of matches in this subset and compute the average attendance

```
nrow(sub13)
```

```
## [1] 32
```

```
mean(sub13$Attendance)
```

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

Solution

- When there are **missing values** in a vector, the **mean** function returns **NA**
 - We need to set the **na.rm** option to **TRUE**

3) Print the number of matches in this subset and compute the average attendance

```
mean(sub13$Attendance, na.rm = T)
```

```
## [1] 19038
```

4) Redo the same exercise on matches that took place at 20h45

```
sub2045 <- fb[fb$Time == "20:45", ]  
nrow(sub2045)
```

```
## [1] 29
```

```
mean(sub2045$Attendance, na.rm = T)
```

```
## [1] 36418.64
```



Today we learn how to manipulate data

1. The dplyr package

- 1.1. Packages
- 1.2. Basic functions
- 1.3. group_by() and summarise()

3. A few words on learning R

- 3.1. When it doesn't work the way you want
- 3.2. Where to find help
- 3.3. When it doesn't work at all

2. Merge and reshape

- 2.1. Merge and append data
- 2.2. Reshape data

4. Wrap up!



Today we learn how to manipulate data

1. The dplyr package

- 1.1. Packages
- 1.2. Basic functions
- 1.3. group_by() and summarise()



1. The dplyr package

1.1. Packages

- So far we only used functions that are directly available in R
 - But there are tons of **user-created functions** out there that can make your life so much easier
 - These functions are shared in what we call **packages**
- Packages are **bundles of functions** that R users put at the disposal of other R users
 - Packages are **centralized** on the Comprehensive R Archive Network (CRAN)
 - To **download** and install a CRAN package you can simply use **install.packages()**
- All the functions of the dplyr grammar are gathered in the **dplyr package**
 - We can download these functions and make them ready to use with the `install.packages()` function

```
install.packages("dplyr") # Requires an internet connection
```

- The tidyverse package is **now installed** on your computer
 - You won't have to do it again



1. The dplyr package

1.1. Packages

- The `dplyr` package is now **on your computer**, but it is **not loaded in R**

```
ls("package:dplyr")
```

```
## Error in as.environment(pos): no item called "package:dplyr" on the search list
```

- You need to use the **library()** command to load it

```
library(dplyr)
ls("package:dplyr")[1:5]
```

```
## [1] "%>%"      "across"     "add_count"   "add_count_" "add_row"
```

- But even though the package is permanently installed, it is **loaded only for your current session**
 - Each time you start a **new R session**, you'll have to load the packages you need with **library()**



1. The dplyr package

1.2. Basic functions

dplyr is a **grammar** of data manipulation providing very **user-friendly functions** to handle the most common **data manipulation** tasks:

- `mutate()`: add/modify variables
- `select()`: keep/drop variables (columns)
- `filter()`: keep/drop observations (rows)
- `arrange()`: sort rows according to the values of given variable(s)
- `summarise()`: aggregate the data into descriptive statistics



- A very handy **operator** to use with the **dplyr** grammar is the **pipe %>%**
 - You can basically read **a %>% b()** as "*apply function b() to object a*"
 - With this operator you can easily **chain the operations** you apply to an object



1. The dplyr package

1.2. Basic functions



1. The dplyr package

1.2. Basic functions

```
fb %>%
  select(Home, xG, Score, xG.1, Away) # Keep/drop certain columns
# # # # #
```

```
##          Home   xG Score xG.1        Away
## 1      Monaco 2.0  1-1   0.3      Nantes
## 2       Lyon 1.4  1-1   0.8      Brest
## 3     Troyes 0.8  1-2   1.2    Paris S-G
## 4     Rennes 0.6  1-1   2.0      Lens
## 5   Bordeaux 0.7  0-2   3.3 Clermont Foot
## 6 Strasbourg 0.4  0-2   0.9      Angers
## 7      Nice 0.8  0-0   0.2      Reims
## 8 Saint-Étienne 2.1  1-1   1.3      Lorient
## 9      Metz 0.7  3-3   1.4      Lille
...      ... ...  ... ...      ...
```



1. The dplyr package

1.2. Basic functions

```
fb %>%
  select(Home, xG, Score, xG.1, Away) %>%
  mutate(home_winner = xG > xG.1) # Keep/drop certain columns
                                    # Create a new variable
                                    #
                                    #
                                    #
                                    #
                                    #
```

```
##          Home   xG Score xG.1          Away home_winner
## 1      Monaco 2.0  1-1  0.3      Nantes      TRUE
## 2       Lyon 1.4  1-1  0.8      Brest      TRUE
## 3     Troyes 0.8  1-2  1.2    Paris S-G     FALSE
## 4     Rennes 0.6  1-1  2.0       Lens     FALSE
## 5   Bordeaux 0.7  0-2  3.3 Clermont Foot     FALSE
## 6 Strasbourg 0.4  0-2  0.9      Angers     FALSE
## 7      Nice 0.8  0-0  0.2      Reims      TRUE
## 8 Saint-Étienne 2.1  1-1  1.3    Lorient      TRUE
## 9      Metz 0.7  3-3  1.4      Lille     FALSE
...      ... ... ... ...      ...      ...
```



1. The dplyr package

1.2. Basic functions

```
fb %>%
  select(Home, xG, Score, xG.1, Away) %>% # Keep/drop certain columns
  mutate(home_winner = xG > xG.1) %>% # Create a new variable
  filter(Home == "Rennes") # Keep/drop certain rows
  # 
  # 
  #
```

##	Home	xG	Score	xG.1	Away	home_winner
## 1	Rennes	0.6	1-1	2.0	Lens	FALSE
## 2	Rennes	0.9	1-0	0.5	Nantes	TRUE
## 3	Rennes	1.0	0-2	0.5	Reims	TRUE
## 4	Rennes	2.4	6-0	0.3	Clermont Foot	TRUE
## 5	Rennes	0.8	2-0	1.4	Paris S-G	FALSE
## 6	Rennes	1.5	1-0	0.6	Strasbourg	TRUE
## 7	Rennes	3.8	4-1	1.1	Lyon	TRUE
## 8	Rennes	3.1	2-0	0.7	Montpellier	TRUE
## 9	Rennes	0.8	1-2	0.6	Lille	TRUE



1. The dplyr package

1.2. Basic functions

```
fb %>%
  select(Home, xG, Score, xG.1, Away) %>%          # Keep/drop certain columns
  mutate(home_winner = xG > xG.1) %>%            # Create a new variable
  filter(Home == "Rennes") %>%                      # Keep/drop certain rows
  arrange(-xG)                                         # Sort rows
  #
  #
```

```
##      Home   xG Score xG.1           Away home_winner
## 1  Rennes 3.8   4-1  1.1        Lyon      TRUE
## 2  Rennes 3.3   6-0  0.4    Bordeaux      TRUE
## 3  Rennes 3.3   6-1  0.9       Metz      TRUE
## 4  Rennes 3.1   2-0  0.7 Montpellier      TRUE
## 5  Rennes 2.7   2-0  0.3      Brest      TRUE
## 6  Rennes 2.6   4-1  0.4     Troyes      TRUE
## 7  Rennes 2.4   6-0  0.3 Clermont Foot      TRUE
## 8  Rennes 1.9   2-3  2.9     Monaco     FALSE
## 9  Rennes 1.7   2-0  0.3    Angers      TRUE
...   ...   ...   ...   ...   ...
```



1. The dplyr package

1.2. Basic functions

```
fb %>%
  select(Home, xG, Score, xG.1, Away) %>%
  mutate(home_winner = xG > xG.1) %>%
  filter(Home == "Rennes") %>%
  arrange(-xG) %>%
  summarise(expected_wins = mean(home_winner),
           expected_goals = sum(xG))
```

Keep/drop certain columns
Create a new variable
Keep/drop certain rows
Sort rows
Aggregate into statistics
#

```
##   expected_wins expected_goals
## 1      0.8421053          36.6
```



1. The dplyr package

1.2. Basic functions

- Here are two very **handy functions** to use within `mutate()`

ifelse

```
fb %>%
  select(Home, Attendance) %>%
  mutate(att_bin = ifelse(Attendance > 10000,
                         "Large",
                         "Low"))
) %>% head()
```

```
##           Home Attendance att_bin
## 1      Monaco       7500     Low
## 2       Lyon      29018   Large
## 3      Troyes     15248   Large
## 4      Rennes     22567   Large
## 5    Bordeaux     18748   Large
## 6 Strasbourg    23250   Large
```

case_when

```
fb %>%
  select(Home, xG, xG.1, Away) %>%
  mutate(xWin = case_when(xG > xG.1 ~ "Home",
                          xG == xG.1 ~ "Draw",
                          xG < xG.1 ~ "Away"))
) %>% head()
```

```
##           Home     xG     xG.1          Away xWin
## 1      Monaco  2.0    0.3        Nantes  Home
## 2       Lyon  1.4    0.8        Brest  Home
## 3      Troyes  0.8    1.2        Paris S-G  Away
## 4      Rennes  0.6    2.0        Lens  Away
## 5    Bordeaux  0.7    3.3 Clermont Foot  Away
## 6 Strasbourg  0.4    0.9        Angers  Away
```



1. The dplyr package

1.3. group_by() and summarise()

- With `group_by()` you can perform **computations separately** for the different **categories of a variable**

```
fb %>%
  select(Wk, Home, xG) %>%
  mutate(all.xG = mean(xG)) %>%
  head(10)
```

```
## #> #> #> #> #> #> #> #> #> #> #>
```

Wk	Home	xG	all.xG
1	Monaco	2.0	1.473421
1	Lyon	1.4	1.473421
1	Troyes	0.8	1.473421
1	Rennes	0.6	1.473421
1	Bordeaux	0.7	1.473421
1	Strasbourg	0.4	1.473421
1	Nice	0.8	1.473421
1	Saint-Étienne	2.1	1.473421
1	Metz	0.7	1.473421
1	Montpellier	0.5	1.473421

```
fb %>%
  select(Wk, Home, xG) %>%
  group_by(Home) %>%
  mutate(home.xG = mean(xG)) %>%
  head(6)
```

```
## #> #> #> #> #> #>
```

Wk	Home	xG	home.xG
1	Monaco	2	1.69
2	Lyon	1.4	2.07
3	Troyes	0.8	1.21
4	Rennes	0.6	1.93
5	Bordeaux	0.7	1.23
6	Strasbourg	0.4	1.73



1. The dplyr package

1.3. group_by() and summarise()

- It is particularly **useful with summarise()**
 - summarise keeps the grouping variable
 - and computes **statistics for each category**

```
fb %>%
  group_by(Wk) %>%
  summarise(n = n(),
            tot_xG = sum(xG)+sum(xG.1),
            avg_WG = tot_xG/n) %>%
head(4)
```

```
## # A tibble: 4 x 4
##       Wk     n  tot_xG  avg_WG
##   <int> <int>  <dbl>    <dbl>
## 1     1     10    23.4    2.34
## 2     2     10    26.6    2.66
## 3     3     10    25.7    2.57
## 4     4     10    30.4    3.04
```

mutate() ≠ summarise()

- **mutate()** takes an operation that converts:
 - **A vector into another vector**
- **summarise()** takes an operation that converts:
 - **A vector into a value**

Ungrouping

- **group_by()** applies to all subsequent operations
- To cancel its effect you must **ungroup()** the data

```
fb %>%
  group_by(Wk) %>%
  mutate(test = mean(xG)) %>%
  ungroup() %>%
  ...
```

Practice

10 : 00

1) Start from the `fb` dataset and keep only the variables `Home`, `Score` and `Away`

2) Use the `separate()` function from `tidyverse` to split the `Score` variable into `home_score` and `away_score`

```
data.frame(x = "a_b") %>%  
  separate(x, c("x", "y"), "_")
```

```
##   x y  
## 1 a b
```

3) Convert these two variables into numeric vectors

4) Create a variable named `winner` that takes the values `Home`, `Draw` and `Away` depending on the score

5) Use `group_by()` and `summarise()` to compute the percentage of draws, home wins and away wins

You've got 10 minutes!

Solution

1) Start from the `fb` dataset and keep only the variables `Home`, `Score` and `Away`

```
fb %>%
  select(Home, Score, Away) %>%
  head(2)
```

```
##      Home Score   Away
## 1 Monaco    1-1 Nantes
## 2 Lyon     1-1 Brest
```

2) Use the `separate()` function from `tidyverse` to split the `Score` variable into `home_score` and `away_score`

```
fb %>%
  select(Home, Score, Away) %>%
  separate(Score, c("home_score", "away_score"), "-") %>%
  head(2)
```

```
##      Home home_score away_score   Away
## 1 Monaco          1         1 Nantes
## 2 Lyon            1         1 Brest
```

Solution

3) Convert these two variables into numeric vectors

4) Create a variable named `winner` that takes the values `Home`, `Draw` and `Away` depending on the score

```
fb %>%
  select(Home, Score, Away) %>%
  separate(Score, c("home_score", "away_score"), "-") %>%
  mutate(home_score = as.numeric(home_score),
        away_score = as.numeric(away_score),
        winner = case_when(home_score < away_score ~ "Away",
                            home_score == away_score ~ "Draw",
                            home_score > away_score ~ "Home")) %>%
  head()
```

```
##          Home home_score away_score          Away winner
## 1    Monaco           1           1      Nantes   Draw
## 2     Lyon           1           1       Brest   Draw
## 3   Troyes           1           2     Paris S-G   Away
## 4   Rennes           1           1       Lens   Draw
## 5  Bordeaux           0           2  Clermont Foot   Away
## 6 Strasbourg           0           2      Angers   Away
```

Solution

5) Use `group_by()` and `summarise()` to compute the percentage of draws, home wins and away wins

```
fb %>%
  select(Home, Score, Away) %>%
  separate(Score, c("home_score", "away_score"), "-") %>%
  mutate(home_score = as.numeric(home_score),
        away_score = as.numeric(away_score),
        winner = case_when(home_score < away_score ~ "Away",
                            home_score == away_score ~ "Draw",
                            home_score > away_score ~ "Home")) %>%
  group_by(winner) %>%
  summarise(pct = 100 * (n() / nrow(fb)))
```

```
## # A tibble: 3 x 2
##   winner     pct
##   <chr>    <dbl>
## 1 Away      30.5
## 2 Draw      26.8
## 3 Home      42.6
```



Overview

1. The dplyr package ✓

- 1.1. Packages
- 1.2. Basic functions
- 1.3. group_by() and summarise()

2. Merge and reshape

- 2.1. Merge and append data
- 2.2. Reshape data

3. A few words on learning R

- 3.1. When it doesn't work the way you want
- 3.2. Where to find help
- 3.3. When it doesn't work at all

4. Wrap up!



Overview

1. The dplyr package ✓

- 1.1. Packages
- 1.2. Basic functions
- 1.3. group_by() and summarise()

2. Merge and reshape

- 2.1. Merge and append data
- 2.2. Reshape data



2. Merge and reshape

2.1. Merge and append data

- Research projects often imply to **combine data** from different sources
 - Either to **add observations** (append rows)
 - Either to **add variables** (merge columns)

Dataset 1 on attainment

country	year	share_tertiary
FRA	2015	44.68760
GBR	2015	49.94341
USA	2015	46.51771

2. Merge and reshape

2.1. Merge and append data

- Research projects often imply to **combine data** from different sources
 - Either to **add observations** (append rows)
 - Either to **add variables** (merge columns)

Dataset 1 on attainment

country	year	share_tertiary
FRA	2015	44.68760
GBR	2015	49.94341
USA	2015	46.51771



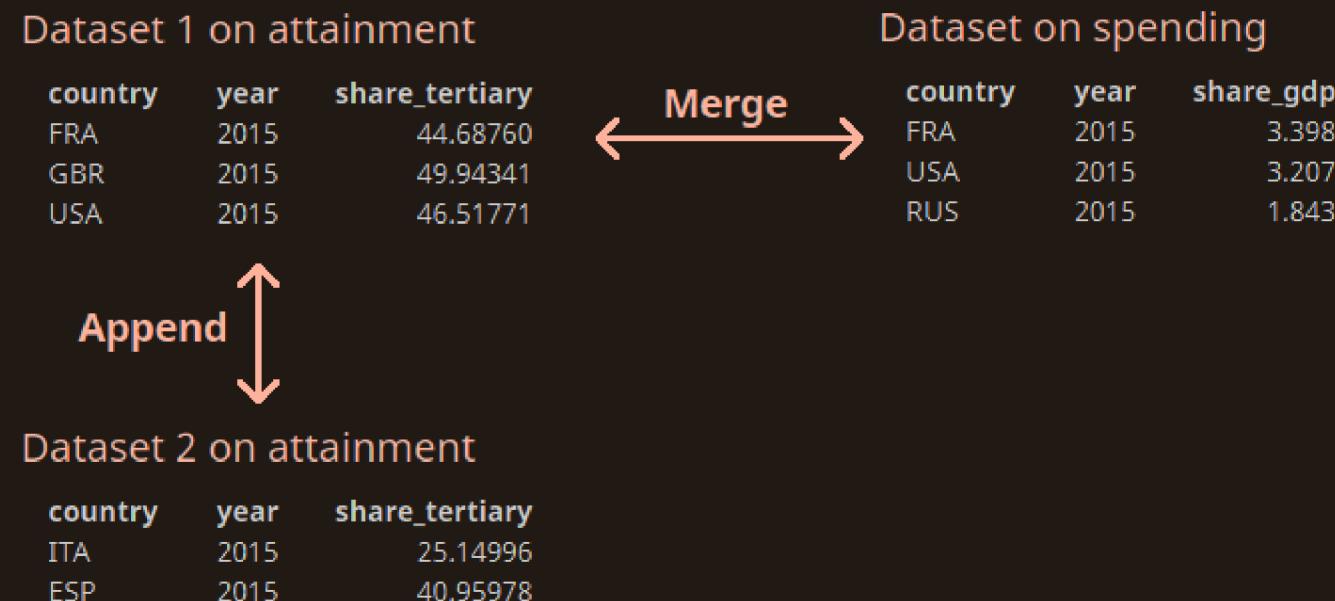
Dataset 2 on attainment

country	year	share_tertiary
ITA	2015	25.14996
ESP	2015	40.95978

2. Merge and reshape

2.1. Merge and append data

- Research projects often imply to **combine data** from different sources
 - Either to **add observations** (append rows)
 - Either to **add variables** (merge columns)





2. Merge and reshape

2.1. Merge and append data: The `bind_rows()` function

```
read.csv("attainment_FR_UK_US.csv")
```

```
##   country year share_tertiary
## 1     FRA 2015      44.68760
## 2     GBR 2015      49.94341
## 3     USA 2015      46.51771
```

```
read.csv("attainment_IT_SP.csv")
```

```
##   country year share_tertiary
## 1     ITA 2015      25.14996
## 2     ESP 2015      40.95978
```

```
attainment <- read.csv("attainment_FR_UK_US.csv") %>%
  bind_rows(read.csv("attainment_IT_SP.csv"))
attainment
```

```
##   country year share_tertiary
## 1     FRA 2015      44.68760
## 2     GBR 2015      49.94341
## 3     USA 2015      46.51771
## 4     ITA 2015      25.14996
## 5     ESP 2015      40.95978
```

Variables in the two datasets should be the same:

- **Same name**
- **Same class**



2. Merge and reshape

2.1. Merge and append data: `*_join()` functions

- Join functions all work the same way:
 - A **dataset A** with a **variable X** and other variables
 - A **dataset B** with a **variable X** and other variables
 - X is the common variable, so datasets will be **joined by X**

The 4 main join functions

Function	For X in A & B	For X in A only	For X in B only	Summary
A %>% left_join(B, by = "X")	Kept	Kept	Dropped	Only keeps what's in A
A %>% right_join(B, by = "X")	Kept	Dropped	Kept	Only keeps what's in B
A %>% inner_join(B, by = "X")	Kept	Dropped	Dropped	Only keeps what's common
A %>% full_join(B, by = "X")	Kept	Kept	Kept	Keeps everything



2. Merge and reshape

⚠ **Beware of NAs!** ⚠

- When you have **values** of X that are **not common** to both datasets
 - Any other join than the inner_join() will **generate NAs**

```
attainment %>% full_join(read.csv("spending.csv"), by = "country")
```

```
##   country year.x share_tertiary year.y share_gdp
## 1     FRA    2015      44.68760  2015      3.398
## 2     GBR    2015      49.94341     NA        NA
## 3     USA    2015      46.51771  2015      3.207
## 4     ITA    2015      25.14996     NA        NA
## 5     ESP    2015      40.95978     NA        NA
## 6     RUS     NA           NA     NA      1.843
```

- Any variable from A (B) other than those stated in by= will be NA for observations that are only in B (A)
- This holds when a variable that is not mentioned in the by= argument appears in both datasets:
 - In that case, R adds a data-specific suffix to the names and keeps them both
 - The variable from B (here year.y) will be NA for observations that are only in A only (here GBR, ITA, ESP)



2. Merge and reshape

2.1. Merge and append data: example

```
attainment %>% left_join(read.csv("spending.csv"), by = "country")
```

```
##   country year.x share_tertiary year.y share_gdp
## 1     FRA    2015      44.68760    2015      3.398
## 2     GBR    2015      49.94341      NA        NA
## 3     USA    2015      46.51771    2015      3.207
## 4     ITA    2015      25.14996      NA        NA
## 5     ESP    2015      40.95978      NA        NA
```

```
attainment %>% right_join(read.csv("spending.csv"), by = "country")
```

```
##   country year.x share_tertiary year.y share_gdp
## 1     FRA    2015      44.68760    2015      3.398
## 2     USA    2015      46.51771    2015      3.207
## 3     RUS     NA          NA       NA      1.843
```

→ *What would be the result of an inner_join() here?*



2. Merge and reshape

2.2. Reshape data

- It is important to be able to switch from the **long** to the **wide** format and conversely
 - Some computations should be done in one format or the other

Wide format

country	year	share_tertiary	share_gdp
FRA	2015	44.69	3.40
USA	2015	46.52	3.21

Long format

country	year	Variable	Value
FRA	2015	share_tertiary	44.69
FRA	2015	share_gdp	3.40
USA	2015	share_tertiary	46.52
USA	2015	share_gdp	3.21



2. Merge and reshape

2.2. Reshape data: From wide to long with `pivot_longer()`

```
wide <- attainment %>%
  inner_join(read.csv("spending.csv") %>% select(-year),
             by = "country")
wide
```

```
##   country year share_tertiary share_gdp
## 1      FRA 2015     44.68760    3.398
## 2      USA 2015     46.51771    3.207
```

→ Pivoting to **long format** can be seen as putting **variables on top of each other** rather side to side

- We need to indicate:
 - **Which variables to stack**
 - The **name of** the variable in which we want the **values** of the stacked variables to be stored
 - The **name of** the variable that will indicate to which **variable** corresponds each value



2. Merge and reshape

2.2. Reshape data: From wide to long with pivot_longer()

```
long <- wide %>%
  pivot_longer(c(share_tertiary, share_gdp),
  values_to = "Value",
  names_to = "Variable")
```

long

```
## # A tibble: 4 x 4
##   country year Variable     Value
##   <chr>    <int> <chr>      <dbl>
## 1 FRA      2015 share_tertiary 44.7
## 2 FRA      2015 share_gdp     3.40
## 3 USA      2015 share_tertiary 46.5
## 4 USA      2015 share_gdp     3.21
```



2. Merge and reshape

2.2. Reshape data: From long to wide with pivot_wider()

- To **pivot in a wide** format we need to indicate:
 - **Which variable** contains **values** of the variables we want to put side to side
 - **Which variable** indicates which **variable** correspond to each value

```
wide <- long %>%
  pivot_wider(values_from = "Value",
  names_from = "Variable")
```

wide

```
## # A tibble: 2 x 4
##   country  year share_tertiary share_gdp
##   <chr>    <int>      <dbl>      <dbl>
## 1 FRA      2015       44.7       3.40
## 2 USA      2015       46.5       3.21
```

Practice

10 : 00

- 1) From the `fb` dataset, create a variable `league` equal to "ligue1" and a variable `season` equal to "2021-2022" and save this new data in an object named `full_fb`
- 2) In `data.zip` you will find the rest of the data for the seasons 2019-2020 to 2021-2022 for the league 1, the bundesliga and the premier league. Append all these data to `full_fb`. Make sure to create the variables `league` and `season` in each data set before appending.
- 3) Use the `separate` function from `tidyverse` to extract the number of goals scored by the home and away team
- 4) Convert these variables as numeric and create a variable equal to the sum of the goals from the two teams
- 5) Summarise your data into the total number of goals score per league/season
- 6) Reshape your data such that you have 1 row per league and 1 column per season

You've got 10 minutes!

Solution

1) From the `fb` dataset, create a variable `league` equal to "ligue1" and a variable `season` equal to "2021-2022" and save this new data in an object named `full_fb`

```
full_fb <- fb %>% mutate(league = "ligue1", season = "2021-2022")
```

2) In `data.zip` you will find the rest of the data for the seasons 2019-2020 to 2021-2022 for the league 1, the bundesliga and the premier league. Append all these data to `full_fb`. Make sure to create the variables `league` and `season` in each data set before appending.

```
full_fb <- full_fb %>%
  bind_rows(read.csv("ligue1_2021.csv")) %>% mutate(league = "ligue1", season = "2020-2021")) %>%
  bind_rows(read.csv("ligue1_1920.csv")) %>% mutate(league = "ligue1", season = "2019-2020")) %>%
  bind_rows(read.csv("preml_2122.csv")) %>% mutate(league = "preml", season = "2021-2022")) %>%
  bind_rows(read.csv("preml_2021.csv")) %>% mutate(league = "preml", season = "2020-2021")) %>%
  bind_rows(read.csv("preml_1920.csv")) %>% mutate(league = "preml", season = "2019-2020")) %>%
  bind_rows(read.csv("bundes_2122.csv")) %>% mutate(league = "bundes", season = "2021-2022")) %>%
  bind_rows(read.csv("bundes_2021.csv")) %>% mutate(league = "bundes", season = "2020-2021")) %>%
  bind_rows(read.csv("bundes_1920.csv")) %>% mutate(league = "bundes", season = "2019-2020"))
```

Solution

3) Use the `separate` function from `tidyverse` to extract the number of goals scored by the home and away team

```
full_fb <- full_fb %>%  
  separate(Score, c("home_score", "away_score"), "-")
```

4) Convert these variables as numeric and create a variable equal to the sum of the goals from the two teams

```
full_fb <- full_fb %>%  
  mutate(home_score = as.numeric(home_score),  
        away_score = as.numeric(away_score),  
        goals = home_score + away_score)
```

5) Summarise your data into the total number of goals score per league/season

```
full_fb <- full_fb %>%  
  group_by(league, season) %>%  
  summarise(goals = sum(goals))
```

Solution

6) Reshape your data such that you have 1 row per league and 1 column per season

```
full_fb %>%
  pivot_wider(names_from = "season", values_from = "goals")
```

```
## # A tibble: 3 x 4
## # Groups:   league [3]
##   league `2019-2020` `2020-2021` `2021-2022`
##   <chr>     <dbl>     <dbl>     <dbl>
## 1 bundes     982      928      954
## 2 ligue1     704     1049     1067
## 3 preml    1034     1024     1071
```



Overview

1. The dplyr package ✓

- 1.1. Packages
- 1.2. Basic functions
- 1.3. group_by() and summarise()

2. Merge and reshape ✓

- 2.1. Merge and append data
- 2.2. Reshape data

3. A few words on learning R

- 3.1. When it doesn't work the way you want
- 3.2. Where to find help
- 3.3. When it doesn't work at all

4. Wrap up!



Overview

1. The dplyr package ✓

- 1.1. Packages
- 1.2. Basic functions
- 1.3. group_by() and summarise()

3. A few words on learning R

- 3.1. When it doesn't work the way you want
- 3.2. Where to find help
- 3.3. When it doesn't work at all

2. Merge and reshape ✓

- 2.1. Merge and append data
- 2.2. Reshape data



3. A few words on learning R

3.1. When it doesn't work the way you want

- When things do not work the way you want, **NAs are the usual suspects**
 - For instance, this is how the mean function reacts to NAs:

```
mean(c(1, 2, NA))
```

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

```
mean(c(1, 2, NA), na.rm = T)
```

```
## [1] 1.5
```

- You should systematically **check for NAs!**

```
is.na(c(1, 2, NA))
```

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



3. A few words on learning R

3.1. When it doesn't work the way you want

- **Don't pipe blindfolded!**
 - **Check** that each command does what it's expected to do
 - View or print your data **at each step**

```
fb %>%
  select(Home, Score, Away) %>%
  head(1)
```

```
##      Home Score   Away
## 1 Monaco    1-1 Nantes
```

```
fb %>%
  select(Home, Score, Away) %>%
  separate(Score, c("home_score", "away_score"), "-") %>%
  head(1)
```

```
##      Home home_score away_score   Away
## 1 Monaco           1           1 Nantes
```



3. A few words on learning R

3.2. Where to find help

- Oftentimes things don't work either because:
 - You don't understand a function's argument
 - Or you don't know that there exists an argument that you should use
- This is precisely what **help files** are made for
 - Every function has a help file, just enter `?` and the name of your **function** in the console
 - The help file will **pop up in the Help tab** of R studio

```
?paste
```

Arguments

... one or more `R` objects, to be converted to character vectors.

`sep` a character string to separate the terms. Not `NA_character_`.

`collapse` an optional character string to separate the results. Not `NA_character_`.

`recycle0` `logical` indicating if zero-length character arguments should lead to the zero-length `character(0)` after the `sep`-phase (which turns into "" in the `collapse`-phase, i.e., when `collapse` is not `NULL`).



3. A few words on learning R

3.2. Where to find help

- Search on the internet!
 - Your question is for sure already asked and answered on stackoverflow

Google

rename column R

All Videos Images News Shopping More Tools

About 12,200,000 results (0.44 seconds)

<https://www.datanovia.com> › Home › Lessons

Rename Data Frame Columns in R - Datanovia

This can be done easily using the function `rename()` [dplyr package]. It's also possible to use R base functions, but they require more typing. [Renaming Columns ...](#)

<https://stackoverflow.com> › questions › how-to-rename...

How to rename a single column in a data.frame? - Stack ...

May 10, 2013 — 20 Answers · 5. I'm also quite new with R, loved this solution! · 3. For regular expression results, use something like `names(df) = sub('pattern', ...)`

20 answers · Top answer: `colnames(trSamp)[2] <- "newname2"` attempts to set the second col...

Changing column names of a data frame - Stack ... 16 answers Jul 20, 2017

How to Rename Column Headers in R - Stack Overflow 3 answers Jun 4, 2018



3. A few words on learning R

3.3. When it doesn't work at all

- Sometimes R breaks and returns an **error** (usually kind of cryptic)

```
read.csv("C:\Users\Documents\R")
```

```
## Error: '\U' used without hex digits in character string starting ""C:\U"
```

1. Look for **keywords** that might help you understand where it comes from
2. Paste in on **Google** with the name of your command

The screenshot shows a Google search results page. The search query in the bar is "'\U' used without hex digits in character string starting ""C:\U""'. The results section displays the following information:

- Google logo
- Search bar with the query
- Navigation tabs: All, Shopping, Videos, News, Images, More, Tools
- Text: About 711,000 results (0.65 seconds)
- Link: <https://stackoverflow.com/questions/starting-r-error-...>
- Text: starting R: Error: '\U' used without hex digits in character string ...
- Text: May 30, 2017 · 8 answers



Overview

1. The dplyr package ✓

- 1.1. Packages
- 1.2. Basic functions
- 1.3. group_by() and summarise()

3. A few words on learning R ✓

- 3.1. When it doesn't work the way you want
- 3.2. Where to find help
- 3.3. When it doesn't work at all

2. Merge and reshape ✓

- 2.1. Merge and append data
- 2.2. Reshape data

4. Wrap up!

4. Wrap up!

1. Packages

```
library(dplyr)
```

2. Main dplyr functions

Function	Meaning
mutate()	Modify or create a variable
select()	Keep a subset of variables
filter()	Keep a subset of observations
arrange()	Sort the data
group_by()	Group the data
summarise()	Summarizes variables into 1 observation per group



4. Wrap up!

3. Merge data

```
a <- data.frame(x = c(1, 2, 3), y = c("a", "b", "c"))
b <- data.frame(x = c(4, 5, 6), y = c("d", "e", "f"))
c <- data.frame(x = 1:6, z = c("alpha", "bravo", "charlie", "delta", "echo", "foxtrot"))
```

```
a %>% bind_rows(b) %>% left_join(c, by = "x")
```

x	y	z
1	a	alpha
2	b	bravo
3	c	charlie
4	d	delta
5	e	echo
6	f	foxtrot



4. Wrap up!

4. Reshape data

country	year	share_tertiary	share_gdp
FRA	2015	44.69	3.40
USA	2015	46.52	3.21

```
data %>% pivot_longer(c(share_tertiary, share_gdp), names_to = "Variable", values_to = "Value")
```

country	year	Variable	Value
FRA	2015	share_tertiary	44.69
FRA	2015	share_gdp	3.40
USA	2015	share_tertiary	46.52
USA	2015	share_gdp	3.21



For next time

Install the R packages needed for Part I of the course:

`ggplot2`

`rmarkdown`

`knitr`

`DT`