Fundamentals of R

Cleaning and Wrangling Data

Henrique Sposito and Livio Silva-Muller

GENEVA GRADUATE INSTITUTE

INSTITUT DE HAUTES ÉTUDES INTERNATIONALES ET DU DÉVELOPPEMENT

GRADUATE INSTITUTE OF INTERNATIONAL AND DEVELOPMENT STUDIES

What did we learn last week?

Conceptually:

- What are objects?
- What are classes?
- What are data structures?

Practically:

```
# ← the assign operator
# $  the extract operator
# == , ≠, ≤, ≥  logical operators
# &, /, | conditions
# [1,3] brackets
class()
dim()
length()
summary()
data.frame()
median()
mean()
subset()
ifelse()
grepl()
```

Homework for today

Let's start by going through the homework together!

- Common issues
- Any other general questions?

Lecture:

- Tidy verse and tidy thinking
- From untidy to tidy data
- From tidy data to findings

Practical:

- Cleaning data
- Wrangling data
- Joining data

Tidyverse? (1/2)

The universe of tidiness?

From the tidyverse website:

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

Tidyverse? (2/2)

Installing and loading the tidyverse: *be careful!*

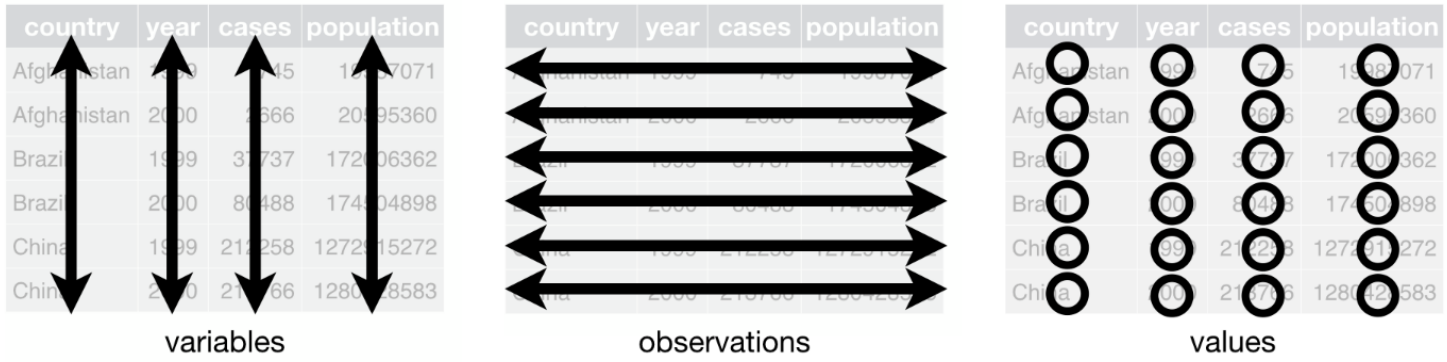
```
install.packages("tidyverse")
library(tidyverse)
```

The tidyverse is actually *eight* different packages!

`

Tidy data (1/4)

- What is a tidy dataset?
  - Each column is one variable
  - Each row is one case/observation
  - Each cell stores one value



Check: Hadley Wickham's dissertation paper that introduced tidy data.

Tidy data (2/4)

Is this tidy?

Animal	Conversion
Domestic dog	24.0 / 5.10
Domestic cat	30.0 / 4.08
American alligator	77.0 / 1.59
Golden hamster	3.9 / 31.41
King penguin	26.0 / 4.71

No! Cells in the second column contain two values.

Author: Fabio Votta

Tidy data (3/4)

Is this tidy?

Animal	Type	Value
Domestic dog	lifespan	24.0
Domestic dog	ratio	5.10
Domestic cat	lifespan	30.0
Domestic cat	ratio	4.08
American alligator	lifespan	77.0
American alligator	ratio	1.59

No! The second column contains two variables.

Author: Fabio Votta

Tidy data (4/4)

One last example:

from untidy...

country	1999	2000
Afghanistan	19987071	20595360
Brazil	172006362	174504898

to tidy!

country	year	population
Afghanistan	1999	19987071
Afghanistan	2000	20595360
Brazil	1999	172006362
Brazil	2000	174504898

Tidying (1/6)

tidyr can help getting from untidy to tidy data in R!

- `separate()` separates data by tidying columns with two variables or values (`unite()` does the opposite.)
- `pivot_wider()` pivots data by tidying columns with more than one variable.
- `pivot_longer()` pivots data by tidying rows that contain variables.

Let's see how this works in the following slides...

Tidying (2/6)

Is this tidy or untidy data?

```
untidy_population ← data.frame("Country" = c("Afghanistan-middle east",
                                             "Brazil-south america",
                                             "China-eastern asia"),
                               "year_1999" = c(19987071, 172006362, 1272915272),
                               "year_2000" = c(20595360, 174504898, 1280428583))
untidy_population
```

```
#>                   Country  year_1999  year_2000
#> 1 Afghanistan-middle east   19987071   20595360
#> 2    Brazil-south america  172006362  174504898
#> 3      China-eastern asia 1272915272 1280428583
```

How to make this data tidy?

Tidying (3/6)

Let's `separate()` columns

```
library(tidyr)
tidy_population ← separate(untidy_population, Country,
                           sep="-", into=c("country", "region"))
tidy_population
```

```
#>       country        region  year_1999  year_2000
#> 1 Afghanistan   middle east   19987071   20595360
#> 2      Brazil south america  172006362  174504898
#> 3       China  eastern asia 1272915272 1280428583
```

What else could we do to make this even tidier?

Tidying (4/6)

Let's pivot the data long

- `pivot_longer()` pivots data by tidying rows that contain variables (i.e. means more observations, a *longer* dataset)

```
pivot_longer(tidy_population,
             cols = c("year_1999","year_2000"),
             names_to = "year",
             values_to = "population")
```

```
#> # A tibble: 6 × 4
#>   country     region        year      population
#>   <chr>       <chr>         <chr>          <dbl>
#> 1 Afghanistan middle east   year_1999   19987071
#> 2 Afghanistan middle east   year_2000   20595360
#> 3 Brazil      south america year_1999  172006362
#> 4 Brazil      south america year_2000  174504898
#> 5 China       eastern asia  year_1999 1272915272
#> 6 China       eastern asia  year_2000 1280428583
```

`

Tidying (5/6)

One last example

```
long_burgers ← data.frame(Lecturer = c("Henrique", "Livio",
                                       "Henrique", "Livio",
                                       "Henrique", "Livio"),
                          Burger_joint = c("Inglewoods", "Inglewoods",
                                           "H. Foundation", "H. Foundation",
                                           "Holy Cow", "Holy Cow"),
                          Grade = c(7, 10, 5, 5, 9, 7))
long_burgers
```

```
#>   Lecturer  Burger_joint Grade
#> 1 Henrique    Inglewoods     7
#> 2    Livio    Inglewoods    10
#> 3 Henrique H. Foundation     5
#> 4    Livio H. Foundation     5
#> 5 Henrique      Holy Cow     9
#> 6    Livio      Holy Cow     7
```

How could we get each burger joint in a column with cells containing grades?

`

Tidying (6/6)

Let's pivot this data wide

- `pivot_wider()` pivots data by tidying columns with more than one variable (i.e. this means more variables, a *wider* dataset)

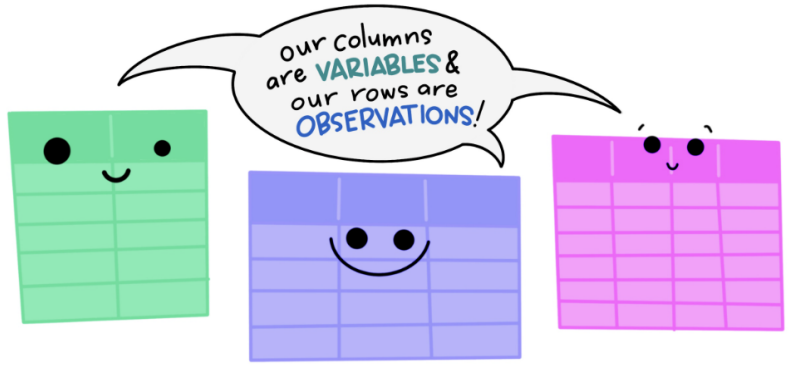
```
pivot_wider(long_burgers,
            names_from = Burger_joint,
            values_from = Grade)
```

```
#> # A tibble: 2 × 4
#>   Lecturer Inglewoods `H. Foundation` `Holy Cow`
#>   <chr>         <dbl>           <dbl>      <dbl>
#> 1 Henrique          7               5          9
#> 2 Livio            10               5          7
```

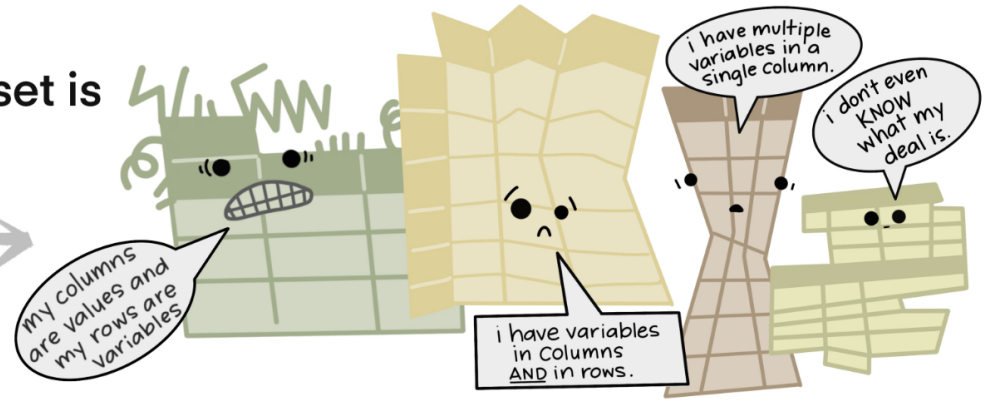
`

… but, a note of caution:

The standard structure of tidy data means that "tidy datasets are all alike…"



"…but every messy dataset is messy in its own way."
–HADLEY WICKHAM



Artist: Allison Horst.

… but, another note of caution:

- Tidy data is great for working with tidy packages but …
- Tidying data takes time (i.e. do you really need to separate first and last name in two columns?)

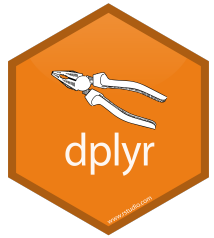
Tidyvice:

Think about it conceptually before tidying everything!

Wrangling (1/8)

Now that things are tidier, it's time to wrangle data!

By wrangling, we mean manipulating data, and for this we usually use the dplyr package!



> The dplyr package is a **grammar of data manipulation**, providing a consistent set of verbs that help you solve the most common data manipulation challenges: creating,adding, picking, reducing, and changing variables.

Wrangling (2/8)

Let's load some data on the income of international organizations

```
library(dplyr)
slice_head(io_income, n = 10)
```

```
#> # A tibble: 10 × 4
#>     year donor                                    type_donor amount_nominal
#>    <dbl> <chr>                                    <chr>               <dbl>
#>  1  2000 Maldives                                 public              1057.
#>  2  2000 Estonia                                  public             18192.
#>  3  2000 Burundi                                  public              2082.
#>  4  2000 Mozambique                               public              1057.
#>  5  2000 Morocco                                  public             43350.
#>  6  2000 Mozambique                               public              3580
#>  7  2000 Sasakawa Endowment Fund Interest (JPN)   private           105000
#>  8  2000 China                                    public           1052019.
#>  9  2000 New Zealand                              public             70848
#> 10  2000 Mc Henry Tichenor (USA)                  private            25000
```

`

Wrangling (3/8)

Let's `select()` some variables (by column)

```
select(io_income, donor, type_donor)
```

```
#> # A tibble: 10 × 2
#>    donor                                  type_donor
#>    <chr>                                  <chr>
#>  1 Maldives                               public
#>  2 Estonia                                public
#>  3 Burundi                                public
#>  4 Mozambique                             public
#>  5 Morocco                                public
#>  6 Mozambique                             public
#>  7 Sasakawa Endowment Fund Interest (JPN) private
#>  8 China                                  public
#>  9 New Zealand                            public
#> 10 Mc Henry Tichenor (USA)                private
```

`

Wrangling (4/8)

`filter()` subsets data by a condition in the observations (by rows)

```
filter(io_income, donor == "Switzerland")
```

```
#> # A tibble: 10 × 4
#>     year donor       type_donor amount_nominal
#>    <dbl> <chr>       <chr>                <dbl>
#>  1  2000 Switzerland public             320345.
#>  2  2000 Switzerland public           58976921.
#>  3  2001 Switzerland public            1276541.
#>  4  2001 Switzerland public              11933
#>  5  2001 Switzerland public             231191
#>  6  2002 Switzerland public            1400389.
#>  7  2002 Switzerland public                 64.2
#>  8  2002 Switzerland public             599408
#>  9  2002 Switzerland public                128.
#> 10  2002 Switzerland public           57295239.
```

`

Wrangling (5/8)

`mutate()` creates new variables, often with new values, from other variables

```
io_income_mutate ← mutate(io_income,
                          thousands_USD = amount_nominal/1000)
```

```
#> # A tibble: 10 × 5
#>     year donor                                  type_donor amount_nomi…¹ thous…²
#>    <dbl> <chr>                                  <chr>              <dbl>   <dbl>
#>  1  2000 Maldives                               public             1057.    1.06
#>  2  2000 Estonia                                public            18192.   18.2
#>  3  2000 Burundi                                public             2082.    2.08
#>  4  2000 Mozambique                             public             1057.    1.06
#>  5  2000 Morocco                                public            43350.   43.3
#>  6  2000 Mozambique                             public             3580     3.58
#>  7  2000 Sasakawa Endowment Fund Interest (JPN) private           105000   105
#>  8  2000 China                                  public           1052019. 1052.
#>  9  2000 New Zealand                            public             70848    70.8
#> 10  2000 Mc Henry Tichenor (USA)                private            25000    25
#> # … with abbreviated variable names ¹amount_nominal, ²thousands_USD
```

Wrangling (6/8)

Grouping and summarizing are powerful tidy tools!

- `group_by()` groups data based on a characteristic to subsequently perform certain operations
- `summarise()` creates a new dataframe by combining grouped variables.

```
io_income_grouped ← group_by(io_income, type_donor)
summarise(io_income_grouped, amount_nominal = sum(amount_nominal,
                                                  na.rm = TRUE))
```

```
#> # A tibble: 3 × 2
#>   type_donor amount_nominal
#>   <chr>                <dbl>
#> 1 private       4503067539.
#> 2 public       35524020021.
#> 3 <NA>           341116650.
```

Wrangling (7/8)

dplyr has a family of functions that help you join multiple datasets: `left_join()`, `right_join()`, `inner_join()`, `full_join()` ...

- This great post on all merging possibilities and details with dplyr.

Let's get a new dataset which contains regions for each country to see these joins work!

```
sample_n(country_region, 5)
```

```
#> # A tibble: 5 × 2
#>   donor        region
#>   <chr>        <chr>
#> 1 Mexico       Latin America
#> 2 Belgium      Europe
#> 3 South Sudan  Africa
#> 4 New Zealand  Oceania
#> 5 Turkmenistan Asia
```

Wrangling (8/8)

If I wanted to add a region column in the income of international organizations data, which join should I use?

```
left_join(io_income, country_region)
```

```
#> # A tibble: 4,386 × 5
#>     year donor                                    type_donor amount_nominal region
#>    <dbl> <chr>                                    <chr>               <dbl> <chr>
#>  1  2000 Maldives                                 public              1057. Asia
#>  2  2000 Estonia                                  public             18192. Europe
#>  3  2000 Burundi                                  public              2082. Africa
#>  4  2000 Mozambique                               public              1057. Africa
#>  5  2000 Morocco                                  public             43350. Africa
#>  6  2000 Mozambique                               public              3580  Africa
#>  7  2000 Sasakawa Endowment Fund Interest (JPN)   private           105000  <NA>
#>  8  2000 China                                    public            1052019. Asia
#>  9  2000 New Zealand                              public              70848  Ocean…
#> 10  2000 Mc Henry Tichenor (USA)                  private             25000  <NA>
#> # … with 4,376 more rows
```

There are many others dplyr verbs that we did cover here....

```
arrange() # orders row by values
rename() # renames variables
distinct() # keep only distinct rows
slice() # selects rows by position
count() # counts number of rows with unique values
```

And much more...

`

The Pipe Operator %>% (1/3)

- So far, we have been doing *object-oriented programming*
- This means we assign (<-) the result of an operation to an object
- And, then, operate on the newly created object


```
io_income_filtered← filter(io_income, year > "2018")
io_income_grouped ← group_by(io_income_filtered, year)
io_income_summarised ← summarise(io_income_grouped,
                                 amount_nominal = sum(amount_nominal,
                                                      na.rm = TRUE))
```

The Pipe Operator %>% (2/3)

- The pipe operator (%>%) changes the object-oriented logic, it takes the output of one function and passes it into another function as an argument!

- This is why we call it *functional-programming*!


```
io_income %>%
filter(year > "2018") %>%
group_by(year) %>%
summarise(amount_nominal = sum(amount_nominal, na.rm = TRUE))
```

Check: Plumbers, chains, and famous painters: The (updated) history of the pipe operator in R.

The Pipe Operator %>% (3/3)

```
io_income_filtered← filter(io_income, year >
"2018")
io_income_grouped ←
group_by(io_income_filtered, year)
io_income_summarised ←
summarise(io_income_grouped,

amount_nominal = sum(amount_nominal,

na.rm = TRUE))
io_income_summarised
```

```
#> # A tibble: 2 × 2
#>    year amount_nominal
#>   <dbl>          <dbl>
#> 1  2019    4332215887.
#> 2  2020    2921693184.
```

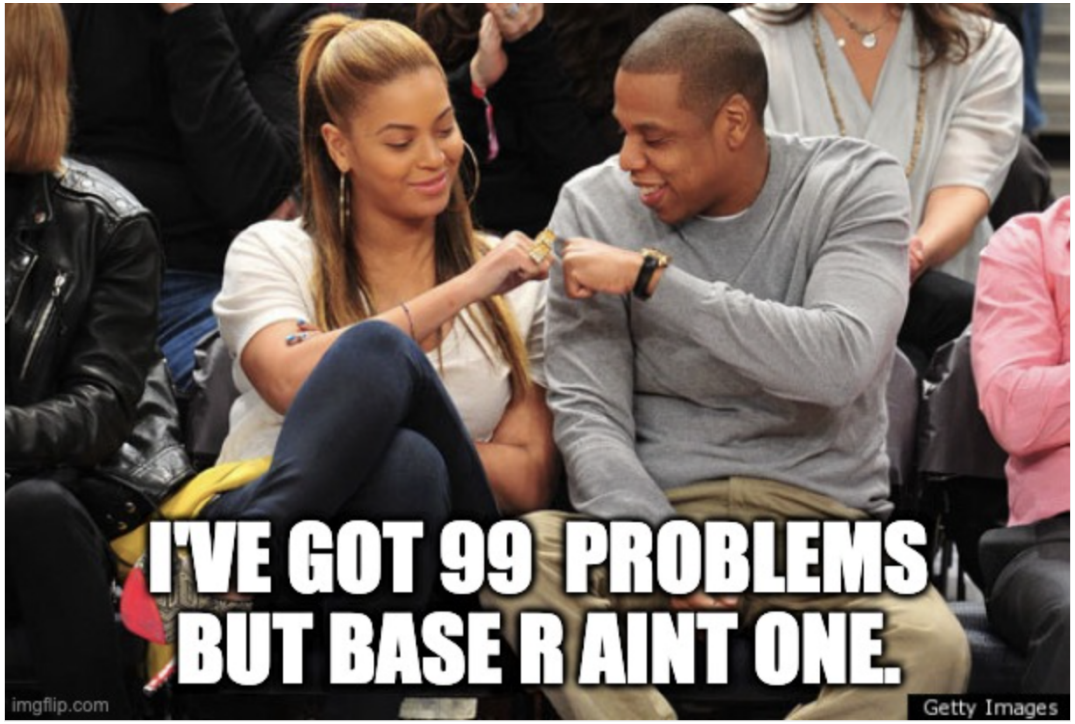
`

```
io_income %>%
filter(year > "2018") %>%
group_by(year) %>%
summarise(amount_nominal = sum(amount_nominal,
na.rm = TRUE))
```

```
#> # A tibble: 2 × 2
#>    year amount_nominal
#>   <dbl>          <dbl>
#> 1  2019    4332215887.
#> 2  2020    2921693184.
```

Base vs. Tidy (re-visited)

- Tidy data and piping are both *intuitive* and *powerful*
- All packages in the tidyverse are optimized for tidy data and piping
- Some packages outside the tidyverse were also developed within the tidy paradigm
- Nevertheless, many *important* R packages and other programming software operate outside the logic of tidy
- While you may like tidy, a better programmer knows both!




QUESTIONS?

Repository I: replace_na()

- `tidyr::replace_na()` replaces NAs with specified values.

```
io_income %>%
  mutate(donor=replace_na(donor,"individuals"))
#all NAs in donor are now "individuals"

io_income$donor %>%
  replace_na("individual")
#all NAs in donor are now "individuals"

io_income %>%
  replace_na(list(donor="individuals",amount_nominal=0))
#all NAs in donor are now individuals, and all NAs in amount_nominal are now 0

io_income%>%
  replace_na(list(0))
#all NAs in the dataset are now 0
```

`

Repository II: case_when()

- `dplyr::case_when()` and `base::ifelse()` can help you return values based on tests.
- Different from `ifelse()`, `case_when()` can **more** easily return more than two values based on more than one test.
- Let's say we want a new variable called un_sg (UN Secretary General):
  - kofi_annan, if years are between 1997 and 2006;
  - ban_kimoon, if years are between 2007 and 2016;
  - antonio_guterres, if years are between 2016 and present.

Repository II: case_when()

```
io_income%>%
  mutate(un_sg=
    ifelse (year<2006, "kofi_annan",
      ifelse(year⩾2016,
        "antonio_guterres", "ban_kimoon")))%>%
  slice_sample(n=5)%>%
  select(year, donor,un_sg)
```

```
#> # A tibble: 5 × 3
#>    year donor      un_sg
#>   <dbl> <chr>      <chr>
#> 1  2008 Belgium    ban_kimoon
#> 2  2005 San Marino kofi_annan
#> 3  2014 Turkey     ban_kimoon
#> 4  2001 Syria      kofi_annan
#> 5  2017 Hungary    antonio_guterres
```

`

```
io_income%>%
  mutate(un_sg= case_when(
      year < 2006 ~ "kofi_annan",
      year < 2016 ~ "ban_kimoon",
      year ⩾ 2016 ~"antonio_guterres"))%>%
#case_when respects order!
  slice_sample(n=5)%>%
  select(year, donor,un_sg)
```

```
#> # A tibble: 5 × 3
#>    year donor   un_sg
#>   <dbl> <chr>   <chr>
#> 1  2001 Sudan   kofi_annan
#> 2  2014 Poland  ban_kimoon
#> 3  2016 Iran    antonio_guterres
#> 4  2007 Austria ban_kimoon
#> 5  2011 Somalia ban_kimoon
```