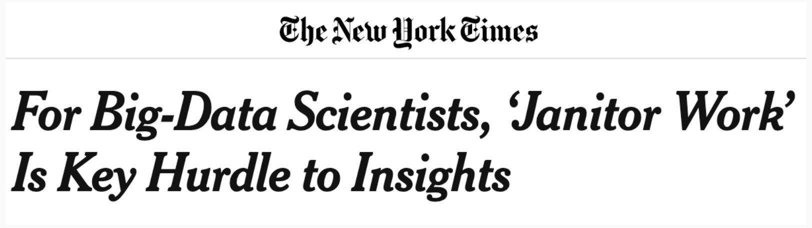
Cleaning dirty data with {janitor}

Workshop: I2DS Tools for Data Science

Abigail Pena Alejos & Nikolina Klatt
Hertie School
November 21st 2022

Agenda


The New York Times

For Big-Data Scientists, 'Janitor Work' Is Key Hurdle to Insights

The New York Times

Overview

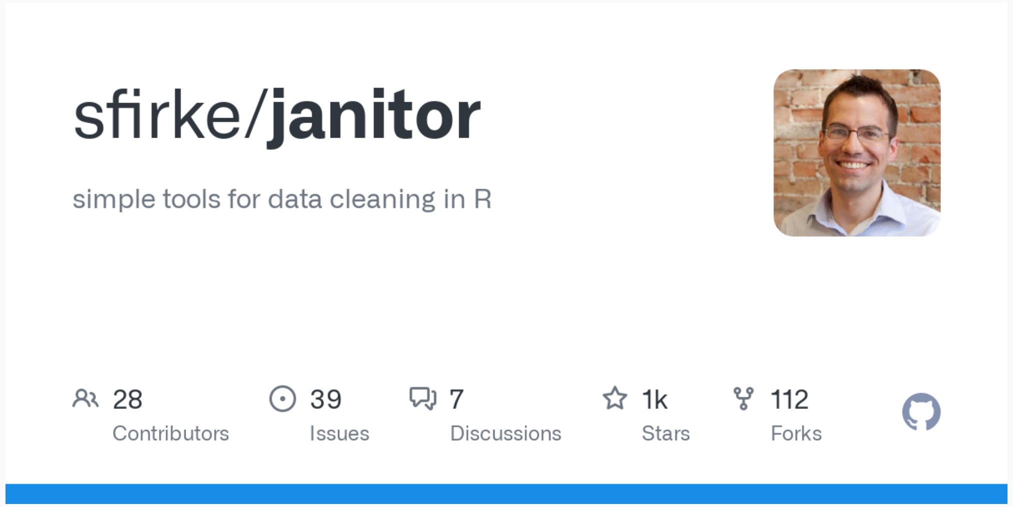
{**janitor**} is a package built by **Sam Firke**.

- Simple functions for **cleaning** and **examining** data
- Optimized for **user-friendliness**
- For beginning and intermediate R users
- Advanced R users: with {janitor} they can do data wrangling faster

Let's get started

--> `install.packages("janitor")`

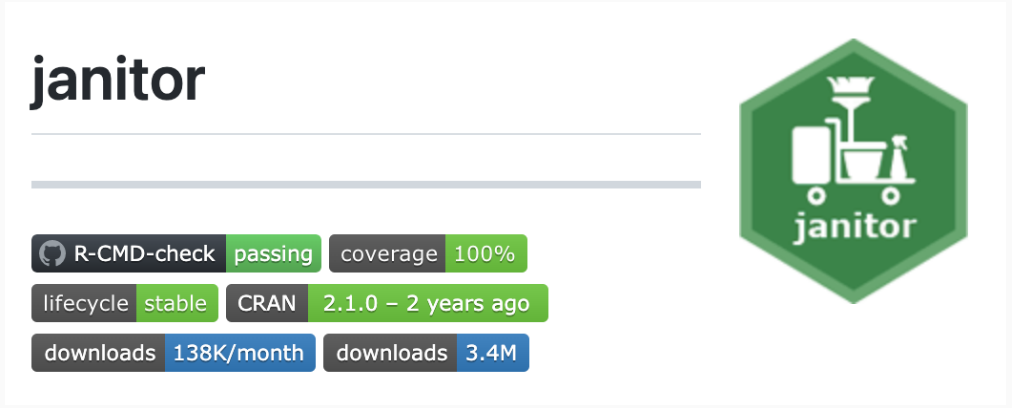
--> `library(janitor)`



sfirke/**janitor**

simple tools for data cleaning in R

28 Contributors · 39 Issues · 7 Discussions · 1k Stars · 112 Forks



janitor

R-CMD-check passing · coverage 100% · lifecycle stable · CRAN 2.1.0 – 2 years ago · downloads 138K/month · downloads 3.4M

GitHub janitor

Cleaning data

Clean data frame names with `clean_names()`

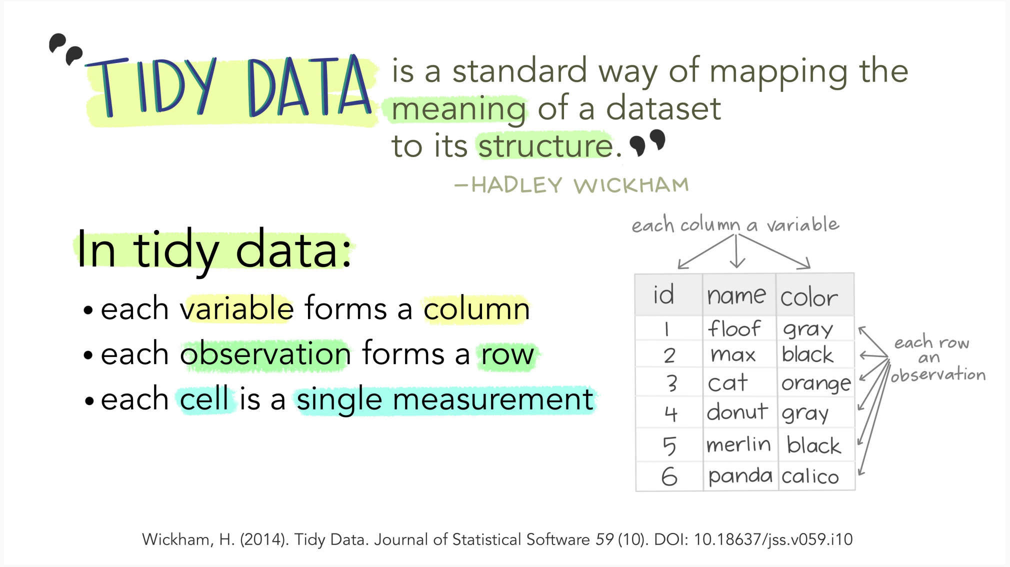
tidyverse grammar

Consistent variable names are important

snake_case

“Variable and function names should use only lowercase letters, numbers, and underscores. Use underscores (so called snake_case) to separate words within a name.”

The tidyverse style guide


“TIDY DATA is a standard way of mapping the meaning of a dataset to its structure.”

—HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement

id	name	color
1	floof	gray
2	max	black
3	cat	orange
4	donut	gray
5	merlin	black
6	panda	calico

Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

Julie Lowndes and Allison Horst

Cleaning data

Clean data frame names with `clean_names()`

```
> # Create a test data frame with dirty variable names
> dirty_df ← as.data.frame(matrix(ncol = 7))
> names(dirty_df) ← c("firstName", "a@b!c'ß??",
+                    "success.rate.in.%.(2022)",
+                    "IDENTICAL", "IDENTICAL",
+                    "", "#")
> clean_df ← dirty_df %>%
+   clean_names()
>
> colnames(clean_df)
```

```
## [1] "first_name"                  "a_b_css"
## [3] "success_rate_in_percent_2022" "identical"
## [5] "identical_2"                 "x"
## [7] "number"
```

Cleaning data

Clean data frame names with `clean_names()`

```
> # Create a test data frame with dirty variable names
> dirty_df ← as.data.frame(matrix(ncol = 7))
> names(dirty_df) ← c("firstName", "a@b!c'ß⁇",
+                   "success.rate.in.%.(2022)",
+                   "IDENTICAL", "IDENTICAL",
+                   "", "#")
> clean_df ← dirty_df %>%
+   clean_names()
>
> colnames(clean_df)
```

```
## [1] "first_name"                      "a_b_css"
## [3] "success_rate_in_percent_2022" "identical"
## [5] "identical_2"                     "x"
## [7] "number"
```

- **Consistent format** for letter cases and separators
    - **snake_case** is default, but other cases like camelCase are available
- Handles **special characters** and **spaces**, including transliterating characters like `ß` to `ss`
- Converts "%" to "**percent**" and "#" to "**number**"
- Adds numbers to **duplicated names**
- **Spacing** (or lack thereof) around numbers is preserved

- Works with the **%>% operator**
- Can be used for data frames and objects
- `make_names_clean` also works for character vectors

Cleaning data

```
> # install.packages("palmerpenguins")
> library(palmerpenguins)
> dirty_penguins_df ← penguins_raw
> colnames(dirty_penguins_df)
```

```
##  [1] "studyName"           "Sample Number"       "Species"
##  [4] "Region"              "Island"              "Stage"
##  [7] "Individual ID"       "Clutch Completion"   "Date Egg"
## [10] "Culmen Length (mm)"  "Culmen Depth (mm)"   "Flipper Length (mm)"
## [13] "Body Mass (g)"       "Sex"                 "Delta 15 N (o/oo)"
## [16] "Delta 13 C (o/oo)"   "Comments"
```

Cleaning data

```
> # install.packages("palmerpenguins")
> library(palmerpenguins)
> dirty_penguins_df ← penguins_raw
> colnames(dirty_penguins_df)
```

```
##  [1] "studyName"           "Sample Number"       "Species"
##  [4] "Region"              "Island"              "Stage"
##  [7] "Individual ID"       "Clutch Completion"   "Date Egg"
## [10] "Culmen Length (mm)"  "Culmen Depth (mm)"   "Flipper Length (mm)"
## [13] "Body Mass (g)"       "Sex"                 "Delta 15 N (o/oo)"
## [16] "Delta 13 C (o/oo)"   "Comments"
```

```
> clean_penguins_df ← penguins_raw %>%
+   clean_names()
> colnames(clean_penguins_df)
```

```
##  [1] "study_name"        "sample_number"     "species"
##  [4] "region"            "island"            "stage"
##  [7] "individual_id"     "clutch_completion" "date_egg"
## [10] "culmen_length_mm"  "culmen_depth_mm"   "flipper_length_mm"
## [13] "body_mass_g"       "sex"               "delta_15_n_o_oo"
## [16] "delta_13_c_o_oo"   "comments"
```

Cleaning data

Remove content with `remove_empty()`

- `remove_empty()` rows and columns
  - Cleans Excel files that contain empty rows and columns after being read into R
  - Adding `quiet = FALSE` let's you know how many rows or columns were removed.

```
> empty_df <- data.frame(v1 = c(1, NA, 3),
+                       v2 = c(NA, NA, NA),
+                       v3 = c("a", NA, "b"))
>
> empty_df %>%
+   remove_empty(c("rows", "cols"), quiet = FALSE) %>%
+   glimpse
```

```
## Removing 1 empty rows of 3 rows total (33.3%).

## Removing 1 empty columns of 3 columns total (Removed: v2).

## Rows: 2
## Columns: 2
## $ v1 <dbl> 1, 3
```

Cleaning data

Drop constant columns with `remove_constant()`

`remove_constant()` columns
Drops columns from a data frame that contain only a single constant value

```
> adelie_df ← clean_penguins_df %>%
+   filter(species == "Adelie Penguin (Pygoscelis adeli
>
> adelie_df %>%
+   names()
```

```
##  [1] "study_name"        "sample_number"     "species"
##  [4] "region"            "island"            "stage"
##  [7] "individual_id"     "clutch_completion" "date_egg"
## [10] "culmen_length_mm"  "culmen_depth_mm"   "flipper_length_mm"
## [13] "body_mass_g"       "sex"               "delta_15_n_o_oo"
## [16] "delta_13_c_o_oo"   "comments"
```

```
> adelie_clean_df ← adelie_df %>%
+   remove_constant()
>
> adelie_clean_df %>%
+   names()
```

```
##  [1] "study_name"        "sample_number"     "island"
##  [4] "individual_id"     "clutch_completion" "date_egg"
##  [7] "culmen_length_mm"  "culmen_depth_mm"   "flipper_len
## [10] "body_mass_g"       "sex"               "delta_15_n_
## [13] "delta_13_c_o_oo"   "comments"
```

Cleaning data

Check content of columns with `compare_df_cols()`

Imagine you have a set of data frames that you want to combine by binding the rows together but `rbind()` fails. You can check if the column classes match and see if they are **matching** by running `compare_df_cols()`.

- takes unquoted names of data frames, tibbles, or a list of data frames

--> Returns a summary of how they compare

- What column types are there?
- How do column types differ?
- Which are missing or present in the different inputs?

```
> chinstrap_df <- clean_penguins_df %>%
+   filter(species == "Chinstrap penguin (Pygoscelis ar
>
> # rbind(adelie, chinstrap)
>
> compare_df_cols(adelie_clean_df, chinstrap_df) %>%
+   tail()
```

```
##      column_name adelie_clean_df chinstrap_df
## 12        region            <NA>    character
## 13 sample_number         numeric      numeric
## 14           sex       character    character
## 15       species            <NA>    character
## 16         stage            <NA>    character
## 17    study_name       character    character
```

Exploring data

`tabyl()` – a tidy, fully-featured approach to counting things

Why not use `table()`?

- It doesn't work with the `%>%` operator
- It doesn't output data frames
- Its results are hard to format

One variable

```
> table(clean_penguins_df$sex)
```

```
##
## FEMALE   MALE
##    165    168
```

Exploring data

`tabyl()` – a tidy, fully-featured approach to counting things

Why not use `table()`?

- It doesn't work with the `%>%` operator
- It doesn't output data frames
- Its results are hard to format

Instead better use `tabyl()`

- Tidyverse-aligned - primarily built upon the `{dplyr}` and `{tidyr}` packages
- Compatible with the `{knitr}` package
- Useful for data exploration
- Generate frequencies along with the percent of total
- Counts combinations of one, two, or three variables

One variable

```
> table(clean_penguins_df$sex)
```

```
##
## FEMALE   MALE
##    165    168
```

```
> tabyl(clean_penguins_df$sex)
```

```
##  clean_penguins_df$sex   n    percent valid_percent
##                 FEMALE 165 0.47965116     0.4954955
##                   MALE 168 0.48837209     0.5045045
##                   <NA>  11 0.03197674            NA
```

```
> clean_penguins_no_na_df ← clean_penguins_df %>%
+   drop_na(sex)
```

Exploring data

`tabyl()` – a tidy, fully-featured approach to counting things

Two variables Two-way tabyl / "crosstab" or "contingency" table

```
> clean_penguins_no_na_df %>%
+   tabyl(island, sex)
```

```
##     island FEMALE MALE
##     Biscoe     80   83
##      Dream     61   62
##  Torgersen     24   23
```

Exploring data

`tabyl()` – a tidy, fully-featured approach to counting things

Two variables Two-way tabyl / "crosstab" or "contingency" table

```
> penguins_crosstab_table <- clean_penguins_no_na_df %>%
+   tabyl(island, sex) %>%
+   adorn_totals("col") %>%                     # total in each row
+   adorn_percentages("row") %>%                # percentage value per row
+   adorn_pct_formatting(digits = 2) %>%        # rounded percentage value
+   adorn_ns() %>%                              # adds the absolute numbers
+   adorn_title()                               # adds the variable name
>
> penguins_crosstab_table
```

```
##                     sex                            
##     island       FEMALE         MALE          Total
##     Biscoe 49.08% (80) 50.92% (83) 100.00% (163)
##      Dream 49.59% (61) 50.41% (62) 100.00% (123)
##  Torgersen 51.06% (24) 48.94% (23) 100.00%  (47)
```

Exploring data

`adorn_*()` options

- `tabyl()` can be formatted with a suite of `adorn_*` functions to add information and for pretty formatting:

- **`adorn_totals()`**: Add totals row, column, or both.
- **`adorn_percentages()`**: Calculate percentages along either axis or the entire tabyl
- **`adorn_pct_formatting()`**: Format percentage columns, controlling the number of digits to display and whether to append the `%` symbol
- **`adorn_rounding()`**: Round a data frame of numbers

- **`adorn_ns()`**: Add Ns to a tabyl - drawn from the tabyl's underlying counts or they can be supplied by the user
- **`adorn_title()`**: Add a title to a tabyl - pptions include putting the column title in a new row on top of the data frame or combining the row and column titles in the data frame's first name slot.

Exploring data

`tabyl()` **Three variables** Three-way tabyl

```
> clean_penguins_no_na_df %>% tabyl(island, species, sex) %>%
+   adorn_percentages("all") %>%
+   adorn_pct_formatting(digits = 1) %>%
+   adorn_title() %>%
+   kable()
```

	species		
island	Adelie Penguin (Pygoscelis adeliae)	Chinstrap penguin (Pygoscelis antarctica)	Gentoo penguin (Pygoscelis papua)
Biscoe	13.3%	0.0%	35.2%
Dream	16.4%	20.6%	0.0%
Torgersen	14.5%	0.0%	0.0%

	species		
island	Adelie Penguin (Pygoscelis adeliae)	Chinstrap penguin (Pygoscelis antarctica)	Gentoo penguin (Pygoscelis papua)
Biscoe	13.1%	0.0%	36.3%
Dream	16.7%	20.2%	0.0%
Torgersen	13.7%	0.0%	0.0%

Cleaning data

Fixing dates

- `excel_numeric_to_date()`
  Fix dates stored as serial numbers
  Converts serial date numbers from Excel to class `Date`

```
> excel_numeric_to_date(44886)
```

```
## [1] "2022-11-21"
```

- `convert_to_date()`
  Convert a mix of date and datetime formats to date

```
> convert_to_date(c("2020-02-29", "40000.1"))
```

```
## [1] "2020-02-29" "2009-07-06"
```

```
> convert_to_datetime(40000.1)
```

```
## [1] "2009-07-06 02:24:00 UTC"
```

Cleaning data

Rounding numbers

Careful: In base R `round()` uses **"banker's rounding"**,
i.e., halves are rounded to the nearest *even* number.

```
> numbers <- c(3.5, 2.5)
> round(numbers)
```

```
## [1] 4 2
```

Instead use: `round_half_up()`
directionally-consistent rounding behavior

```
> round_half_up(numbers)
```

```
## [1] 4 3
```

`round_to_fraction()`
round decimals to precise fractions of a given denominator

```
> round_to_fraction(0.175, denominator = 4)
```

```
## [1] 0.25
```

```
> round_to_fraction(0.2, denominator = 4)
```

```
## [1] 0.25
```

```
> round_to_fraction(0.25000000001, denominator = 4)
```

```
## [1] 0.25
```

Exploring data

Detect duplicated records with `get_dupes()`

Checking for duplicated records is important because they can interfere with your tabulations in the analysis. {`janitor`} makes this tedious task simple.

`get_dupes()`

- Detects duplicate records during data cleaning
- Returns the records (and inserts a count of duplicates) so you can examine the potentially problematic cases

```
> clean_penguins_df %>%
+   get_dupes(individual_id) %>%
+   head(6)
```

```
## # A tibble: 6 × 18
##   individual_id dupe_c…¹ study…² sampl…³ species region i
##   <chr>            <int> <chr>     <dbl> <chr>   <chr>  <
## 1 N11A1                2 PAL0708      21 Adelie… Anvers B
## 2 N11A1                2 PAL0809      47 Gentoo… Anvers B
## 3 N11A2                2 PAL0708      22 Adelie… Anvers B
## 4 N11A2                2 PAL0809      48 Gentoo… Anvers B
## 5 N12A1                2 PAL0708      23 Adelie… Anvers B
## 6 N12A1                2 PAL0809      49 Gentoo… Anvers B
## # … with 9 more variables: date_egg <date>, culmen_length
## #   culmen_depth_mm <dbl>, flipper_length_mm <dbl>, body_
## #   sex <chr>, delta_15_n_o_oo <dbl>, delta_13_c_o_oo <db
## #   comments <chr>, and abbreviated variable names ¹dupe_
## #   ²study_name, ³sample_number, ⁴clutch_completion
```

Resources

Original material:

- Package janitor documentation
- GitHub Repository

Further resources:

- exploringdata.org - How to Clean Data: {janitor} Package
- towardsdatascience.com - Cleaning and Exploring Data with the "janitor" Package
- jenrichmond.rbind.io - Cleaning penguins with the janitor package

More on tidying data:
tidyr cheatsheet

Unfortunately, there is no cheatsheet for {`janitor`}.
Here are two alternative tips:

- `ls("package:janitor")` gives you a list of all functions
- or type `janitor::` in your console and you get to scroll up and down the list of all functions in the package.

BTW, this works for all packages ;)

Summary

- Integrate `{janitor}` into your data wrangling and cleaning pipeline
- It works faster than regular functions from the `{tidyverse}`
- It can help you to:
    - **quickly** clean variable names
    - remove empty rows and columns
    - remove constant columns
    - **easily** create **better** tables that work in the tidyverse, can be stored as data frames and are almost worth publishing!

Happy data cleaning!