Data import

Reading in data files

Daniela Palleschi

Wed, Nov 29, 2023

Table of contents

Readings

- **required reading**: Ch. 8 (data import) in Wickham et al. (2023)
- **supplementary reading**: Ch. 4 (data import) in Nordmann & DeBruine (2022)

Review

So far, we've learned how to...

- use Quarto scripts for reproducible data analysis
- load in built-in datasets
- wrangle data with `dplyr` verbs
- visualise distributions and relationships of different variable types

Learning objectives

Today we will learn how to:

- create and store local data files (`.csv`)
- import local data files with the `readr` package
- handle missing values
- change variables to factors

1 Packages

- we will start using the `pacman` package instead of `install.packages()` and `library`
  - the function `p_load()` takes package names as arguments
  - then checks if you have the package installed
    - if you do –> loads the package (just like `library()`)
    - if you don't –> the package is installed and then loaded (like using `install.packages()` + `library()`)
- this saves us from having to install new packages individually

---

```r
# install new packages IN THE CONSOLE!
install.packages("pacman")
```

```r
# load packages
pacman::p_load(tidyverse, # wrangling
               languageR, # linguistic datasets
               janitor, # wrangling
               here, # relative file paths
               patchwork # plot layout
               )
```

- we now have `tidyverse` loaded, and the new packages `janitor` and `here` installed and loaded
  - to find out more about these packages, try typing `?janitor` and `?here` in the Console.

2 CSV: Comma separated value

- there are many different file types that data can take, e.g., .xlsx, .txt, .csv, .tsv.
- `csv` is the most typical data file type, and stands for: Comma Separated Values.
- this is what a simple CSV file looks like when viewed as raw text

```
Student ID,Full Name,favourite.food,mealPlan,AGE
1,Sunil Huffmann,Strawberry yoghurt,Lunch only,4
2,Barclay Lynn,French fries,Lunch only,5
3,Jayendra Lyne,N/A,Breakfast and lunch,7
4,Leon Rossini,Anchovies,Lunch only,
5,Chidiegwu Dunkel,Pizza,Breakfast and lunch,five
6,Güvenç Attila,Ice cream,Lunch only,6
```

- the first row (the "header row") contains the columns names
- the subsequent rows contain the data
- How many variables are there? how many observations?

2.1 Tidy data

- you want your data to be *tidy*
  - tidy data is rectangular, and:
  - each column represents a variable
  - each row an observation
  - each cell a data point (**?@fig-tidy-data**).

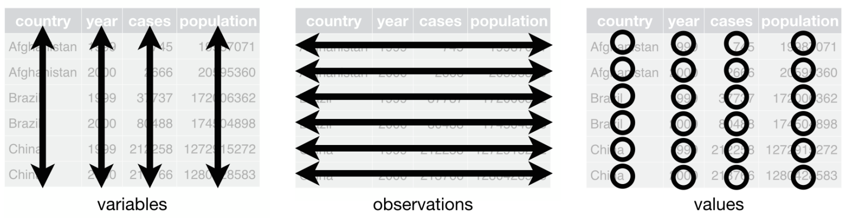


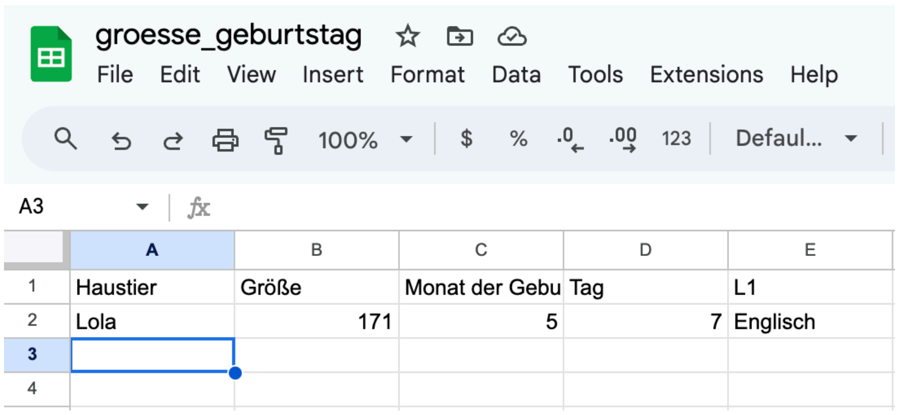
Figure 1: Source: Wickham et al. (2023) (all rights reserved)

RProjects

- make sure you're working in the class RProject!
- if you aren't, follow the exercise on the course website here

3 Spreadsheet to csv

- let's collect some toy data in a spreadsheet which we will then save as a CSV file and load into R
  - Click here to go to an editable spreadsheet
  - Enter the relevant information about yourself, or make up some data: the name of a pet you have/had, height, birth month and day, plus your first language. If you have no pet, leave the cell blank.



	A	B	C	D	E
1	Haustier	Größe	Monat der Gebu	Tag	L1
2	Lola	171	5	7	Englisch
3					
4					

Figure 2: Our spreadsheet

3.0.1 Save csv

- save the spreadsheet as `groesse_geburtstag.csv` to your computer, directly in a folder called `daten` in our project directory

Aufgabe 3.1: `Saving a CSV`

Example 3.1.

1. Create a new folder called `daten` in your project folder (if you haven't already).
2. Download the Google Sheet and save it in your `daten` folder as `groesse_geburtstag.csv`
3. Go to your `daten` folder and check that the CSV file is there.

4 The `readr` package

- we now have to *read in the data*
- we have to use a function that reads CSV data, and to specify *where* the data is in our RProject folder
- the `readr` package (part of `tidyverse`) can load in most data types, and has multiple functions for different data types

---

```r
read_csv(here::here("daten", "groesse_geburtstag.csv"))
```

Table 1: Data from the groesse_geburtstag.csv file as a table.

Haustier	Größe	Monat der Geburt	Tag	L1
Lola	171	5	7	Englisch
NA	168	11	26	Deutsch
N/A	182	4	15	Deutsch

Aufgabe 4.1: `readr`

Example 4.1.

1. Import the `groesse_geburtstag.csv` dataset and save it as an object called

`df_groesse`

   - `df_` is short for DataFrame; it's a good idea to use a prefix before object names so we know what each object contains

2. When importing data with `read_csv`, some information is printed in the Console. What is printed?
3. Explore the dataset using the functions like `summary()` or `head()`
4. Do you see anything odd?

5 The `here` package

- how did R know exactly where to find the `daten` folder?
- our *working directory* is set to the location of our RProject within our computer
  - whenever we want to access data in our RProject, we should nest our filepath in `here()`
- to see where `here()` is starting from, run `here()`

```
here()
```

```
[1] "/Users/danielapalleschi/Documents/IdSL/Teaching/WiSe2324/B.A./r4ling"
```

- the output will look different for all of our machines, because it's relative to where we placed our project folder

---

6 Working with data

6.1 Missing values

- you might've noticed some `NA` or `N/A` values
  - `N/A` was written as text in one of our observations, and so R reads it as such
  - `NA`s in R refer to missing data ("Not Available")
  - true missing values are completely empty, so having `N/A` written in our `df_groesse` data is not actually read as a missing value
  - to fix this, we can use the argument `na =` for the `read_csv()` function, which tells `read_csv()` which values it should equate with missing values

Figure 3: Image source: Allison Horst (all rights reserved)

```r
# force "N/A" to missing values
df_groesse <- read_csv(here::here("daten", "groesse_geburtstag.csv"),
                        na = "N/A")
```

```r
# print the head of the data set
head(df_groesse)
```

```
# A tibble: 3 x 5
  Haustier Größe `Monat der Geburt`   Tag L1
  <chr>    <dbl>              <dbl> <dbl> <chr>
1 "Lola"     171                  5     7 Englisch
2 ""         168                 11    26 Deutsch
3  <NA>      182                  4    15 Deutsch
```

- the value that was formerly `""` is read as an `NA`
- but what about the empty cell? We have now overwritten `read_csv()` reading empty cells as `NA`
    - now we want to tell `read_csv()` to read *more than one* type of input as `NA`, i.e., we want to tell it to read `""` *and* `"N/A"` as `NA`
    - for this, we use our always useful concatenate function: `c()`.

```r
# force "N/A" and empty cells to missing values
df_groesse <- read_csv(here::here("daten", "groesse_geburtstag.csv"),
                        na = c("N/A",""))
```

```r
# print the head of the data set
head(df_groesse)
```

```
# A tibble: 3 x 5
  Haustier Größe `Monat der Geburt`   Tag L1
  <chr>    <dbl>              <dbl> <dbl> <chr>
1 Lola       171                  5     7 Englisch
2 <NA>       168                 11    26 Deutsch
3 <NA>       182                  4    15 Deutsch
```

6.2 Column names

- one column name in our data is surrounded by backticks (e.g., `` `Monat der Geburt` ``)

    - this is because it contains an empty space, which is not syntactically valid
    - a quick fix is to the function `clean_names()` from the `janitor` package, which we've already loaded in

```r
clean_names(df_groesse)
```

```
# A tibble: 3 x 5
  haustier grosse monat_der_geburt   tag l1
  <chr>     <dbl>            <dbl> <dbl> <chr>
1 Lola        171                5     7 Englisch
2 <NA>        168               11    26 Deutsch
3 <NA>        182                4    15 Deutsch
```

- that looks better! But if you now run `head(df_groesse)`, do you see the cleaned column names?
- you shouldn't because when we pass an object through a function, the object is not 'updated'
    - so we have to again assign the object using the assignment operator `<-`

```r
df_groesse <- janitor::clean_names(df_groesse)
```

6.3 Pipes

- pipes are placed at the end of function call when the result of this function should be passed through a subsequent function. Pipes can be read as "and then…".

```r
read_csv(here::here("daten", "groesse_geburtstag.csv")) |>
  head()
```

```
# A tibble: 3 x 5
  Haustier Größe `Monat der Geburt`   Tag L1
  <chr>    <dbl>              <dbl> <dbl> <chr>
1 Lola       171                  5     7 Englisch
2 <NA>       168                 11    26 Deutsch
3 N/A        182                  4    15 Deutsch
```

There are currently 2 pipes that can be used in R.

1. the `magrittr` package pipe: `%>%`

2. the new native R pipe: `|>`

- there aren't any major differences that are important for our current uses
- you can use the keyboard shortcut `Cmd/Ctrl + Shift/Strg + M` to produce a pipe

Aufgabe 6.1: pipes

Example 6.1.

1. Load the `groesse_geburtstag.csv` dataset again with fixed `NA`s *and then*
    - Use a pipe to call `clean_names()` on the dataset, *and then*
    - call the `head()` function
    - check the number of observations and variables, is there an issue?
2. Load the `groesse_geburtstag.csv` dataset again with fixed `NA`s, saving it as the object `df_groesse`, *and then*
    - use a pipe to call `clean_names()` on the data set
3. Why shouldn't you use a pipe and the `head()` function when you're saving the dataset as an object?

6.4 Other file types

- `readr` has other functions which are also easy to use, you just have to know when to use which ones
- `read_csv2()` reads semicolon-separated csv files (`;`)
    - this file type is common in countries that use `,` as the decimal marker (like Germany)
- `read_tsv()` reads tab-delimited files
- `read_delim()` function reads in files with any delimiter
    - it will try to guess the delimiter unless you specify it with the argument `delim =` (e.g., `read_delim(groesse_geburtstag.csv, delim = ",")`)

7 Working with variables

- the major columns types to know are `numerical` and `factor` (categorical)
- factors contain *categories* or *groups* of data, but can sometimes *look* like `numerical` data

  - for example, our column `month` contains numbers, but it could also contain the name of each month
  - a good way to know which is which: it makes sense to calculate the mean of a `numerical` variable, but not of a `factor`
  - for example, it makes sense to calculate our average height, but not our average birth month

`as_factor()`

- we can use the `as_factor()` function to change a variable type to factor
- We can either use base R syntax to do this by using an `$` to index a column in a dataframe:

```
# mit base R
df_groesse$monat_der_geburt <- as_factor(df_groesse$monat_der_geburt)
```

- or we can use `tidyverse` syntax and the `mutate()` function

```
# mit tidyverse
df_groesse <-
  df_groesse |>
  mutate(monat_der_geburt = as_factor(monat_der_geburt))
```

Learning objectives

Today we learned how to...

- import local data files with the `readr` package
- handle missing values
- change variables to factors

Let's now put this new knowledge to use.

Homework

Let's now practice using the `readr` package and wrangling our data.

`readr` functions

1. What function would you use to read a file where fields were separated with "`|`"?
2. What arguments do `read_csv()` and `read_tsv()` have in common?
3. Which function(s) could you use to load in a dataset with a semicolon (`;`) as delimiter?

Data wrangling

Re-load the `groesse_geburtstag.csv` file. Use pipes to also use the `clean_names` function, and to make the following changes in the object `df_groesse`:

1. Convert the variable `l1` to a factor.
2. Rename
    - `grosse` to `groesse`
    - `monat_der_geburt` to `geburtsmonat`

Plots

1. produce a scatterplot using our `df_groesse` dataset, visualising the relationship between our birth day and our birth days (this doesn't make sense to compare, but it's just an exercise). Set the colour and shape to correspond to `L1`. Add a plot title.
2. Find your birthday on the plot.
3. Produce a barplot showing the number of observations per `l1`

Session Info

Hergestellt mit R version 4.3.0 (2023-04-21) (Already Tomorrow) und RStudioversion 2023.9.0.463 (Desert Sunflower).

```
sessionInfo()
```

```
R version 4.3.0 (2023-04-21)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.2.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;
```

```
locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: Europe/Berlin
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
 [1] magick_2.7.4    patchwork_1.1.3 here_1.0.1      janitor_2.2.0
 [5] languageR_1.5.0 lubridate_1.9.2 forcats_1.0.0   stringr_1.5.0
 [9] dplyr_1.1.3     purrr_1.0.2     readr_2.1.4     tidyr_1.3.0
[13] tibble_3.2.1    ggplot2_3.4.3   tidyverse_2.0.0

loaded via a namespace (and not attached):
 [1] utf8_1.2.3       generics_0.1.3   stringi_1.7.12   hms_1.1.3
 [5] digest_0.6.33    magrittr_2.0.3   evaluate_0.21    grid_4.3.0
 [9] timechange_0.2.0 fastmap_1.1.1    rprojroot_2.0.3  jsonlite_1.8.7
[13] fansi_1.0.4      scales_1.2.1     cli_3.6.1        crayon_1.5.2
[17] rlang_1.1.1      bit64_4.0.5      munsell_0.5.0    withr_2.5.0
[21] yaml_2.3.7       parallel_4.3.0   tools_4.3.0      tzdb_0.4.0
[25] colorspace_2.1-0 pacman_0.5.1     vctrs_0.6.3      R6_2.5.1
[29] lifecycle_1.0.3  snakecase_0.11.0 bit_4.0.5        vroom_1.6.3
[33] pkgconfig_2.0.3  pillar_1.9.0     gtable_0.3.4     Rcpp_1.0.11
[37] glue_1.6.2       xfun_0.39        tidyselect_1.2.0 rstudioapi_0.14
[41] knitr_1.44       htmltools_0.5.5  rmarkdown_2.22   compiler_4.3.0
```

Literaturverzeichnis

Nordmann, E., & DeBruine, L. (2022). *Applied data skills.* Zenodo. https://doi.org/10.5281/zenodo.6365078

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (2023). *R for Data Science* (2nd ed.).