

Datenimport

Einlesen lokaler Datendateien

Daniela Palleschi

Humboldt-Universität zu Berlin

Di. den 04.06.2024

Lernziele

Heute werden wir lernen, wie man:

- lokale Datendateien (. **CSV**) erstellen und speichern
- lokale Datendateien mit dem Paket **readr** importieren
- mit fehlenden Werten umzugehen
- Variablen in Faktoren umwandeln

Lesungen

- Kurs-Website: [Kap. 5 - Datenvisualisierung 2](#)
- [Kap. 8 \(Datenimport\)](#) in Wickham et al. ([2023](#))
- [Kap. 4 \(Datenimport\)](#) in Nordmann & DeBruine ([2022](#))

Wiederholung

Bis jetzt haben wir gelernt, wie man...

- Quarto-Skripte für die reproduzierbare Datenanalyse zu verwenden
- eingebaute Datensätze zu laden
- Daten mit **dplyr**-Verben zu verarbeiten
- Verteilungen und Beziehungen zwischen verschiedenen Variablentypen zu visualisieren

Einrichtung

Pakete mit **pacman**

- wir fangen an, das Paket **pacman** anstelle von **install.packages()** und **library** zu benutzen
 - die Funktion **p_load()** nimmt Paketnamen als Argumente
 - prüft dann, ob Sie das Paket installiert haben
 - wenn ja → lädt das Paket (genau wie **library()**)
 - wenn nicht → wird das Paket installiert und dann geladen (wie mit **install.packages() + library()**)
- dies erspart uns die individuelle Installation neuer Pakete

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

1 # load packages (in your script)
2 pacman::p_load(
3     tidyverse, # wrangling
4     janitor, # tidying
5     here # relative file paths
6 )
```

- wir haben jetzt **tidyverse** geladen und die neuen Pakete **janitor** und **here** installiert und geladen
 - Um mehr über diese Pakete herauszufinden, geben Sie **?janitor** und **?here** in der Konsole ein.

daten-Ordner

- fügen Sie Ihrem Projektverzeichnis einen Ordner mit dem Namen **daten** hinzu (der *genau* gleich geschrieben ist).

RProjects

- Stellen Sie sicher, dass Sie in der Klasse RProject arbeiten!
- Falls nicht, folgen Sie der Übung auf der Kurs-Website [hier](#)

CSV: Comma separated value

- Es gibt viele verschiedene Dateitypen, die Daten annehmen können, z. B. .xlsx, .txt, .csv, .tsv
- **.CSV** ist der typischste Dateityp und steht für: Comma Separated Values
- So sieht eine einfache CSV-Datei aus, wenn man sie als Rohtext betrachtet

```
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
```

- die erste Zeile (die “Kopfzeile”) enthält die Spaltennamen
- die folgenden Zeilen enthalten die Daten
- Wie viele Variablen gibt es? Wie viele Beobachtungen?

‘Tidy’ Daten

- Sie wollen, dass Ihre Daten *aufgeräumt* sind
 - aufgeräumte Daten sind rechteckig, und:
 - jede Spalte steht für eine Variable
 - jede Zeile eine Beobachtung
 - jede Zelle ein Datenpunkt ([Abbildung 1](#))

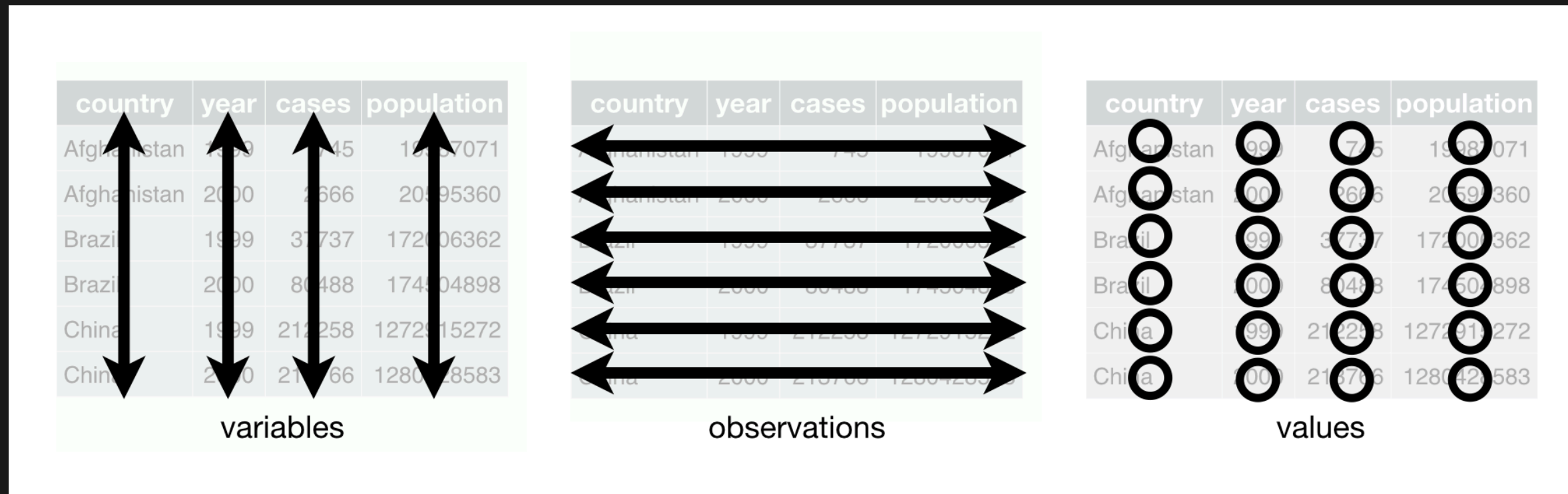


Abbildung 1: Source: Wickham et al. (2023) (all rights reserved)

Tabelle zu csv

- Lassen Sie uns einige Spielzeugdaten in einer Tabellenkalkulation sammeln, die wir dann als CSV-Datei speichern und in R laden werden
 - Klicken Sie [hier](#), um zu einem bearbeitbaren Arbeitsblatt zu gelangen.
 - Geben Sie die relevanten Informationen über sich selbst ein, oder erfinden Sie einige Daten: den Namen eines Haustiers, das Sie haben/hatten, Größe, Geburtsmonat und -tag sowie Ihre erste Sprache. Wenn Sie kein Haustier haben, lassen Sie die Zelle leer.

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

Abbildung 2: Klicken Sie [hier](#) zum Hinzufügen und Herunterladen der Tabelle

CSV speichern

- Speichern Sie die Tabelle als **groesse_geburtstag.csv** auf Ihrem Computer, direkt in einem Ordner namens **daten** in unserem Projektverzeichnis

Aufgabe

Aufgabe 1: Saving a CSV

Aufgabe 1

1. Erstellen Sie einen neuen Ordner mit dem Namen **daten** in Ihrem Projektordner (falls Sie das nicht schon getan haben).
2. Laden Sie das Google Sheet herunter und speichern Sie es in Ihrem **daten** Ordner als **groesse_geburtstag.csv**.
3. Gehen Sie zu Ihrem **daten**-Ordner und überprüfen Sie, ob die CSV-Datei dort ist.

Das Paket **readr**

- müssen wir nun *die Daten einlesen*
- wir müssen eine Funktion verwenden, die CSV-Daten liest, und angeben, *wo* sich die Daten in unserem RProject-Ordner befinden
- Das **readr**-Paket (Teil von **tidyverse**) kann die meisten Datentypen einlesen und hat mehrere Funktionen für verschiedene Datentypen

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

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

Größe	Geburtsmonat	L1	Haustier	Was für ein Haustier?
171	5	Englisch	Lola	Hundin
168	11	Deutsch	keine	keine
182	4	Deutsch	N/A	NA
190	8	Deutsch	Knut	Kater
170	10	Deutsch	Emma	Hundin
163	2	Deutsch	Üzgür	Kater
164	7	Italienisch	Fipsy	Katze
167	12	Schwedisch	Anna	Fisch
189	10	Norwegisch	Arvid	Papagei

Aufgabe

Aufgabe 2: readr

Aufgabe 2

1. Importieren Sie den Datensatz “groesse_geburtstag.csv” und speichern Sie ihn als Objekt mit dem Namen `df_groesse`.
 - `df_` ist die Abkürzung für DataFrame; es ist eine gute Idee, ein Präfix vor Objektnamen zu verwenden, damit wir wissen, was jedes Objekt enthält
2. Beim Importieren von Daten mit `read_csv` werden einige Informationen in der Konsole ausgegeben. Was wird gedruckt?
3. Untersuche den Datensatz mit Funktionen wie `summary()` oder `head()`
4. Sehen Sie etwas Ungewöhnliches?

Das Paket **here**

- Woher weiß R genau, wo der Ordner **daten** zu finden ist?
- unser *Arbeitsverzeichnis* ist auf den Ort unseres RProjekts auf unserem Computer festgelegt
 - wann immer wir auf Daten in unserem RProjekt zugreifen wollen, sollten wir unseren Dateipfad in **here()** verschachteln
- um zu sehen, von wo aus **here()** startet, führen Sie **here()** aus

```
1 here()
```

```
[1] "/Users/danielapalleschi/Documents/IdSL/Teaching/SoSe24/B.A./r4ling_bose2024"
```

- Die Ausgabe wird auf allen Rechnern unterschiedlich aussehen, da sie relativ zu dem Ort ist, an dem wir unseren Projektordner abgelegt haben

here package



Image source: Allison Horst (all rights reserved)

Arbeiten mit Daten

Fehlende Werte

- Sie haben vielleicht einige **NA** oder **N/A** Werte bemerkt
 - **N/A** wurde als Text in einer unserer Beobachtungen geschrieben, und so liest R es als solches
 - **NA** in R bezieht sich auf fehlende Daten (“Nicht verfügbar”)
 - Echte fehlende Werte sind komplett leer, so dass **N/A** in unseren **df_groesse**-Daten nicht wirklich als fehlender Wert gelesen wird.
 - Um dies zu beheben, können wir das Argument **na =** für die Funktion **read_csv()** verwenden, das der Funktion **read_csv()** mitteilt, welche Werte sie mit fehlenden Werten gleichsetzen soll

Fehlende Werte

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

```
1 # print the head of the data set
2 head(df_groesse)
```

```
# A tibble: 6 × 5
  Größe Geburtsmonat L1      Haustier `Was für ein Haustier?`
  <dbl>      <dbl> <chr>    <chr>    <chr>
1   171         5 Englisch Lola      "Hundin"
2   168        11 Deutsch keine    "keine"
3   182         4 Deutsch <NA>     ""
4   190         8 Deutsch Knut     "Kater"
5   170        10 Deutsch Emma     "Hundin"
6   163         2 Deutsch Üzgür    "Kater"
```

- der Wert, der vorher `""` war, wird als **NA** gelesen
- aber was ist mit der leeren Zelle? Wir haben jetzt überschrieben, dass `read_csv()` leere Zellen als **NA** liest
 - Nun wollen wir `read_csv()` anweisen, *mehr als eine* Art von Eingabe als **NA** zu lesen, d.h. wir wollen es anweisen, `""` und `"N/A"` als **NA** zu lesen
 - Dazu verwenden wir unsere immer nützliche Verkettungsfunktion: `c()`
 - lassen Sie uns auch 'keine' als **NA**'s einschließen

Fehlende Werte

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

```
1 # print the head of the data set
2 head(df_groesse)
```

A tibble: 6 × 5

	Größe	Geburtsmonat	L1	Haustier	`Was für ein Haustier?`
	<dbl>	<dbl>	<chr>	<chr>	<chr>
1	171	5	Englisch	Lola	Hundin
2	168	11	Deutsch	<NA>	<NA>
3	182	4	Deutsch	<NA>	<NA>
4	190	8	Deutsch	Knut	Kater
5	170	10	Deutsch	Emma	Hundin
6	163	2	Deutsch	Üzgür	Kater

Spaltennamen

- Ein Spaltenname in unseren Daten ist von Backticks umgeben (z.B. ``Was für ein Haustier?``)
 - Das liegt daran, dass er ein Leerzeichen enthält, das syntaktisch nicht gültig ist.
 - Eine schnelle Lösung ist die Funktion `clean_names()` aus dem Paket `janitor`, das wir bereits geladen haben

Spaltennamen

```
1 clean_names(df_groesse)
```

```
# A tibble: 9 × 5
```

	grosse	geburtsmonat	l1	haustier	was_fur_ein_haustier
	<dbl>	<dbl>	<chr>	<chr>	<chr>
1	171	5	Englisch	Lola	Hundin
2	168	11	Deutsch	<NA>	<NA>
3	182	4	Deutsch	<NA>	<NA>
4	190	8	Deutsch	Knut	Kater
5	170	10	Deutsch	Emma	Hundin
6	163	2	Deutsch	Üzgür	Kater
7	164	7	Italienisch	Fipsy	Katze
8	167	12	Schwedisch	Anna	Fisch
9	189	10	Norwegisch	Arvid	Papagei

- Das sieht besser aus! Aber wenn Sie jetzt `head(df_groesse)` ausführen, sehen Sie dann die bereinigten Spaltennamen?
- Sie sollten nicht, denn wenn wir ein Objekt durch eine Funktion übergeben, wird das Objekt nicht ‘aktualisiert’
 - Deshalb müssen wir das Objekt erneut mit dem Zuweisungsoperator `<-` zuweisen.

► Code

Pipes

- Pipes werden am Ende eines Funktionsaufrufs platziert, wenn das Ergebnis dieser Funktion durch eine nachfolgende Funktion weitergegeben werden soll
 - Pipes können als “und dann...” gelesen werden

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

```
# A tibble: 6 × 5
```

	Größe	Geburtsmonat	L1	Haustier	`Was für ein Haustier?`
	<dbl>	<dbl>	<chr>	<chr>	<chr>
1	171	5	Englisch	Lola	Hundin
2	168	11	Deutsch	keine	keine
3	182	4	Deutsch	N/A	<NA>
4	190	8	Deutsch	Knut	Kater
5	170	10	Deutsch	Emma	Hundin
6	163	2	Deutsch	Üzgür	Kater

Pipes

Derzeit gibt es 2 Pipes, die in R verwendet werden können.

1. die **magrittr**-Package-Pipe: **%>%**

2. die neuer (seit 2023) native R-Pipe: **|>**

- es gibt keine großen Unterschiede, die für unsere aktuellen Anwendungen wichtig sind
- Sie können das Tastaturkürzel **Cmd/Ctrl + Shift/Strg + M** verwenden, um eine Pipe zu erzeugen

Aufgabe

Aufgabe 3: pipes

Aufgabe 3

1. Laden Sie den Datensatz `groesse_geburtstag.csv` erneut mit festen **NAs** und dann
 - Benutzen Sie eine Pipe, um `clean_names()` für den Datensatz aufzurufen, *und dann*
 - rufen Sie die Funktion “head()” auf
 - Überprüfen Sie die Anzahl der Beobachtungen und Variablen, gibt es ein Problem?
2. Laden Sie den Datensatz `groesse_geburtstag.csv` erneut mit festen **NAs**, speichern Sie ihn als Objekt `df_groesse`, *und dann*
 - Verwenden Sie eine Pipe, um `clean_names()` auf den Datensatz anzuwenden.
3. Warum sollte man nicht eine Pipe und die Funktion “head()” verwenden, wenn man den Datensatz als Objekt speichert?

Variablentypen

- die wichtigsten Spaltentypen, die man kennen sollte, sind “numerisch” und “Faktor” (kategorisch)
- Faktoren enthalten *Kategorien* oder *Gruppen* von Daten, können aber manchmal *aussehen* wie **numerische** Daten
 - Unsere Spalte “Monat” enthält zum Beispiel Zahlen, aber sie könnte auch den Namen jedes Monats enthalten
 - Es ist sinnvoll, den Mittelwert einer “numerischen” Variable zu berechnen, aber nicht den eines “Faktors”
 - Es ist zum Beispiel sinnvoll, die durchschnittliche Körpergröße zu berechnen, aber nicht den durchschnittlichen Geburtsmonat

as_factor()

- Wir können die Funktion “as_factor()” verwenden, um einen Variablentyp in einen Faktor zu ändern.
- Wir können entweder die R-Basissyntax verwenden, um dies zu tun, indem wir ein **\$** verwenden, um eine Spalte in einem Datenrahmen zu indizieren:

```
1 # mit base R
2 df_groesse$geburtsmonat <- as_factor(df_groesse$geburtsmonat)
```

- oder wir können die Syntax **tidyverse** und die Funktion **mutate()** verwenden




```
1 # mit tidyverse
2 df_groesse <-
3   df_groesse |>
4   mutate(geburtsmonat = as_factor(geburtsmonat))
```

Andere Dateitypen und Begrenzungszeichen

- `readr` hat weitere Funktionen, die ebenfalls einfach zu benutzen sind, man muss nur wissen, wann man welche benutzt
- `read_csv2()` liest Semikolon-getrennte csv-Dateien (`;`)
 - Dieser Dateityp ist in Ländern üblich, die `,` als Dezimaltrennzeichen verwenden (wie Deutschland)
- `read_tsv()` liest Tabulator-getrennte Dateien
- Die Funktion `read_delim()` liest Dateien mit beliebigen Trennzeichen ein.
 - sie versucht, das Trennzeichen zu erraten, es sei denn, Sie geben es mit dem Argument `delim` an (z.B. `read_delim(groesse_geburtstag.csv, delim = ",")`)

Lernziele

Heute haben wir gelernt, wie man...

- lokale Datendateien mit dem Paket `readr` importiert 
- fehlende Werte behandeln 
- Variablen in Faktoren umwandeln 

Lassen Sie uns nun dieses neue Wissen anwenden.

Hausaufgaben

Wir wollen nun üben, das Paket `readr` zu verwenden und unsere Daten zu verarbeiten.

Session Info

Hergestellt mit R version 4.4.0 (2024-04-24) (Puppy Cup) und RStudioversion 2023.12.1.402 (Ocean Storm).

```
1 sessionInfo()
```

```
R version 4.4.0 (2024-04-24)
```

```
Platform: aarch64-apple-darwin20
```

```
Running under: macOS Ventura 13.2.1
```

```
Matrix products: default
```

```
BLAS: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
```

```
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib; LAPACK version 3.12.0
```

```
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:
```

```
stats      graphics  grDevices  utils      datasets  MASS      R6          Rcpp        RcppEigen  RcppArmadillo
```

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* (2. Aufl.).

