Datenimport

Einlesen von Datendateien

Daniela Palleschi

2023-05-09

Inhaltsverzeichnis

Wiederholung

Letzte Woche haben wir…

- den Unterschied zwischen numerischen und kategorialen Datentypen gelernt
- haben wir unsere ersten Diagramme mit `ggplot2` erstellt
- gelernt, welche Diagramme für verschiedene Datentypen geeignet sind

Heutige Ziele

Heute werden wir…

- lernen, wie man einen neuen Datensatz in Augenschein nimmt
- lernen, wie man verschiedene Datentypen importiert
- lernen, wie man Daten von Hand eingibt
- einen neuen Datensatz visualisieren

Lust auf mehr?

- Ch. 8 in Wickham et al. (o. J.)
- Ch. 4 in Nordmann & DeBruine (2022)

1 Daten erforschen

- letzte Woche haben wir mit Daten aus dem R-Paket `palmerpenguings` gearbeitet
  - Daten aus Paketen sind ein guter Weg, um Data Science Tools zu lernen
- aber ihr werdet irgendwann mit euren eigenen Daten arbeiten wollen
- Heute lernen wir die Grundlagen des Einlesens von Datendateien in R

1.1 Pakete

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

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

- wir werden mit dem Paket `pacman` beginnen
  - die Funktion `p_load()` nimmt Paketnamen als Argumente
  - sie prüft dann, ob wir das Paket installiert haben
    - wenn ja, dann lädt sie das Paket
    - wenn nicht, wird das Paket installiert und geladen
- dies erspart uns, jedes Mal neue Pakete zu installieren
- wir haben jetzt `tidyverse` und `patchwork` geladen und die neuen Pakete `janitor` und `here` installiert und geladen

1.2 Dateien

- Speichert die Dateien unter 'daten' in den Moodle-Materialien für diese Woche
- Speichert die Dateien im Ordner `daten` in eurem .RProj für diesen Kurs!

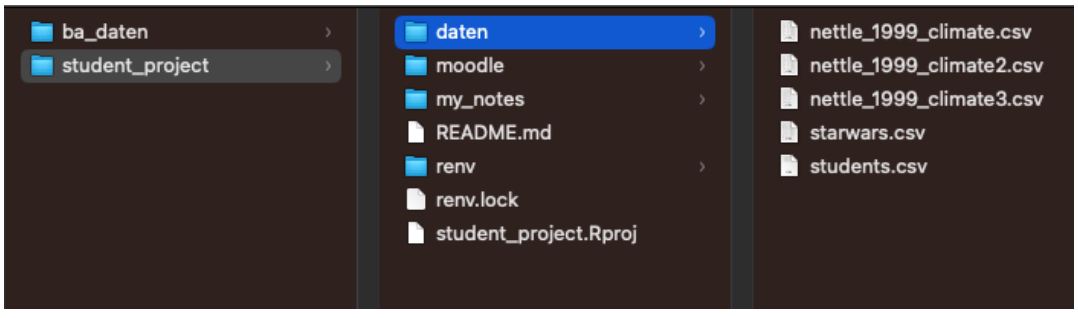


Abbildung 1: 'data' file path

1.3 Eingebaute Daten

- Lasst uns zunächst einige eingebaute Daten untersuchen: den `iris`-Datensatz
  - wie viele *Variablen* gibt es?
  - wie viele *Beobachtungen*?

```
# load tidyverse
library(tidyverse)
# read-in iris dataset
data("iris")
```

- iris = Iris
- petal = Blumenblatt
- sepal = Kelchblatt

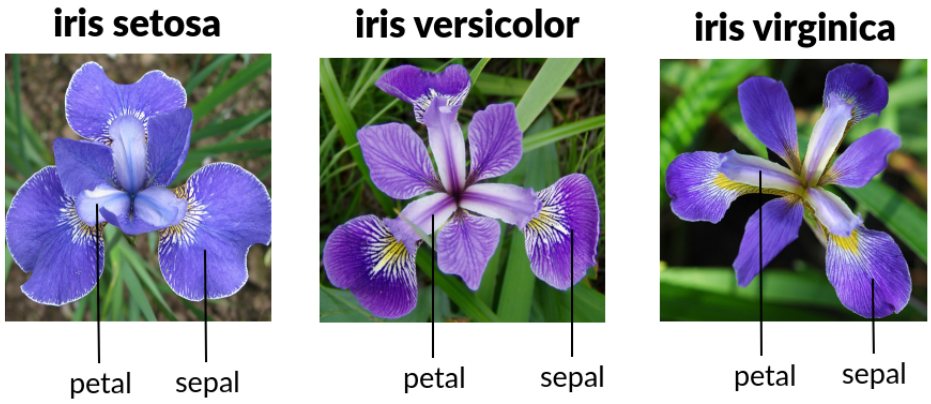


Abbildung 2: Image source: Analytics Vidhya (all rights reserved)

1.4 Einen Datensatz erforschen

- `view()`: Datensatz öffnen
  - Führt dies nur **in der Konsole** aus! Euer Dokument wird nicht gerendert, wenn ihr es in eurem Skript habt
- die Funktion `head()` gibt die ersten 6 Zeilen der Daten aus

```
# print dataset (first 6 rows)
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

1.4.1 `glimpse()`

- EN: glimpse = DE: Einblick
- aus dem Paket `tibble`
- gibt eine seitliche Vorschau des Datenrahmens

```
glimpse(iris)
```

```
Rows: 150
Columns: 5
$ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
$ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
$ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
$ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
$ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, s~
```

```
# print dataset (first 6 rows)
head(iris)
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

1.4.2 `summary()`

- druckt eine Zusammenfassung für jede Variable
  - Minimum, Maximum, Mittelwert von numerischen Variablen

  - Anzahl der Beobachtungen pro Stufe einer kategorischen Variable

Aufgabe 1.1: `table1`

Beispiel 1.1.

1. Versuch, den eingebauten Datensatz `table1` zu laden
2. Untersuche den Datensatz mit den Funktionen, die wir gerade gelernt haben
3. Gibt es irgendetwas Komisches in der Zusammenfassung? Wie könnten wir das beheben?

2 Daten importieren

- wir wollen normalerweise mit unseren *eigenen* Daten arbeiten, nicht mit eingebauten Spielzeugdaten
- wir konzentrieren uns nur auf *rechteckige* Daten (d. h. aufgeräumte Daten)
- Es gibt viele verschiedene Dateitypen, die Daten annehmen können, z. B. `.xlsx`, `.txt`, `.csv`, `.tsv`
- `csv` ist der am häufigsten verwendete Dateityp: Kommagetrennte Werte (EN: `Comma Separated Value`)
  - Versucht, `.xlsx` zu vermeiden; wenn ihr einen Excel-Datensatz habt, versucht, ihn als `.csv` zu speichern, bevor ihr ihn in R lest
- So sieht eine einfache CSV-Datei aus

```
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?

2.1 `readr` Paket

- dieselben Daten können als Tabelle angezeigt werden, so wie wir es mit `penguins` und `iris` gemacht haben
  - aber zuerst müssen wir die Daten *einlesen*
- das `readr` Paket (Teil von `tidyverse`) kann die meisten Datentypen einlesen

```
read_csv(here::here("daten", "students.csv"))
```

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

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	NA
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five
6	Güvenç Attila	Ice cream	Lunch only	6

2.2 `here` Paket

- 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 `here::here()` verwenden.
- um zu sehen, von wo aus `here()` startet, führt `here()` aus

```
here()
```

```
[1] "/Users/danielapalleschi/Documents/IdSL/Teaching/SoSe23/BA/ba_daten"
```

- dies wird auf allen unseren Rechnern anders aussehen
  - Was jedoch *gleich* sein sollte, ist unsere Ordnerstruktur innerhalb unserer Projekte (z. B. `daten/students.csv`).

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

here Paket

Vor dem `here`-Paket mussten wir R explizit mitteilen, wo sich eine Datei auf unserem Computer befindet (z.B., `/Users/danielapalleschi/Documents/IdSL/Teaching/SoSe23/BA/ba_daten/daten/students.csv`), oder die Funktion `setwd()` (set Working Directory) benutzen, um R mitzuteilen, wo alle Dateien zu finden sind (z.B. `setwd(/Users/danielapalleschi/Documents/IdSL/Teaching/SoSe23/BA/ba_daten)`). Glücklicherweise brauchen wir diese absoluten Dateipfade oder `setwd()` nie zu verwenden!
Aus der here-Paketdokumentation:

> The goal of the here package is to enable easy file referencing in project-oriented workflows. In contrast to using `setwd()`, which is fragile and dependent on the way you organize your files, here uses the top-level directory of a project to easily build paths to files.

Das bedeutet, dass wir nun den ***großen*** Vorteil haben, dass wir unseren Projektordner überall hin verschieben können und unser Dateipfad immer noch relativ zu dem Ort ist, an den wir unseren Projektordner verschoben haben. Das bedeutet, dass das Projekt unabhängig davon läuft, wo es sich auf eurem Computer befindet. Ihr könnt auch jemandem den Projektordner schicken, und alles sollte auf dessen Rechner laufen!

Aufgabe 2.1: `table1`

Beispiel 2.1.

1. Importiert den Datensatz `students.csv` und speichert ihn als Objekt mit dem Namen `df_students`.
    - `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. Untersucht den Datensatz mit den Funktionen, die wir gerade gelernt haben
4. Fällt Ihnen etwas Ungewöhnliches auf?

2.3 Fehlende Werte

- Die Datentransformation bezieht sich auf die “Korrektur” unserer Daten, wenn sie nicht “ordentlich” sind.

- in unserem `df_students` Datenrahmen habt ihr vielleicht einige `NA` oder `N/A` Werte bemerkt
  - `N/A` wurde als Text geschrieben und wird daher von R als solcher gelesen.
  - `NA` in R bezieht sich auf fehlende Daten (EN: `N`ot `A`vailable = DE: "Nicht verfügbar")
- letzte Woche haben wir einige Warnmeldungen gesehen, als wir unsere Streudiagramme erstellt haben
  - Diese Warnungen informierten uns über fehlende Werte (`NA`s), die nicht geplottet wurden
- Echte fehlende Werte sind völlig leer, so dass die Angabe `N/A` in unseren `df_students`-Daten nicht wirklich als fehlender Wert gelesen wird.
- Um dies zu beheben, können wir das Argument `na =` für die Funktion `read_csv()` verwenden
  - Dieses Argument teilt `read_csv()` mit, welche Werte mit fehlenden Werten gleichgesetzt werden sollen.

```
df_students <- read_csv(here::here("daten", "students.csv"),
                        na = "N/A")
head(df_students)
```

```
# A tibble: 6 x 5
  `Student ID` `Full Name`      favourite.food     mealPlan            AGE
         <dbl> <chr>            <chr>              <chr>               <chr>
1            1 Sunil Huffmann   Strawberry yoghurt Lunch only          "4"
2            2 Barclay Lynn     French fries       Lunch only          "5"
3            3 Jayendra Lyne    <NA>               Breakfast and lunch "7"
4            4 Leon Rossini     Anchovies          Lunch only          ""
5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch "five"
6            6 Güvenç Attila    Ice cream          Lunch only          "6"
```

- jetzt wird der Wert, der vorher `N/A` war, als `NA` gelesen
  - aber was ist mit der leeren Zelle?
- Wir haben jetzt überschrieben, dass `read_csv()` leere Zellen als `NA` liest.
  - Wie können wir `read_csv()` anweisen, *mehr als eine* Art von Eingabe als `NA` zu lesen?
  - d.h. wir wollen, dass es `""` *und* `"N/A"`x als `NA` liest

```r
df_students <- read_csv(here::here("daten", "students.csv"),
                        na = c("N/A",""))
head(df_students)
```

```
# A tibble: 6 x 5
  `Student ID` `Full Name`      favourite.food     mealPlan            AGE
         <dbl> <chr>            <chr>              <chr>               <chr>
1            1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
2            2 Barclay Lynn     French fries       Lunch only          5
3            3 Jayendra Lyne    <NA>               Breakfast and lunch 7
4            4 Leon Rossini     Anchovies          Lunch only          <NA>
5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
6            6 Güvenç Attila    Ice cream          Lunch only          6
```

2.4 Spaltennamen

- Wenn wir `df_students` in der Konsole ausgeben, werden wir sehen, dass die ersten beiden Spaltennamen von Backticks umgeben sind (z.B. `` `Student ID` ``)
  - Das liegt daran, dass sie ein Leerzeichen enthalten, das syntaktisch nicht gültig ist (Variablennamen müssen mit einem Buchstaben beginnen und dürfen keine Leer- oder Sonderzeichen enthalten).
- eine schnelle Lösung ist die Funktion `clean_names()` aus dem Paket `janitor`

```r
janitor::clean_names(df_students)
```

```
# A tibble: 6 x 5
  student_id full_name        favourite_food     meal_plan           age
       <dbl> <chr>            <chr>              <chr>               <chr>
1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
2          2 Barclay Lynn     French fries       Lunch only          5
3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
4          4 Leon Rossini     Anchovies          Lunch only          <NA>
5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
6          6 Güvenç Attila    Ice cream          Lunch only          6
```

- Das sieht besser aus!
  - aber wenn wir jetzt `head(df_students)` ausführen, sehen wir dann die bereinigten Spaltennamen?

- Wenn wir ein Objekt durch eine Funktion übergeben, wird das Objekt nicht "aktualisiert".
  - so müssen wir das Objekt erneut zuweisen

```
df_students <- janitor::clean_names(df_students)
```

- aber wir wissen oft, dass wir mehrere Funktionen (`read_csv()`, `clean_names()`) auf demselben Objekt ausführen wollen
  - können wir das mit Pipes tun

2.5 Pipes

- Pipes werden am Ende eines Funktionsaufrufs gesetzt, wenn das Ergebnis dieser Funktion durch eine nachfolgende Funktion weitergegeben werden soll
  - sie können als "und dann…" gelesen werden

```
read_csv(here::here("daten", "students.csv")) %>%
  head()
```

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

- Es gibt derzeit 2 Pipes, die in R verwendet werden können
  1. die `magrittr`-Paket-Pipe: `%>%`
  2. die neue native R-Pipe: `|>`
- Bis jetzt habe ich noch keinen großen Unterschied zwischen den beiden entdeckt, aber im Moment bleibe ich bei `%>%`

Aufgabe 2.2: pipes

Beispiel 2.2.

1. Ladet den `students.csv`-Datensatz erneut mit festen `NA`s *und dann*
   - Benutze eine Pipe, um `clean_names()` auf dem Datensatz aufzurufen, *und dann*
   - Ruft man die Funktion `head()` auf
2. Ladet den Datensatz `students.csv` erneut mit festen `NA`s und speichert ihn als Objekt `df_students`, *und dann*
   - Verwendet 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?

2.6 Andere Dateitypen

- Sobald wir mit `read_csv()` vertraut sind, sind die anderen Funktionen von `readr` einfach zu benutzen
  - man muss nur wissen, wann man welche benutzt
- `read_csv2()` liest Semikolon-getrennte Dateien
  - Diese verwenden `;` anstelle von `,` zur Trennung von Feldern und sind in Ländern üblich, die `,` als Dezimalzeichen verwenden
- `read_tsv()` liest Tabulator-getrennte Dateien
- `read_delim()` liest Dateien mit beliebigen Begrenzungszeichen ein
  - versucht, das Trennzeichen zu erraten
  - es sei denn, ihr gebt es mit dem Argument `delim =` an (z. B. `read_delim("students.csv", delim = ",")`)

Andere habe ich noch nicht gebraucht:

- `read_fwf()` liest Dateien mit fester Breite
- `read_table()` liest eine gängige Variante von Dateien mit fester Breite, bei der die Spalten durch Leerzeichen getrennt sind
- `read_log()` liest Log-Dateien im Apache-Stil

Aufgabe 2.3: filetypes

Beispiel 2.3.

1. Welche Funktion wird verwendet, um eine Datei zu lesen, deren Felder durch `|` getrennt sind?
2. Welche Argumente haben `read_csv()` und `read_tsv()` gemeinsam?
3. Wie würdet ihr einen Datensatz mit einem Semikolon als Begrenzungszeichen einlesen?
4. Ladet den Datensatz `nettle_1999_climate.csv` ein
   - wie viele Variablen gibt es?
5. Ladet den Datensatz `nettle_1999_climate2.csv` ein
   - Wie viele Variablen gibt es? Ist das richtig?
6. Ladet den Datensatz `nettle_1999_climate3.csv` ein
   - wie viele Variablen sind vorhanden? Ist dies richtig?

3 Dateneingabe

- wenn wir kleine Datenmengen sammeln, möchten wir sie vielleicht von Hand in R eingeben
  - Es gibt zwei nützliche Funktionen, die uns dabei helfen, gesammelte Daten zu nehmen und ein "Tibble" zu erstellen
- `tibbles` sind *moderne* Datenrahmen, macht euch noch keine Gedanken über die Definition eines Tibbles
- Sammeln wir die Initialen, die Körpergröße (cm) und das Geburtsdatum (ttmm) von allen

```
i <- "DP" # a "string"
h <- 171 # a number
m <- 05 # a number
d <- 07
```

3.1 `tibble()`

```
tibble(
  initial = i,
  height = h,
  month = "Mai",
  day = 7)
```

```
# A tibble: 1 x 4
  initial height month   day
  <chr>    <dbl> <chr> <dbl>
1 DP         171 Mai       7
```

3.2 `tribble()`

- es könnte einfacher sein, die Daten Zeile für Zeile einzugeben
  - dies ist mit einem *transponierten* Tibble (`Tribble`) möglich

```
tribble(
  ~initial, ~height, ~month, ~day,
  "DP", 171, 07, 05
)
```

```
# A tibble: 1 x 4
  initial height month   day
  <chr>    <dbl> <dbl> <dbl>
1 DP         171     7     5
```

Aufgabe 4.1: tibbles

Beispiel 3.1.

1. Speichert das Tibble (d.h. den Datenrahmen) als das Objekt `df_wir`.
2. Den Datensatz untersuchen (z.B. Zusammenfassungen drucken)
3. Was kommt Ihnen seltsam vor?

```
   initial              height        month         day
 Length:1           Min.   :171   Min.   :5   Min.   :7
 Class :character   1st Qu.:171   1st Qu.:5   1st Qu.:7
 Mode  :character   Median :171   Median :5   Median :7
```

```
                  Mean   :171   Mean   :5   Mean   :7
                  3rd Qu.:171   3rd Qu.:5   3rd Qu.:7
                  Max.   :171   Max.   :5   Max.   :7
```

4 Arbeiten mit Variablen

- In einem Datenrahmen sind die Variablen in Spalten organisiert

4.1 Variablentypen

- `readr` errät den Typ der Daten, die jede Spalte enthält
  - Die wichtigsten Spaltentypen, die man kennen sollte, sind `numerisch` und `kategorisch` (= `factor`).
- `factor`s enthalten *Kategorien* oder *Gruppen* von Daten, können aber manchmal wie `numerische` Daten *aussehen*.
  - Unsere Spalte `month` enthält zum Beispiel Zahlen, aber sie kann auch den Namen jedes Monats enthalten.
  - Es ist sinnvoll, den Mittelwert einer `numerischen` Variable zu berechnen, aber nicht den eines `factor`s.
    - Es ist sinnvoll, den Mittelwert der Körpergröße zu berechnen, aber nicht den Mittelwert des Geburtsmonats.

```
df_wir$month <- as_factor(df_wir$month)
```

4.2 Indizierung

- manchmal wollen wir auf eine bestimmte Variable (Spalte) in einem Datenrahmen zugreifen
  - Bei Verwendung von Base R tun wir das mit `$`: `Datenrahmen$Variable`

```
df_wir$height
```

```
[1] 171
```

- und wir können dies als Argument für eine Funktion verwenden

  - Versuche, die minimale und maximale Höhe in unserer Gruppe zu finden.
  - die Summe (EN: `sum`) unserer Höhen berechnen

Aufgabe 4.1: tibbles

Beispiel 4.1.

1. Umrechnung von `df_wir$month` in einen Faktor
2. Berechne unsere mittlere Höhe mit der Funktion `mean()` und der Indizierung
3. Berechne die Summe unserer Höhen.

4.3 In Datei schreiben

- wir können unseren Datenrahmen auch speichern, so dass wir später darauf zurückkommen können mit `write_csv(object_name, "desired_filename.csv")`
  - oder, mit `here`: `write_csv(object_name, here::here("ordner", "desired_filename.csv"))`
- Seid ***sehr vorsichtig***! Wenn ihr einen existierenden Dateinamen verwendet, könnt ihr einen bereits existierenden Datensatz überschreiben

```
write_csv(df_wir, file = here("daten", "wir.csv"))
```

5 Übungen

Hier findet ihr einige vertiefende Übungen.

5.1 Import/Export

1. Ladet den eingebauten `starwars`-Datensatz, der Informationen über Star Wars Charaktere enthält.
2. Exportiert die Daten als `csv` Datei mit dem Namen `starwars` in euren `daten` Ordner.
3. Importiere die `starwars.csv` Datei mit `read_csv()`.

5.2 Spaltentypen konvertieren

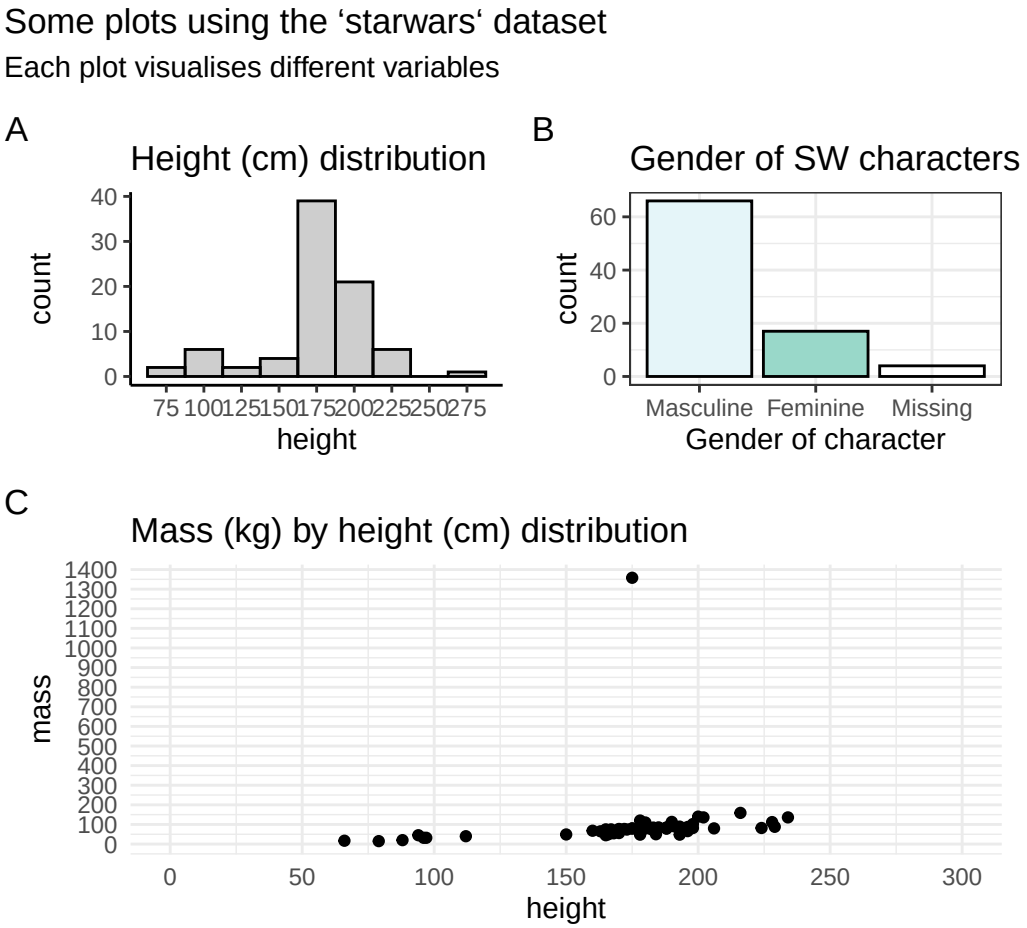
1. Konvertiere die folgenden Variablen in Faktoren:
   - `hair_color`
   - `skin_color`

- `eye_color`
- `sex`
- `gender`
- `homeworld`
- `species`

5.3 Plots

1. Erstellt die folgenden drei Diagramme und beschreibt kurz, was sie zeigen und welche Schlussfolgerungen daraus gezogen werden können.
2. Erstellt ein weiteres Diagramm eurer Wahl aus dem "Starwars"-Datensatz. Fügt sie in das Diagrammgitter ein (ihr müsst die Syntax anpassen). Beschreibt was sie zeigt.

```
(fig_sw_height + fig_sw_gender) /
  fig_sw_height_mass +
  plot_layout(nrow = 2,  heights = c(.4,.6)) +
  plot_annotation(
    title = "Some plots using the `starwars` dataset",
    subtitle = "Each plot visualises different variables",
    tag_levels = "A")
```



Heutige Ziele

Heute haben wir…

- gelernt, wie man einen neuen Datensatz in Augenschein nimmt
- gelernt, wie man verschiedene Datentypen importiert
- gelernt, wie man Daten von Hand eingibt
- einen neuen Datensatz visualisiert

Session Info

Hergestellt mit R version 4.2.3 (2023-03-15) (Shortstop Beagle) und RStudioversion 2023.3.0.386 (Cherry Blossom).

```r
sessionInfo()
```

```
R version 4.2.3 (2023-03-15)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.2.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.2-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

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

other attached packages:
 [1] kableExtra_1.3.4 patchwork_1.1.2  here_1.0.1       janitor_2.2.0
 [5] lubridate_1.9.2  forcats_1.0.0    stringr_1.5.0    dplyr_1.1.1
 [9] purrr_1.0.1      readr_2.1.4      tidyr_1.3.0      tibble_3.2.1
[13] ggplot2_3.4.2    tidyverse_2.0.0

loaded via a namespace (and not attached):
 [1] tidyselect_1.2.0   xfun_0.38          snakecase_0.11.0   colorspace_2.1-0
 [5] vctrs_0.6.1        generics_0.1.3     viridisLite_0.4.1  htmltools_0.5.5
 [9] yaml_2.3.7         utf8_1.2.3         rlang_1.1.0        pillar_1.9.0
[13] glue_1.6.2         withr_2.5.0        RColorBrewer_1.1-3 bit64_4.0.5
[17] lifecycle_1.0.3    munsell_0.5.0      gtable_0.3.3       rvest_1.0.3
[21] evaluate_0.20      labeling_0.4.2     knitr_1.42         tzdb_0.3.0
[25] fastmap_1.1.1      parallel_4.2.3     fansi_1.0.4        scales_1.2.1
[29] webshot_0.5.4      vroom_1.6.1        jsonlite_1.8.4     farver_2.1.1
[33] systemfonts_1.0.4  bit_4.0.5          hms_1.1.3          digest_0.6.31
[37] stringi_1.7.12     grid_4.2.3         rprojroot_2.0.3    cli_3.6.1
[41] tools_4.2.3        magrittr_2.0.3     pacman_0.5.1       crayon_1.5.2
[45] pkgconfig_2.0.3    xml2_1.3.3         timechange_0.2.0   rmarkdown_2.21
[49] svglite_2.1.1      httr_1.4.5         rstudioapi_0.14    R6_2.5.1
[53] compiler_4.2.3
```

Literaturverzeichnis

Nordmann, E., & DeBruine, L. (2022). *Applied Data Skills* (Version 2.0). Zenodo. https://doi.org/10.5281/zenodo.6365078

Wickham, H., Çetinkaya-Rundel, M., & Grolemund, G. (o. J.). *R for Data Science* (2. Aufl.). https://r4ds.hadley.nz/