Data Science für normale Anwender*innen

Prof. Dr. Nicolas Meseth

Aktualisiert am 16.01.2022

Contents

Übersicht

Zielgruppe

Dieses Buch gibt eine Einführung in wichtige Themen bei der Arbeit mit Daten. Wie der Titel schon ahnen lässt, adressiert dieses Buch *normale* Anwender*innen. Damit meine ich Personen, die in ihrem beruflichen (oder privaten) Alltag von Kenntnissen in der Analyse von Daten (über Excel hinaus) profitieren, diese aber nicht deren Haupttätigkeit ist.

In genau diese Zielgruppe fallen mehr als 95% der Studierenden, die an der Hochschule Osnabrück in jedem Semester an meinen Vorlesungen und Seminaren teilnehmen. Sie studieren einen Studiengang aus der Fachrichtung Agrar- oder Lebensmittelwirtschaft und haben in meinen Veranstaltungen die Möglichkeit, zusätzliche Kompetenzen zu digitalen Themen zu entwickeln. Sie arbeiten später aber nicht in IT-Berufen oder als Data Scientist. Sie sind aber dafür gerüstet, mit diesen Abteilungen (IT / Data Scientists) besser zusammenzuarbeiten und viele Dinge können sie auch in Eigenregie umsetzen. Dazu zählen beispielsweise Datenanalysen und -visualisierungen mit R.

Didaktisches Konzept

Das Buch ist in fortlaufend nummerierte Kapitel gegliedert. Man folgt meinem didaktischen Konzept, wenn man diese Reihenfolge einhält. Dieses Konzept beruht darauf, sich zuerst *hands-on* mit einem Thema auseinanderzusetzen, bevor die relevanen theoretischen Hintergründe eingeführt werden. Meiner Erfahrung nach sind theoretische Hintergründe einfacher zu verstehen, wenn das Subjekt der Betrachtung bereits in den Händen war. Der Kontext ist präsent und man weiß bei fachlichen Begriffen sofort die richtige Assoziation herzustellen.

Ein zweiter wichtiger Gedanke beim didaktischen Aufbau des Buches ist die Verwendung von Beispielen aus der Praxis. Dazu zählen insbesodere Datensätze und Fragen an diese Daten. Jedes Kapitel beinhaltet Beispiele und Übungen mit

Bezug zu Daten aus der Praxis. Die Anwendungsfälle hinter den Daten reichen von Onlineumfragen aus der Markforschung über Daten aus Laborexperimenten bis hin zu Datensätzen aus sozialen Medien wie Twitter. Alle Datensätze, die in diesem Buch verwendet werden, sind im Kapitel Datensätze gelistet.

Nicht immer ist es sinnvoll, ein neues Konzept anhand eines Datensatzes aus der Praxis einzuführen. In diesem Fall greife ich auf synthetische Datensätze zurück, um die Idee oder das Konzept möglichst plastisch darstellen zu können. Die Anwendung des eingeführten Konzepts auf Datensätze aus der Praxis erfolgt dann anschließend. Es sollte in diesem Buch kein Konzept eingeführt werden, das nicht in der Praxis relevant ist.

Das Buch kann selbstverständlich abseits des didaktischen Konzepts verwendet werden. Ich habe beim Verfassen darauf geachtet, das jedes Kapitel in sich geschlossen ist und einzeln gelesen und bearbeitet werden kann. Das gilt auch für die Übungen für die Anwendung mit R.

Erste Schritte mit R

Chapter 1

Die Arbeitsumgebung

TL;DR

- Für die Arbeit mit R benötigen wir eine Installation von R auf unserem Computer.
- R ist für alle Betriebssysteme verfügbar.
- Mit R bekommen wir auch eine einfache grafische Benutzeroberfläche (RGui) mitgeliefert. Diese bietet aber wenig Funktionen.
- Mit dem kostenlosen RStudio bekommen wir eine vollwertige integrierte Entwicklungsumgebung (IDE) für die Arbeit R.

1.1 R installieren

R ist eine Open-Source-Software und für alle gängigen Betriebssysteme verfügbar. Ladet euch zunächst die neueste Version von R für euer Betriebssystem herunter und installiert es anschließend:

- R für Windows
- R für macOS
- R für Linux

Neben der Sprache und dem Interpreter für R erhaltet ihr mit der Installation auch eine rudimentäre grafische Oberfläche mit dem Namen *RGui* (GUI = Graphical User Interface). Diese besteht aus einer einfachen Konsole, über die ihr R-Befehle eingeben und ausführen könnt.

Erweiterte Funktionen wie Autovervollständigung beim Schreiben von R-Code, ein integrierter Debugger für die Fehlersuche, eine Echtzeit-Vorschau für R-Markdown und viele andere Features mehr bietet dieses einfache Tool nicht.

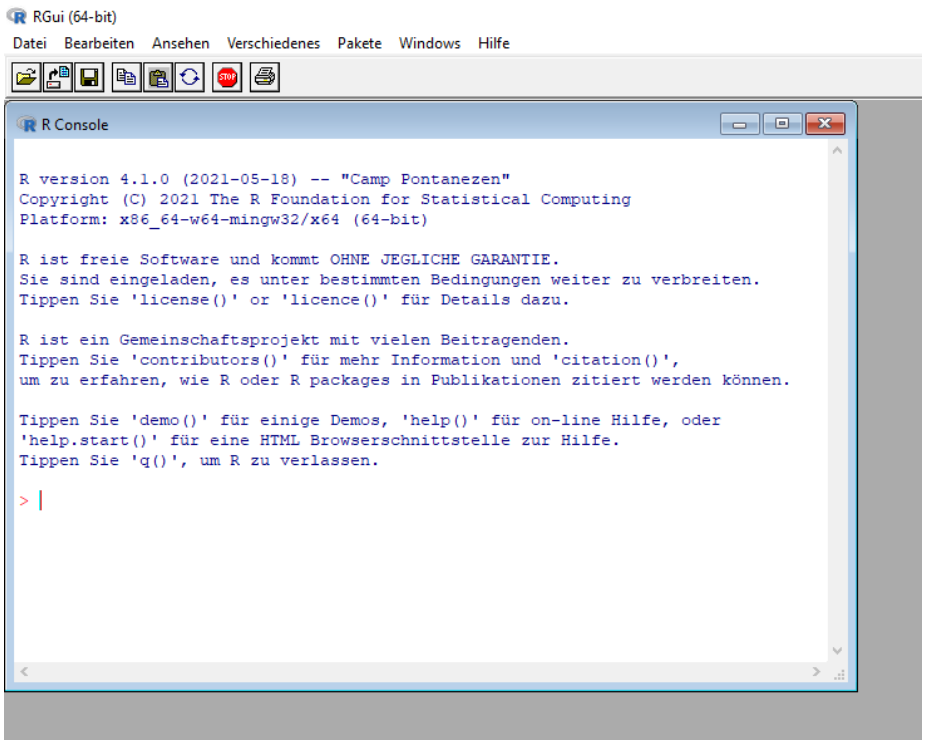


Figure 1.1: Die RGui bietet einen rudimentären Editor für R-Befehle.

Deshalb verwenden wir für die Arbeit mit R nicht die RGui, sondern das ebenfalls kostenlos nutzbare RStudio.

1.2 RStudio installieren

Das RStudio ist das Standardwerkzeug für die Arbeit mit R und bietet dafür viele nützliche Funktionen. Das RStudio ist ein sogenanntes **Integrated Development Environment (IDE)** für R. Einen schnellen Überblick über die grafische Benutzeroberfläche findet ihr in dem offiziellen RStudio Cheatsheet.

Klickt auf den Link unten und wählt RStudio für euer Betriebssystem aus. Installiert RStudio und öffnet es:

- RStudio herunterladen

Nach dem Öffnen seht ihr die Oberfläche des RStudio, die wie auf dem Screenshot unten aussieht:

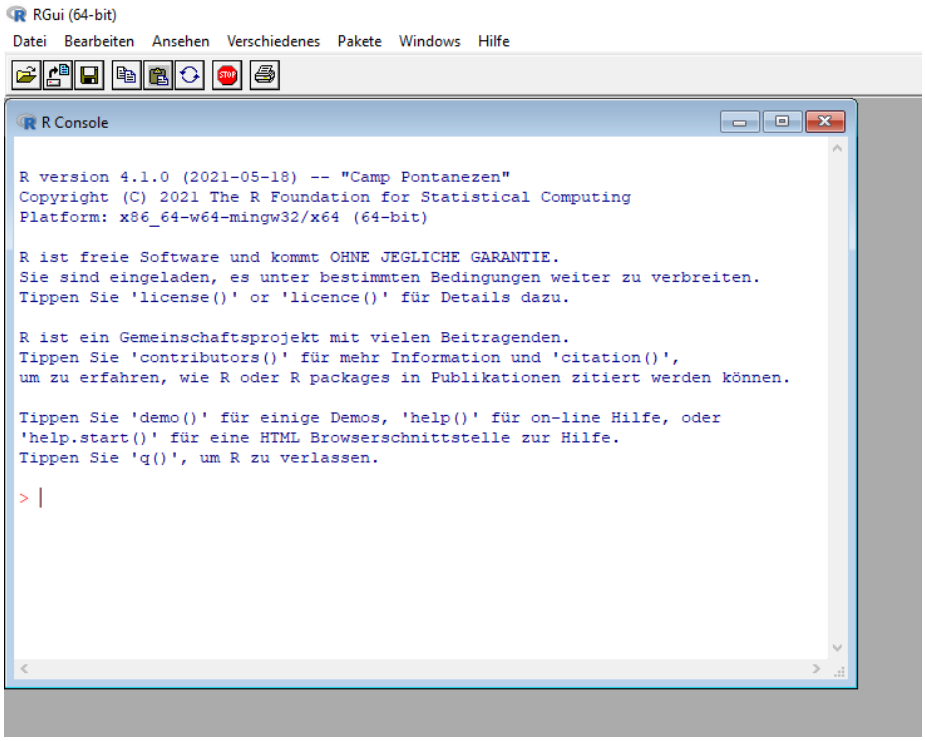


Figure 1.2: Das RStudio ist in vier Bereiche eingeteilt.

1.3 Überblick über RStudio

Das Werkzeug besteht in der Standardansicht aus vier Bereichen (s. Screenshot oben):

1.3.1 Der Skripteditor

Der wichtigste Bereich ist der Skripteditor. Hier schreiben wir unseren R-Code und speichern ihn in Dateien auf unserem Computer. Dabei unterstützt uns RStudio mit vielen nützlichen Funktionen.

Wir können in RStudio unterschiedliche Arten von Dateien verwenden, um unsere Skripte zu speichern. Die einfachste Art sind sogenannte **R-Skripte** mit der Dateiendung `.R`. Wenn wir nicht nur R-Code, sondern auch Visualisierungen und formatierten Text in einem Dokument verwenden und anzeigen wollen, bietet sich die Verwendung eines **R-Notebooks** an. Eine R-Notebook-Datei endet auf `.Rmd` (R-Markdown) und wir können neben R-Code auch Markdown verwenden. Über Markdown lernen wir zu einem späteren Zeitpunkt mehr.

1.3.2 Informationen zur aktuellen R-Umgebung

R erstellt für die Ausführung von Skripten eine sogenannte Session. In einer Session werden alle Objekte, wie die momentan verwendeten Daten, eigene Variablen oder Funktionen, im Arbeitsspeicher des lokalen Rechners gespeichert. In dem zweiten Fenster können im Tab *Environment* alle Objekte, die es in der aktuellen Session gibt, in der Übersicht und im Detail betrachtet werden. Der *History* Tab enthält eine Liste aller ausgeführten Befehle in der aktuellen Session. Daneben gibt es noch den *Connections* Tab für die Verbindung zu unterschiedlichen Datenquellen, wie Datenbanken oder Webservices, sowie den *Tutorial* Tab, in dem direkt in RStudio unterschiedliche Anleitungen angezeigt werden können.

{% hint style="info" %} Für die Anzeige von Tutorials direkt in RStudio muss das Paket `learnr` installiert werden. Ihr müsst der Installation einmalig zustimmen und das Paket installieren. {% endhint %}

1.3.3 Konsole und Terminal

Die Konsole ermöglicht, R-Befehle einzugeben und mit der Taste Enter auszuführen. Die Konsole in RStudio ist fast identisch zu der RGui. Für das schnelle Ausprobieren von Befehlen kann die Konsole nützlich sein. Für die strukturierte Arbeit mit den Daten sollten wir aber den Skripteditor verwenden, weil wir dort unsere Arbeit speichern und mit Kommentaren versehen können.

1.3.4 Dateien, Vorschau und Plots

In diesem Bereich werden Vorschauen unterschiedlicher Artefakte angezeigt. Dazu gehören gerenderte R-Notebooks, Markdown, aber auch Visualisierungen. In diesem Bereich gibt es auch einen rudimentären Datei-Explorer, um Dateien wie Skripte oder Datendateien zu finden und von dort zu öffnen.

Chapter 2

Einen Datensatz laden

Nachdem die Arbeitsumgebung eingerichtet ist, können wir mit den ersten Schritten in R beginnen. Wir steigen direkt ein und lernen, wie wir mit Daten in R arbeiten.

TL;DR

- R bietet verschiedene Möglichkeiten, um mit Daten zu arbeiten.
- Für strukturierte Daten in Tabellenform (Zeilen und Spalten) verwenden wir in R den Dataframe.
- Eine modernere Version des klassischen Dataframe ist das Tibble aus dem Tidyverse.

2.1 Tabellarische Daten in R

2.1.1 Der Dataframe

In R gibt es verschiedene Strukturen für die Speicherung von Daten. Darunter sind beispielsweise Vektoren, Listen oder Matritzen. Um tabellarische Daten abzubilden, die in Spalten und Zeilen organisiert sind, verwenden wir in R den sogenannten **Dataframe**.

2.1.2 Tibbles

Der Tibble ist eine Weiterentwicklung des klassischen Dataframe in R. Tibbles werden im Tidyverse standardmäßig verwendet und mit dem Paket `tibble` eingeführt.

2.2 Das CSV-Format

In den meisten Fällen erzeugen wir einen Dataframe oder Tibble, indem wir einen Datensatz aus einer Datenquelle laden. Häufig ist die Quelle eine einfache Texdatei, in der Daten zeilenweise gespeichert sind und jede Zeile aus mehreren einzelnen Werten besteht, die durch ein bestimmtes **Trennzeichen** voneinander getrennt sind. Wenn die Werte mit einem Komma voneinander getrennt sind, nennt man das Format **CSV**. CSV steht für **Comma Separated Values**, was auf Deutsch so viel wie *Durch Kommata getrennte Werte* bedeutet.

Unten seht ihr ein Beispiel für Daten im CSV-Format.

```
id,firstname,lastname
1,Boris,Becker
2,Steffi,Graf
3,Rafael, Nadal
```

Die erste Zeile nennen wir auch Kopfzeile oder *Header*. Sie beinhaltet die Spaltennamen. Jede nachfolgende Zeile stellt einen Datensatz oder *Record* dar. Dabei müssen die Werte in jeder Zeile mit Komma getrennt in der selben Reihenfolge wie im Header aufgeführt werden. Nur so ist eindeutig erkennbar, welcher Wert zu welcher Spalte gehört.

Manchmal fehlen Werte für eine bestimmte Spalte. In diesem Fall werden einfach zwei (oder mehrere) Kommata hintereinander gesetzt.

```
id,firstname,lastname,height,weight,gender
1,Boris,Becker,,,male
2,Steffi,Graf,175,64,female
3,Rafael,Nadal,185,,male
```

2.3 CSV-Daten laden

```
library(readr)
food_production <- read_csv("food_production.csv")
```

Chapter 3

Einen Datensatz erkunden

TL;DR

- In RStudio können wir mit `view` den gesamten Dataframe oder Tibble im Skripteditor anzeigen.
- Die Spaltennamen eines Tibble erhalten wir mit `colnames`.
- Mit `ncol` und `nrow` erhalten wir die Anzahl Spalten und Zeilen eines Tibbles. Das Gleiche bekommen wir in einem Schwung mit der Funktion `dim`.
- Einen schnellen Überblick über eine nominalskalierte Spalte und deren Werte bekommen wir mit `distinct` (eindeutige Werte) oder `count` (zusätzlich die Häufigkeiten).
- Für intervallskalierte Werte können wir besser mit `min` und `max` die Spannweite ermitteln oder mit einem Histogramm die Verteilung der Werte visualisieren.
- Mit `is.na` können wir prüfen, ob ein Wert NA ist (nicht vorhanden) und so auch die Anzahl fehlender Werte ermitteln.
- Mit der `summary` Funktion erzeugen wir eine Übersicht wichtiger statistischer Kennzahlen für eine Spalte.
- Mit `head` und `tail` erhalten wir die ersten oder letzten 10 Zeilen eines Tibble. Mit dem einzigen Parameter der Funktion können wir auch eine andere Anzahl zurückgeben.

3.1 Alle Daten anzeigen

Oft ist es hilfreich, einen schnellen Blick in einen Datensatz zu werfen, um beispielsweise die Werte einer Spalte zu überprüfen. Dafür können wir in RStudio die Funktion `view()` verwenden. Der Funktion übergeben wir den Tibble

`pokemon`:

```
view(pokemon)
```

Mit Strg + Enter oder über den Button „Run“ in der rechten oberen Ecke des Skriptfensters führen wir die Zeile aus. Es öffnet sich ein neuer Tab mit einer tabellarischen Ansicht der Daten.

first_steps.R × food_production ×

Filter

	food_product	land_use_change	animal_feed	farm	processing	transport	packging	retail	total_emissions
1	Wheat & Rye (Bread)	0.1	0.0	0.8	0.2	0.1	0.1	0.1	1.4
2	Maize (Meal)	0.3	0.0	0.5	0.1	0.1	0.1	0.0	1.1
3	Barley (Beer)	0.0	0.0	0.2	0.1	0.0	0.5	0.3	1.1
4	Oatmeal	0.0	0.0	1.4	0.0	0.1	0.1	0.0	1.6
5	Rice	0.0	0.0	3.6	0.1	0.1	0.1	0.1	4.0
6	Potatoes	0.0	0.0	0.2	0.0	0.1	0.0	0.0	0.3
7	Cassava	0.6	0.0	0.2	0.0	0.1	0.0	0.0	0.9
8	Cane Sugar	1.2	0.0	0.5	0.0	0.8	0.1	0.0	2.6
9	Beet Sugar	0.0	0.0	0.5	0.2	0.6	0.1	0.0	1.4
10	Other Pulses	0.0	0.0	1.1	0.0	0.1	0.4	0.0	1.6

Showing 1 to 10 of 43 entries, 23 total columns

Figure 3.1: Die Anzeige eines Tibbles in RStudio mit ‘view‘.

3.2 Spaltenamen ermitteln

Um mit einem Datensatz arbeiten zu können, benötigen wir eine Übersicht über seine Spalten. Diese liefert und die Funktion `colnames`:

```
pokemon %>%
  colnames()
```

```
##  [1] "abilities"         "against_bug"       "against_dark"
##  [4] "against_dragon"    "against_electric"  "against_fairy"
##  [7] "against_fight"     "against_fire"      "against_flying"
## [10] "against_ghost"     "against_grass"     "against_ground"
## [13] "against_ice"       "against_normal"    "against_poison"
## [16] "against_psychic"   "against_rock"      "against_steel"
## [19] "against_water"     "attack"            "base_egg_steps"
## [22] "base_happiness"    "base_total"        "capture_rate"
## [25] "classfication"     "defense"           "experience_growth"
## [28] "height_m"          "hp"                "japanese_name"
## [31] "name"              "percentage_male"   "pokedex_number"
```

```
## [34] "sp_attack"         "sp_defense"        "speed"
## [37] "type1"             "type2"             "weight_kg"
## [40] "generation"        "is_legendary"
```

Auf diese Information greifen wir zurück, wenn wir später Analysen mit dem Datensatz durchführen oder bestimmte Spalten genauer untersuchen wollen.

3.3 Anzahl Spalten und Zeilen ermitteln

Einfache Informationen wie die Anzahl Spalten und Zeilen ermitteln wir mit den Funktionen `ncol`, `nrow` oder `dim`.

```
pokemon %>%
  ncol()
```

```
## [1] 41
```

```
pokemon %>%
  nrow()
```

```
## [1] 801
```

Wenn wir beide Werte in einer Ausgabe haben wollen können wir `dim` verwenden:

```
pokemon %>%
  dim()
```

```
## [1] 801  41
```

`dim` gibt einen Vektor mit zwei Werten zurück. Der erste Wert steht für die Anzahl Zeilen, der zweite für die Anzahl Spalten. Wir können mit der Angabe der Position in eckigen Klammern auf jeden Wert einzeln zugreifen:

```
dimensions <- pokemon %>% dim()

# Anzahl Zeilen
dimensions[1]
```

```
## [1] 801
```

Und auf die Spalten:

```
# Anzahl Spalten
dimensions[2]
```

```
## [1] 41
```

3.4 Wertebereich von Spalten bestimmen

3.4.1 Nominalskalierte Spalten

3.4.1.1 Eindeutige Werte mit `distinct`

Für nominalskalierte Spalten kann es interessant sein, alle Werte und eventuell deren Häufigkeit anzuzeigen. Die eindeutigen Werte bekommen wir mit der `distinct` Funktion:

```
pokemon %>%
  distinct(type2)
```

```
## # A tibble: 19 x 1
##    type2
##    <chr>
##  1 poison
##  2 <NA>
##  3 flying
##  4 dark
##  5 electric
...
```

3.4.1.2 Eindeutige Werte und deren Häufigkeiten

Wenn wir zusätzlich noch die Häufigkeiten ermitteln wollen, können wir die Funktion `count` verwenden und als Parameter den Spaltennamen übergeben:

```
pokemon %>%
  count(type1)
```

```
## # A tibble: 18 x 2
##    type1        n
##    <chr>    <int>
##  1 bug         72
##  2 dark        29
```

```
##  3 dragon      27
##  4 electric    39
##  5 fairy       18
...
```

Meist es nützlich, direkt nach Häufigkeiten zu sortieren:

```
pokemon %>%
  count(type1, sort = TRUE)
```

```
## # A tibble: 18 x 2
##    type1        n
##    <chr>    <int>
##  1 water      114
##  2 normal     105
##  3 grass       78
##  4 bug         72
##  5 psychic     53
...
```

3.4.2 Intervallskalierte Spalten

Für intervallskalierte Spalten, also Spalten mit Zahlenwerten, eignen sich die Funktionen `distinct` und `count` nur bedingt. Oft sind es zu viele unterschiedliche Werte und das reine Zählen bringt nicht viel Aufschluss. Wenn wir distinct auf die Spalte `weight_kg` anwenden, erhalten wir zum Beispiel 422 unterschiedliche Werte:

```
pokemon %>%
  distinct(weight_kg)
```

```
## # A tibble: 422 x 1
##    weight_kg
##        <dbl>
##  1       6.9
##  2      13
##  3     100
##  4       8.5
##  5      19
##  6      90.5
##  7       9
##  8      22.5
##  9      85.5
## 10       2.9
## # ... with 412 more rows
```

In diesem Fall ist es sinnvoller, den kleinsten und größten Wert zu kennen, um die **Spannweite** der Spalte zu ermitteln. Interessant wäre auch die Verteilung der Werte, etwa in Form eines Histogramm.

3.4.2.1 Größter und kleinster Wert

Den größten oder kleinsten Wert können wir mit `max` und `min` ermitteln:

```
pokemon %>%
  select(weight_kg) %>%
  max(na.rm = TRUE)
```

```
## [1] 999.9
```

```
pokemon %>%
  select(weight_kg) %>%
  min(na.rm = TRUE)
```

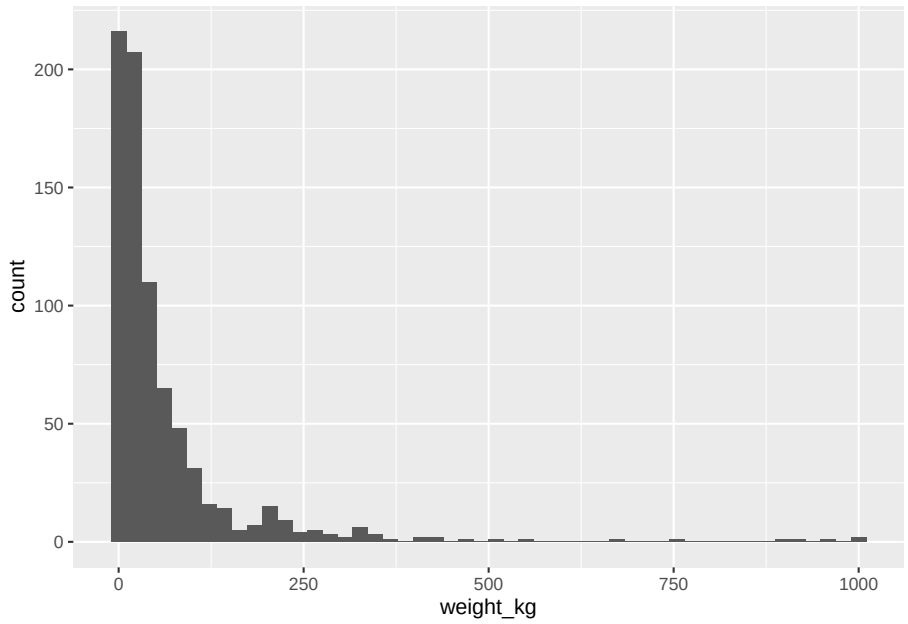
```
## [1] 0.1
```

Wir wissen also, dass die Werte sich zwischen 0,1 und 999,9 bewegen. Wie aber sind die Werte dazwischen verteilt?

3.4.2.2 Verteilung der Werte als Histogramm

Um die Verteilung der Werte innerhalb der Spalte `weight_kg` sinnvoll zu ermitteln müssen wir an dieser Stelle einen Exkurs in die Visualisierung von Daten machen:

```
pokemon %>%
  select(weight_kg) %>%
  filter(!is.na(weight_kg)) %>%
  ggplot() +
  aes(x = weight_kg) +
  geom_histogram(bins = 50)
```



Das Historgramm zeigt uns, dass der größte Teil der Daten im kleineren Bereich liegt und nur wenige Ausnahmen große Werte jenseits der 300 aufweisen.

3.4.2.3 Anzahl fehlender Werte

Die Anzahl fehlender Werte, die in R als `NA` dargestellt werden, ist ein wichtiges Merkmal für die Qualität der Daten einer Spalte. Um diese Zahl zu ermitteln filtern wir die Daten mit der Funktion `is.na` und zählen anschließend die übrig gebliebenen Werte:

```
pokemon %>%
  select(weight_kg) %>%
  filter(is.na(weight_kg)) %>%
  count()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1    20
```

Insgesamt haben somit 20 Pokémon keine Gewichtsangabe.

3.4.2.4 Zusammenfassung einer Spalte ausgeben

Um gleich mehrer statistische Größen zu berechnen und auszugeben können wir die Funktion `summary` verwenden:

```
pokemon %>%
  select(weight_kg) %>%
  summary()
```

```
##    weight_kg
##  Min.   :  0.10
##  1st Qu.:  9.00
##  Median : 27.30
##  Mean   : 61.38
##  3rd Qu.: 64.80
##  Max.   :999.90
##  NA's   :20
```

3.5 Erste oder letzte Zeilen anzeigen

Die Funktion `head` gibt die ersten 10 Zeilen eines Tibbles zurück:

```
pokemon %>%
  head()
```

```
## # A tibble: 6 x 41
##   abilities             against_bug against_dark against_dragon against_electric
##   <chr>                       <dbl>        <dbl>          <dbl>            <dbl>
## 1 ['Overgrow', 'Chloro~        1              1              1              0.5
## 2 ['Overgrow', 'Chloro~        1              1              1              0.5
## 3 ['Overgrow', 'Chloro~        1              1              1              0.5
## 4 ['Blaze', 'Solar Pow~        0.5            1              1              1
## 5 ['Blaze', 'Solar Pow~        0.5            1              1              1
## 6 ['Blaze', 'Solar Pow~        0.25           1              1              2
## # ... with 36 more variables: against_fairy <dbl>, against_fight <dbl>,
## #   against_fire <dbl>, against_flying <dbl>, against_ghost <dbl>,
## #   against_grass <dbl>, against_ground <dbl>, against_ice <dbl>,
## #   against_normal <dbl>, against_poison <dbl>, against_psychic <dbl>,
## #   against_rock <dbl>, against_steel <dbl>, against_water <dbl>, attack <dbl>,
## #   base_egg_steps <dbl>, base_happiness <dbl>, base_total <dbl>,
## #   capture_rate <chr>, classfication <chr>, defense <dbl>, ...
```

Über den Parameter der Funktion können wir die Anzahl der Zeilen verändern, um etwa nur die erste Zeile zu erhalten:

```
pokemon %>%
  head(1)
```

Die Funktion `tail` ist das Pendant dazu und gibt die 10 letzten Zeilen aus. Genau wie bei der `head()` Funktion können wir den Parameter für die Angabe der konkreten Anzahl Zeilen verwenden:

```
pokemon %>%
  tail(1)
```

Chapter 4

Der Werkzeugkasten

Bei der Arbeit mit Daten ist es wichtig zu wissen, welches Werkzeug (hier: R-Paket oder Funktion) wir für welche Aufgabe verwenden. Genauso wie in einer Werkstatt, in der es eine Vielzahl an Werkzeugen gibt, die für unterschiedliche Zwecke geeignet sind.

Figure 4.1: Verschiedene Werkzeuge für unterschiedliche Aufgaben.

Die Tabelle unten listet wichtige Werkzeuge auf, die wir im weiteren Verlauf dieses Buches kennenlernen werden.

Paket	Funktion	Aufgabe
`tibble`	`as.tibble`	Erstellt einen modernen Dataframe für tabellarische Daten.
`readr`	`read_csv`	Lesen von tabellarischen Datenformaten wie CSV-Dateien.
`dplyr`	`select`	Auswählen von Spalten (Variablen) eines Datensatzes.
`dplyr`	`filter`	Filtern von Daten auf Basis fast beliebiger Ausdrücke.
`dplyr`	`mutate`	Hinzufügen neuer Spalten (Variablen).
`dplyr`	`recode`	Spaltenwerte neu kodieren.
`dplyr`	`arrange`	Die Reihenfolge von Zeilen verändern.
`dplyr`	`group_by`	Gruppieren von Daten.
`dplyr`	`summarise`	Zusammenfassen von Daten.
`ggplot2`	`ggplot`, `aes`, `geom_line`, `geom_bar`, `geom_col` `geom_point` u.v.m.	Visualisieren von Daten.

4.1 Das Paket `tibble`

Das Paket `tibble` führt das moderne Pendant zum klassischen Dataframe in R ein.

- Zur offiziellen Webseite des `tibble` Pakets

4.2 Das Paket `readr`

Das Paket `readr` beinhaltet Funktionen für das Laden von Daten aus strukturierten Datenformaten wie CSV-Dateien. Alle Funktionen zum Datenimport aus readr erzeugen automatisch einen `tibble`.

- Zur offiziellen Webseite des `readr` Pakets

4.3 Das Paket `dplyr`

Das Paket `dplyr` hat einen etwas merkwürdigen Namen. Er setzt sich aus dem Buchstaben „d“ und dem abgekürzten Wort „plyr“ zusammen. Das „d“ steht für Dataframe, während „plyr“ für den englischen Begriff „plier“ steht, was auf Deutsch „Zange“ bedeutet. Passend dazu bildet das offizielle Symbol des Pakets mehrere Zangen ab.

- Zur offiziellen Dokumentation des `dplyr` Pakets

`dplyr` liefert uns eine Vielzahl wichtiger Funktionen für die Manipulation von Daten, die in Form eines Tibble vorliegen. Eine Übersicht der Funktionen findet ihr in dem Cheat Sheet Data Transformation wit dplyr.

4.4 Das Paket `ggplot2`

`ggplot2` ist eines der umfassendsten Pakete für die professionelle Visualisierung von Daten mit R:

- Zur offiziellemn Dokumentation des `ggplot2` Pakets

Das Cheat Sheet Data Visualization with ggplot2 beinhaltet alle wichtigen Funktionen im Überblick.

Explorative Datenanalyse mit R

Chapter 5

Der explorative Analyseprozess

Dieser Abschnitt führt euch in die Grundlagen der **explorativen Datenanalyse** mit R ein. In der explorativen Datenanalyse versuchen wir einen unbekannten Datensatz mit geeigneten Verfahren kennenzulernen und schnell Muster in den Daten zu erkennen. Auf Basis dieser Muster formulieren wir **Hypothesen**. Diese Hypothesen können anschließend mit statistischen Modellen aus dem Bereich der schließenden Statistik auf ihre Gültigkeit überprüft werden. Die statistische Überprüfung ist jedoch nicht Teil der explorativen Datenanalyse nach dem Verständnis dieses Buches.

Eine ausgezeichnete Einführung in die explorative Datenanalyse mit R gibt auch das Buch R for Data Science von Wickham and Grolemund [2016]. Das Buch ist online frei zugänglich.

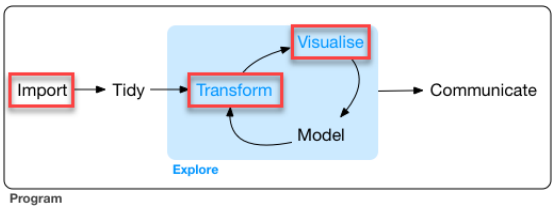


Figure 5.1: Der Datenanalyseprozess.

Wickham and Grolemund [2016] definieren den Datenanalyseprozess durch eine Abfolge bestimmter Schritte, wie in der Abbildung **??** gezeigt. In diesem Abschnitt stehen die rot markierten Schritte im Fokus.

5.1 Daten laden

Jeder Analyseprozess beginnt mit dem Laden eines Datensatzes. Dabei gibt es verschiedene Datenquellen, die in Betracht gezogen werden müssen. Ein häufig verwendetes Format sind Komma-separierte Werte (**C**omma **S**eparated **V**alues = CSV) in einfachen Textdateien. Dieses Format steht auch hier im Vordergrund.

Gemäß der Abbildung aus Wickham and Grolemund [2016] folgt auf das Laden der Daten der Arbeitsschritt „Tidy“. Dieser ist dann notwendig, wenn die Daten nicht in der typischen Form bestehend aus Spalten und Zeilen vorliegen. Leider ist das in der Praxis oft der Fall. In diesem Buch wird aber zunächst davon ausgegangen, dass die Daten das entsprechende Format aufweisen. Die interessierte Leserin verweise ich auf das Kapitel 12 im Buch „R for Data Science“.

5.2 Daten transformieren

Das Ziel der explorativen Datenanalyse ist die Visualisierung der Daten mit geeigneten Diagrammen, um interessante Muster sichtbar werden zu lassen. Visualisierungen benötigen häufig nur einen Teil der Daten (wenige Spalten oder bestimmte Zeilen). Auch müssen wir oft neue Spalten berechnen oder bestehende Daten aggregieren, bevor wir sie visualisieren können. Alle diese Aufgaben können wir unter dem Bereich der **Datentransformation** zusammenfassen.

Für diese Aufgaben bietet R mit dem Paket `dplyr` mächtige Funktionen. Insbesondere lernen wir in dem Abschnitt:

- Wie wir bestimmte Spalten auswählen können (dplyr-Verb: `select`).
- Wie wir Zeilen fast beliebig filtern können (dplyr-Verb: `filter`).
- Wie wir neue, berechnete Spalten hinzufügen können (dplyr-Verb: `mutate`).
- Wie wir Zeilen sortieren können (dplyr-Verb: `arrange`).
- Wie wir Zeilen zusammenfassen und gruppieren können (dplyr-Verben: `summarize` und `group_by`)

5.3 Daten visualisieren

Das wichtigste Werkzeug in der explorativen Datenanalyse ist die **Visualisierung von Daten**. In R steht uns dafür mit `ggplot2` ein leistungsfähiges Instrument zur Verfügung. Wir lernen für bestimmte Anwendungsfälle die richtigen Visualisierungen zu identifizieren und mit `ggplot2` umzusetzen.

5.4 Literatur

5.4.1 Bücher

Wickham, Hadley, and Garrett Grolemund. R for Data Science: Import, Tidy, Transform, Visualize, and Model Data. First edition, O'Reilly, 2016. Online verfügbar: https://r4ds.had.co.nz/

Wickham, Hadley. ggplot2. Springer Science+Business Media, LLC, 2016. Online verfügbar: https://ggplot2-book.org/

Kabacoff, Robert. R in Action: Data Analysis and Graphics with R. Second edition, Manning, 2015.

Sauer, Sebastian. Moderne Datenanalyse mit R: Daten einlesen, aufbereiten, visualisieren, modellieren und kommunizieren. Springer Gabler, 2019. Online verfügbar: https://link.springer.com/book/10.1007/978-3-658-21587-3

5.4.2 Online-Dokumentationen

- Die offizielle Dokumentation der Tidyverse-Bibliotheken
- Ein Tutorial für die Einführung in R mit RStudio
- Ein Tutorial zu den Grundlagen der Datenmanipulation mit R, `tidyr` und `dplyr`
- Weiterführende Anleitungen zur Datenmanipulation mit `dplyr` (Efficient Data Manipulation)
- Weiterführende Anleitungen zur Datenmanipulation mit `dplyr` (Advanced Data Manipulation)

Chapter 6

Spalten auswählen mit `select`

TL;DR

- Mit dem `select` Befehl können wir aus allen Spalten einer Tabelle eine Untermenge auswählen.
- Neben einer Liste von gewünschten Spaltennamen können wir die Untermenge auch über den Ausschluss von nicht gewollten Spalten eingrenzen. Dafür verwenden wir das Zeichen `-` vor dem Spaltennamen.
- Mit der Funktion `last_col()` können wir die letzte oder die n-letzte Spalte auswählen
- Mit dem Semikolon können wir Spaltenbereiche auswählen und so beispielsweise alle Spalten von der 5. bis zur vorletzten selektieren (`select(5 : last_col(2))`).
- Wir können Spalten anhand ihres Datentyps selektieren.
- Wir können Spalten über die Zugehörigkeit zu einer Liste selektieren.

6.1 Beispieldaten: Pokémon

Es ist ratsam die verfügbaren Spalten eines Datensatzes zu kennen, bevor wir uns mit einigen Beispielen beschäftigen, wie wir Bestimmte davon auswählen können. Für dieses Kapitel verwenden wir den Pokémon-Datensatz, der auf Kaggle.com bereitgestellt wird.

```
pokemon <- read_csv("datasets/pokemon.csv")
```

Nach dem Laden des Datensatzes lassen wir uns mit `colnames()` die Spalten anzeigen:

```
pokemon %>% colnames()
```

```
##  [1] "abilities"         "against_bug"       "against_dark"
##  [4] "against_dragon"    "against_electric"  "against_fairy"
##  [7] "against_fight"     "against_fire"      "against_flying"
## [10] "against_ghost"     "against_grass"     "against_ground"
## [13] "against_ice"       "against_normal"    "against_poison"
## [16] "against_psychic"   "against_rock"      "against_steel"
## [19] "against_water"     "attack"            "base_egg_steps"
## [22] "base_happiness"    "base_total"        "capture_rate"
## [25] "classfication"     "defense"           "experience_growth"
## [28] "height_m"          "hp"                "japanese_name"
## [31] "name"              "percentage_male"   "pokedex_number"
## [34] "sp_attack"         "sp_defense"        "speed"
## [37] "type1"             "type2"             "weight_kg"
## [40] "generation"        "is_legendary"
```

Das sind mehr als 40 Spalten. Wie viele genau?

```
pokemon %>% dim()
```

```
## [1] 801  41
```

Oder mit `ncol()`:

```
pokemon %>% ncol()
```

```
## [1] 41
```

Es sind also 41 Spalten. Genug um die wichtigsten Funktionen für das Auswählen von Spalten zu zeigen.

6.2 Bestimmte Spalten nach Namen auswählen

Die Funktion `select` verwenden wir für das Auswählen einer Untermenge an Spalten. Wir können im einfachsten Fall einfach die gewünschten Spalten mit Kommata getrennt auflisten:

```
pokemon %>%
  select(name, type1)
```

```
## # A tibble: 801 x 2
##    name       type1
##    <chr>      <chr>
##  1 Bulbasaur  grass
##  2 Ivysaur    grass
##  3 Venusaur   grass
##  4 Charmander fire
##  5 Charmeleon fire
##  6 Charizard  fire
##  7 Squirtle   water
##  8 Wartortle  water
##  9 Blastoise  water
## 10 Caterpie   bug
## # ... with 791 more rows
```

Dabei können wir beliebig viele Spalten in die Liste aufnehmen:

```
pokemon %>%
  select(name, type1, type2, is_legendary)
```

```
## # A tibble: 801 x 4
##    name       type1 type2  is_legendary
##    <chr>      <chr> <chr>         <dbl>
##  1 Bulbasaur  grass poison            0
##  2 Ivysaur    grass poison            0
##  3 Venusaur   grass poison            0
##  4 Charmander fire  <NA>              0
##  5 Charmeleon fire  <NA>              0
##  6 Charizard  fire  flying            0
##  7 Squirtle   water <NA>              0
##  8 Wartortle  water <NA>              0
##  9 Blastoise  water <NA>              0
## 10 Caterpie   bug   <NA>              0
## # ... with 791 more rows
```

6.3 Spalten aus der Auswahl ausschließen

Wollen wir fast alle verfügbaren Spalten auswählen, so ist es effizienter, die Spalten zu nennen, die wir ausschließen wollen. Das können wir mit dem Minus-Symbol erreichen:

```
pokemon %>%
  select(-is_legendary)
```

6.4 Die letzte oder n-letzten Spalten auswählen

```
pokemon %>%
  select(last_col())
```

```
## # A tibble: 801 x 1
##    is_legendary
##           <dbl>
##  1            0
##  2            0
##  3            0
##  4            0
##  5            0
##  6            0
##  7            0
##  8            0
##  9            0
## 10            0
## # ... with 791 more rows
```

Chapter 7

Zeilen filtern mit `filter`

TL;DR

- Mit dem `filter` Befehl können wir eine Untermenge der gesamten Zeilen eines Datensatzes auswählen.
- Alphanumersiche nominalskalierte Spalten können wir mit dem == Operator auf einen bestimmten Wert filtern.

7.1 Filtern von alphanumerischen Werten (Zeichenketten)

7.1.1 Filtern auf Gleichheit

```
pokemon %>%
  select(name, type1) %>%
  filter(type1 == "electric")
```

```
## # A tibble: 39 x 2
##    name       type1
##    <chr>      <chr>
##  1 Pikachu    electric
##  2 Raichu     electric
##  3 Magnemite  electric
##  4 Magneton   electric
##  5 Voltorb    electric
...
```

7.1.2 Filtern über eine Liste von Werten

```
pokemon %>%
  select(name, type1) %>%
  filter(type1 %in% c("electric", "poison", "fire"))
```

```
## # A tibble: 123 x 2
##    name       type1
##    <chr>      <chr>
##  1 Charmander fire
##  2 Charmeleon fire
##  3 Charizard  fire
##  4 Ekans      poison
##  5 Arbok      poison
##  6 Pikachu    electric
...
```

Wenn wir die Liste im mehreren Operationen wiederverwenden wollen, können wir sie auch einmal auf einer Variable speichern und in der `filter` Funktion darauf zugreifen:

```
search_types <- c("electric", "poison", "fire")

pokemon %>%
  select(name, type1) %>%
  filter(type1 %in% search_types)
```

```
## # A tibble: 123 x 2
##    name       type1
##    <chr>      <chr>
##  1 Charmander fire
##  2 Charmeleon fire
##  3 Charizard  fire
##  4 Ekans      poison
##  5 Arbok      poison
...
```

7.1.3 Suchen nach dem Vorkommen eines Wertes in einem String

Um einen String auf das Vorhandensein eines bestimmten Wertes zu durchsuchen können wir auf die `str_detect` Funktion aus dem `stringr` Paket zurückgreifen. `stringr` ist ebenfalls im Tidyverse enthalten:

```
pokemon %>%
  select(name) %>%
  filter(str_detect(name, "chu"))
```

```
## # A tibble: 4 x 1
##   name
##   <chr>
## 1 Pikachu
## 2 Raichu
## 3 Pichu
## 4 Smoochum
```

Anhang

Datensätze

Table 7.1: Datensätze in diesem Buch

Name	**Download-Link**	**Kategorie**
Online-Umfrage zum Kaufverhalten von Orangenlimonade	Download	Marktforschung
Tweets ausgewählter deutscher Politiker:innen	Download	Soziale Medien
Pokémon	Download	TV

Bibliography

Hadley Wickham and Garrett Grolemund. *R for data science: import, tidy, transform, visualize, and model data*. O'Reilly, Sebastopol, CA, first edition edition, 2016. ISBN 9781491910399 9781491910368. OCLC: ocn968213225.