

Einführung in R

Fortbildung im Institut der deutschen Wirtschaft

2 Erste Schritte

Inhalte und Ziele der Sitzung

- Verständnis der Grundfunktionen von R und RStudio
- Erweiterte Einstellungen
- Erstellung eines Projekts
- Grundverständnis verschiedener Datentypen in R



Quelle: pickardpredictives.com

- Leseempfehlung: [An Introduction to R, Kapitel 1 & 2 \(Douglas et al., 2021\)](#)

Installation von R und RStudio

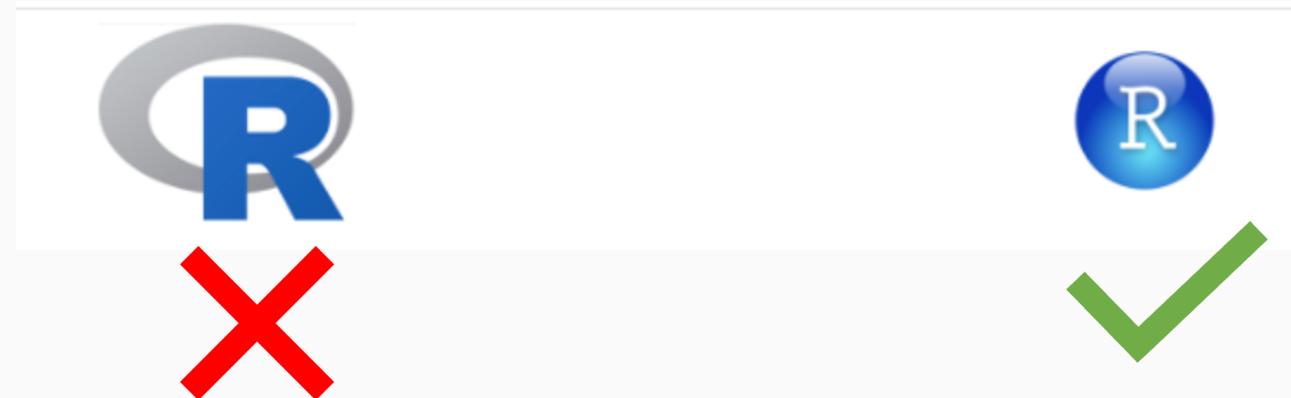
R und RStudio

- Im Kurs nutzen wir R und RStudio.
- Unsere Berechnungen, Abbildungen und so weiter werden alle mit R erstellt.
- RStudio bietet eine komfortable Benutzeroberfläche, um Ergebnisse anzuzeigen und mit R zu arbeiten.
- Die Arbeit mit R wird durch RStudio **deutlich benutzerfreundlicher** und dadurch nicht zuletzt auch einfacher.



Installation von R und RStudio

- Es muss **zuerst** R **und dann** RStudio installiert werden.
- R ist für Windows, Mac und Linux verfügbar unter: cloud.r-project.org.
- Rstudio steht unter rstudio.com zum Download bereit. Für Privatanwender bietet sich die kostenfreie RStudio Desktop Version an.
- Aus der Motor/Cockpit-Analogie folgt auch, dass wir nie die R-Anwendung direkt starten, sondern immer RStudio.



Benutzeroberfläche von RStudio

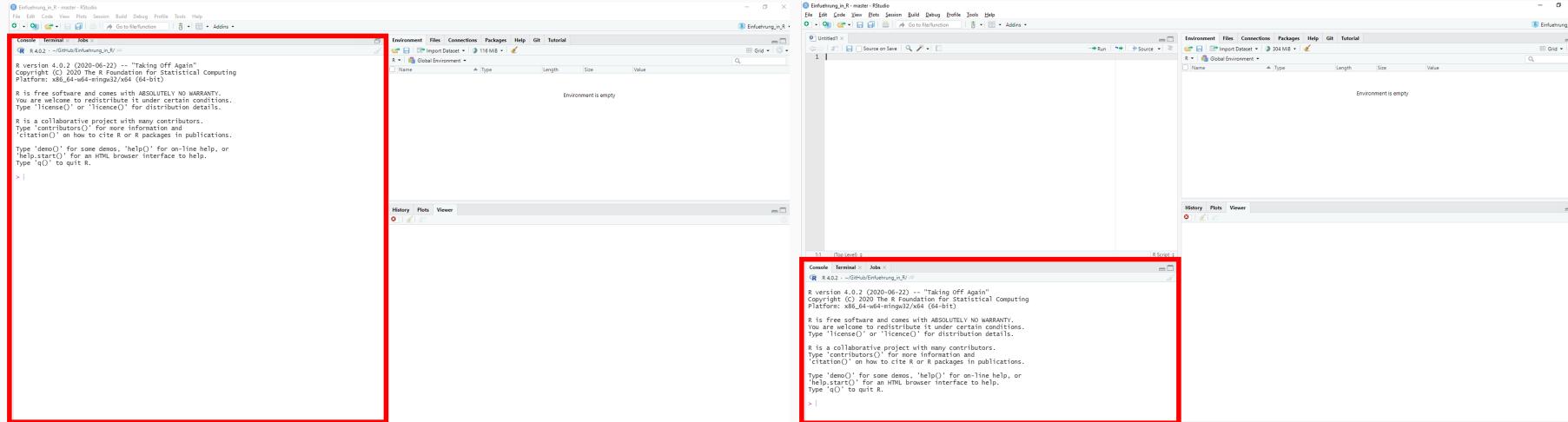
Orientierung auf der Benutzeroberfläche von RStudio

Console

Skripte (Source Oberfläche)

Environment und Co.

Plots und Viewer



- In der Console evaluiert R den Code.
- Man kann Code direkt in der Console schreiben und mit Enter ausführen.

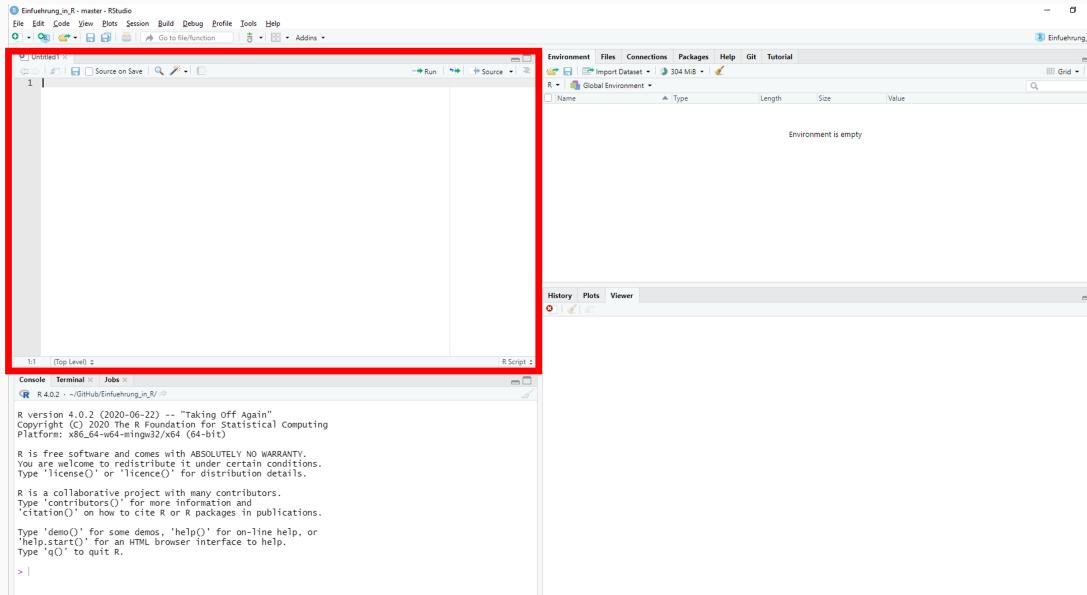
Orientierung auf der Benutzeroberfläche von RStudio

Console

Skripte (Source Oberfläche)

Environment und Co.

Plots und Viewer



- Zum Speichern von Code wird dieser in R-Skripten geschrieben.
- R-Skripte sind lediglich Textdateien mit einer **.R**-Dateiendung.
- Ein neues R-Skript wird über die Menüleiste (**File → New File → R Script**) oder über den Tastatur-Shortcut **Strg + Umsch. + N** geöffnet.
- Code wird über den Run-Button oder mit **Strg + Enter** ausgeführt.

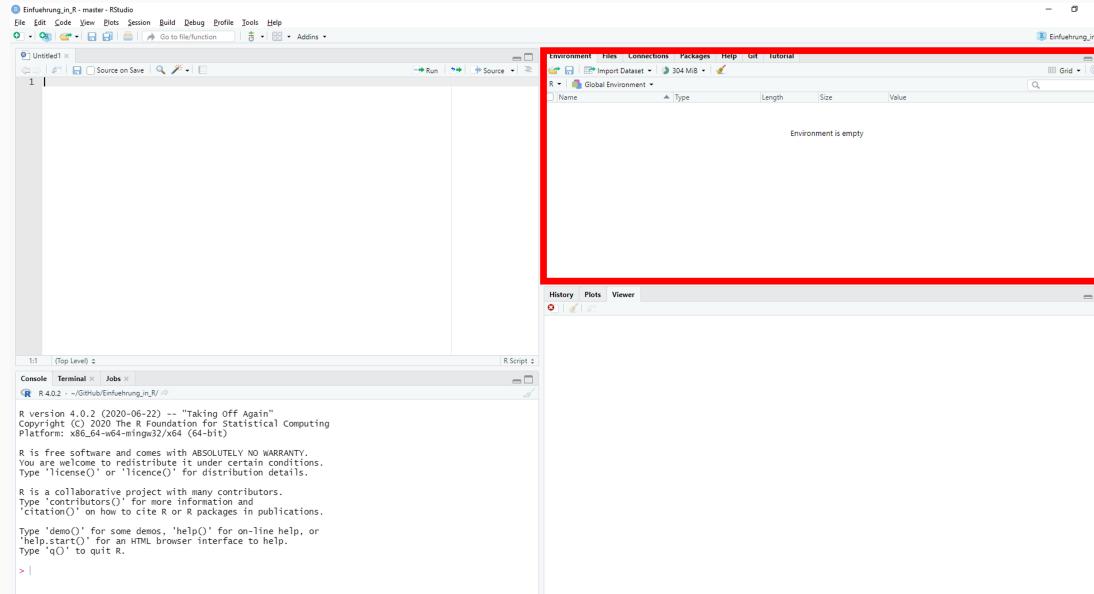
Orientierung auf der Benutzeroberfläche von RStudio

Console

Skripte (Source Oberfläche)

Environment und Co.

Plots und Viewer



- Das Environment zeigt uns gespeicherte Objekte.
- Die anderen Tabs zeigen weitere interessante Informationen, wie z.B. die Dateistruktur.

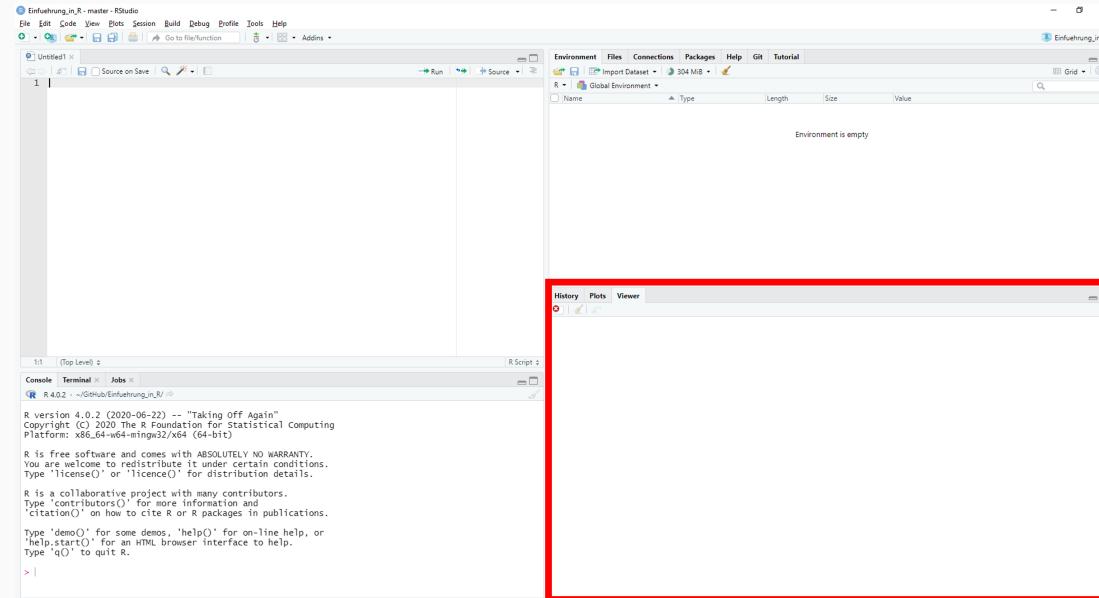
Orientierung auf der Benutzeroberfläche von RStudio

Console

Skripte (Source Oberfläche)

Environment und Co.

Plots und Viewer



- Im Plot-Fenster werden Abbildungen angezeigt.
- Der Viewer kommt z.B. bei interaktiven Abbildungen zum Einsatz.
- Das History-Fenster zeigt die Historie der ausgeführten Codezeilen.

Arbeiten mit Skripten und Projekten

Skripte und Projekte

- Skripte sind einfache Textdateien, die unseren R-Code und Kommentare enthalten. Sie haben die Dateiendung **.R**.
- Die Dokumentation der Datenanalyse in Skripten ist entscheidend für die Reproduzierbarkeit unserer Arbeit.
- Für die Organisation der Datenanalyse bietet RStudio die Möglichkeit, **Projekte** zu erstellen. Diese haben die Dateiendung **.Rproj**.
- Jedes Analyseprojekt sollte in einem eigenen Projekt durchgeführt werden.
- Wir werden gleich ein eigenes Projekt für unsere Übungseinheiten anlegen.

Mathematische Operatoren

Mathematische Operatoren

- Im Kern ist R ein (semi-)schicker Taschenrechner.
- Alle klassischen Rechenoperationen sind möglich:

+ Addition

- Subtraktion

* Multiplikation

/ Division

^ Potenzen

- Darüber hinaus stehen zahlreiche Hilfsfunktionen zur Verfügung, z.B.: `mean()`, `median()`, `min()`, `max()`, `sd()` für das arithmetische Mittel, den Median, das Minimum, das Maximum und die Standardabweichung.

Mathematische Operatoren

Ein einfaches Beispiel:

- Stellen wir uns vor, wir wollen aus dem BIP eines Landes und der Zahl der Einwohner das BIP pro Kopf berechnen.

| BIP | Einwohner |
|---------|-----------|
| 1000000 | 5000 |

- Teilen des BIPs durch die Einwohnerzahl ergibt das gewünschte Ergebnis. In R schreiben wir:

```
1000000 / 5000
```

```
#> [1] 200
```

Mathematische Operatoren

Ein einfaches Beispiel:

- Mathematische Operatoren können natürlich beliebig gemischt werden:

| Land | BIP | Einwohner |
|--------------|---------|-----------|
| FantasiAland | 1000000 | 5000 |
| FantasiBland | 2000000 | 6250 |

- Der bevölkerungsgewichtete Durchschnitt des BIPs pro Kopf der beiden Länder liegt zum Beispiel bei:

```
1000000 / 5000 * (5000 / (5000 + 6250)) + 2000000 / 6250 * (6250 / (5000 + 6250))
```

```
#> [1] 266.6667
```

R Objekte

- Objekte lassen sich als gespeicherte Informationen interpretieren. Dies können einfache Zahlen sein, aber auch Abbildungen oder Tabellen.
- Sobald ein Objekt gespeichert wurde, können wir es jederzeit wieder aufrufen.

Everything in R is an **object**. ~ John M. Chambers, Entwickler von R

- Objekte werden mit dem *assignment operator* einem Namen zugewiesen.
- Assignment operator:

<-

- Die Syntax ist dann also:

```
Name <- Objekt
```

R Objekte

Ein einfaches Beispiel

- Unser BIP pro Kopf Beispiel auf Basis von Objekten:

```
bip <- 1000000  
einwohner <- 5000
```

- Schreibt man lediglich die jeweiligen Objektnamen in eine Zeile und führt diese aus, so wird der Inhalt des zugehörigen Objekts angezeigt, in diesem Fall die beiden Zahlen:

```
bip
```

```
#> [1] 1e+06
```

```
einwohner
```

```
#> [1] 5000
```

R Objekte

Ein einfaches Beispiel

- Mit den Objekten kann dann gerechnet werden:

```
bip_pro_kopf <- bip / einwohner
```

- Das Ergebnis ist dann:

```
bip_pro_kopf
```

```
#> [1] 200
```

Noch mehr Operatoren

- Neben mathematischen Operatoren gibt es in R auch **logische Operatoren**.
- Logische Operatoren werden für logische Tests genutzt.
- Logische Operatoren können nur zwei Ergebnisse liefern: *wahr* oder *falsch*.
- Oder in der R-Syntax: `TRUE` oder `FALSE`.

Logische Operatoren

Logische Operatoren

- Zuerst erstellen wir einige weitere Objekte, um mit diesen logische Tests durchzuführen:

```
FantasiAland <- 1000000  
FantasiBland <- 1100000  
FantasiCland <- 1020000  
FantasiDland <- 3100000  
FantasiEland <- 1000000
```

- Logische Tests sind Vergleiche:

| Logischer Operator | Bedeutung |
|--------------------|-------------------------|
| == | ist gleich |
| != | ist nicht gleich |
| > | ist größer |
| < | ist kleiner |
| >= | ist größer oder gleich |
| <= | ist kleiner oder gleich |

Logische Operatoren

- `==` evaluiert, ob zwei Werte gleich sind. Dies müssen keine Zahlen sein.
- Der untenstehende Code testet folgende Beziehung: Das BIP in FantasiAland ist gleich groß wie in FantasiCland.

```
FantasiAland == FantasiCland
```

```
#> [1] FALSE
```

- Da die beiden Werte nicht gleich sind, ist die Antwort `FALSE`.
- Zur Erinnerung: Statt mit den Objekten zu rechnen, könnten wir auch einfach die Zahlenwerte verwenden:

```
1000000 == 1020000
```

```
#> [1] FALSE
```

Logische Operatoren

- `!=` evaluiert, ob zwei Werte nicht gleich sind.
- Der nachfolgende Code testet folgende Beziehung: Das BIP in FantasiAland ist nicht gleich groß wie in FantasiCland.

```
FantasiAland != FantasiCland
```

```
#> [1] TRUE
```

- Da die beiden Werte nicht gleich sind, ist die Antwort `TRUE`.

Logische Operatoren kombinieren

- Logische Operatoren lassen sich mit `&` und `|` beliebig untereinander kombinieren.

| Logischer Operator | Bedeutung |
|--------------------|-----------|
| <code>&</code> | und |
| <code> </code> | oder |

- Falls uns interessiert ob **beide Aussagen** korrekt sind, nutzen wir `&`.
- Ob eine Kombination sinnhaftig ist, bleibt unsere Entscheidung.
- Denn `1000000 > 1020000 & 1000000 < 1020000` ist ein zulässiger logischer Test. Mit Objekten:

```
FantasiAland > FantasiCland & FantasiAland < FantasiCland
```

```
#> [1] FALSE
```

Logische Operatoren kombinieren

- Falls uns interessiert, ob **mindestens eine von beiden Aussagen** stimmt, schreiben wir:

```
FantasiAland > FantasiCland | FantasiAland < FantasiCland
```

```
#> [1] TRUE
```

- Auch die logischen Operatoren können beliebig häufig verkettet werden:

```
(FantasiAland > FantasiCland | FantasiAland < FantasiCland) & FantasiDland == FantasiEland
```

```
#> [1] FALSE
```

Noch mehr Objekte

- Bisher haben wir mathematische und logische Operatoren kennengelernt und erste Objekte erstellt.
- Die Objekte bisher enthielten allerdings jeweils nur eine Zahl (Skalare).
- R kann natürlich noch viel mehr.
- Objekte können jegliche Art von Datentypen enthalten.
- Wir schauen uns als Nächstes die häufigsten Daten**typen** in R an.
- Auf dem Weg erstellen wir **Vektoren**, nach den Skalaren, die nächst komplexere Daten**struktur** in R.

Vektoren

Vektoren

- Vektoren sind Matrizen der Dimension $n \times 1$ (bedeutet n Zeilen und 1 Spalte).
- Um in R einen Vektor zu erstellen, nutzen wir die Funktion `c()` (steht für *concatenate* [verketten]).
- Beispiel mit Zahlen:

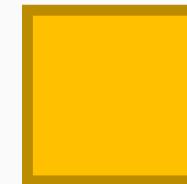
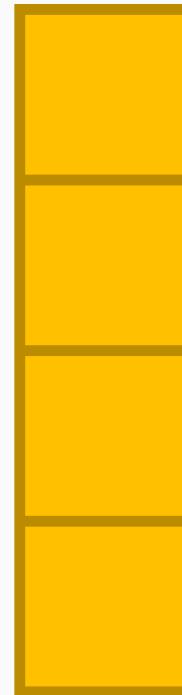
```
c(1, 100, 1000, 10000)
```

```
#> [1] 1 100 1000 10000
```

- Beispiel mit Text:

```
c("Hallo", "ich", "bin", "ein", "Text")
```

```
#> [1] "Hallo" "ich"   "bin"   "ein"   "Text"
```



scalar

vector

- Wir können unsere zuvor erstellten BIPs, die wir einzelnen Objekten zugewiesen haben, verbinden.

```
alle_bips <- c(FantasiAland, FantasiBland, FantasiCland,  
FantasiDland, FantasiEland)
```

- Es entsteht ein Vektor, der die BIPs der fünf Länder enthält:

```
alle_bips
```

```
#> [1] 1000000 1100000 1020000 3100000 1000000
```

- Mit (numerischen) Vektoren kann man rechnen:

```
alle_bips / 5000
```

```
#> [1] 200 220 204 620 200
```

- Und wir können logische Tests durchführen:

```
alle_bips == FantasiAland
```

```
#> [1] TRUE FALSE FALSE FALSE TRUE
```

- Zur Erinnerung: Die vergebenen Namen für die Objekte sind für logische Tests irrelevant. Es kommt darauf an, welche Inhalte hinter den jeweiligen Namen stecken.

Datentypen

- Datenstrukturen können verschiedene Datentypen enthalten.
- Zu den relevantesten Datentypen in R gehören die folgenden:

| Datentyp | Beispiel | Hinweis |
|-----------|--------------------------|---------------------------|
| double | 3.1, 7.9, 10.3 | numeric |
| integer | 1, 2, 3, 4 | numeric |
| logical | TRUE oder FALSE | - |
| character | hallo, ich bin, ein text | strings/text |
| factor | divers, frau, mann | numeric/character/logical |

Datentypen

- Beispielvektoren:

```
vect_double <- c(3.1, 7.9, 10.3)
vect_integer <- c(1L, 2L, 3L)
vect_logical <- c(TRUE, FALSE, TRUE, TRUE)
vect_character <- c("strings", "text", "character")
vector_factor <- factor(c("divers, frau, mann"))
```

- Wir können die Datentypen mit der Funktion `typeof()` überprüfen, z. B.:

```
typeof(vect_double)
```

```
#> [1] "double"
```

- Oder:

```
typeof(vect_character)
```

```
#> [1] "character"
```

Datentypen

- Es kommt gelegentlich vor, dass wir einen Datentyp erzwingen wollen.
- Das ist z. B. hilfreich, wenn beim Einlesen der Daten, der Datentyp nicht korrekt erkannt wurde.
- Wir nutzen hierfür die `as. . . ()`-Funktionen.

`as.numeric()`

`as.character()`

`as.factor()`

...

Datentypen

- Insbesondere die Funktion `as.factor()` nutzen wir häufiger.
- Faktoren sind eigentlich eine Daten**klasse** und kein Datentyp. Die Funktion `as.factor()` wandelt verschiedene Datentypen um. Sie werden dadurch zum Typ `integer` aber zur Klasse `factor`.
- Ausgangspunkt:

```
typeof(vect_character)
```

```
#> [1] "character"
```

- Datentyp:

```
vect_character_to_vector <- as.factor(vect_character)
typeof(vect_character_to_vector)
```

```
#> [1] "integer"
```

- Datenklasse:

```
class(vect_character_to_vector)
```

```
#> [1] "factor"
```

Datentypen

- Braucht man das wirklich?
- Ja.
- Sind Datentypen kompliziert?
- Nein.
- Die gute Nachricht: **Oft funktioniert es einfach wie es soll.**
- Im Moment stehen diese noch im leeren Raum. Bei der Arbeit mit Datensätzen wird uns klarer, welche Funktionsweisen welcher Datentyp bedingt.

Genug der Theorie. Ab nach

