

Eine Einführung in R

Basics

Peer Keßler

peer.kessler@uni-greifswald.de

10. November 2025

1. Wie arbeiten wir in R?
2. Was gibt es in R?
3. Atomare Datentypen
4. Funktionen
5. Objekte an Namen binden
6. Strukturierte Datentypen
7. Vektoren
8. Matrizen
9. Listen
10. Data Frames
11. Ausblick

Was ist R?





- Freie Programmiersprache
- Ursprünglich für statistische Berechnungen entwickelt
- Mittlerweile universell einsetzbar (Analyse, Visualisierung, Apps, u.v.m.)
- Durch Pakete praktisch unbegrenzte Erweiterbarkeit
- Große und aktive Community (Stack Overflow, R Bloggers, uvm.)

Wie kann man R lernen?



- Freie Programmiersprache
- Ursprünglich für statistische Berechnungen entwickelt
- Mittlerweile universell einsetzbar (Analyse, Visualisierung, Apps, u.v.m.)
- Durch Pakete praktisch unbegrenzte Erweiterbarkeit
- Große und aktive Community (Stack Overflow, R Bloggers, uvm.)

Wie kann man R lernen?

- Basic “Vokabeln” und “Grammatik”
- Sprachen kann man nur durch Üben lernen!
- Problem: Viele verschiedene Wege zum Ziel (in etwa: “Es gibt viele Dialekte”)

Wie arbeiten wir in R?



- Integrierte Entwicklungsumgebung (IDE) für R
- Bietet viele nützliche Features
- Kostenlos und Open Source
- Download: R Studio Download
- Bietet Oberflächen für Python, LaTex, C++, SQL, uvm.

Was gibt es in R?

"Everything that exists in R is an object. Everything that happens in R is the result of a function call."

- John M. Chambers

Atomare Datentypen

In R existieren verschiedene atomare Datentypen aus welchen sich wiederum komplexere Datentypen konstruieren lassen.

| Datentyp | Beschreibung | Beispiel |
|-------------|-------------------------------|----------|
| integer | ganze Zahlen | -2L |
| numeric | reelle Zahlen | 5.2456 |
| logical | logische Werte | TRUE |
| character | Zeichenfolge | "Mexico" |
| NA,NULL,NaN | Unbestimmt, Leer, Undefiniert | |
| Inf | Unendlich | |

Welche Operationen/Funktionen auf Daten anwendbar sind, hängt zentral vom Datentyp ab!

→ Know your data!

Die Klasse bzw. der Datentyp eines Objekts lässt sich mit der Funktion `class()` ermitteln oder aber spezifisch/logisch prüfen mit der `is. ...()` Funktionsfamilie.

```
class(-2L)
```

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

```
is.numeric(5)
```

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

Arithmetische Grundoperationen

R beherrscht für alle Zahlen (numeric und integer) die arithmetischen Grundoperationen.

| Operation | Beschreibung | Beispiel |
|-----------|--------------------------|----------|
| +,- | Addition, Subtraktion | 3-1.2 |
| *,/ | Multiplikation, Division | 4.8/4 |
| ^ | Potenz | 5^2 |

Beispiel:

`1+2`

`## [1] 3`

`2*3.5`

`## [1] 7`

`10.1^3`

`## [1] 1030.301`

Lassen sich mathematische Operationen auch auf Werte des Typs logical anwenden?

Umwandeln von Datentypen

Datentypen lassen sich im begrenzten Maße in andere Datentypen umwandeln. Das ist vor allem über die `as. ...()` Funktionsfamilie möglich.

```
as.numeric(TRUE)

## [1] 1

as.character(3.14)

## [1] "3.14"

as.logical(0)

## [1] FALSE

as.numeric("eins")

## Warning: NAs introduced by coercion

## [1] NA
```

Funktionen

Funktionen...

- sind *self-contained* (eigenständig).
- helfen beim Erreichen bestimmter Ziele.
- bestehen im Wesentlichen aus einer Komposition von *base*-Funktionen.

Funktionensafrufe erfolgen in der Regel über das Schema

`Funktionsname(Argument 1, Argument 2, ...)`

Argumente sind *meistens* ...

- ein oder mehrere **Objekte**, auf die sich die Funktion bezieht und
- **Optionen**, die die Funktionsweise der Funktion verändern.

```
round(x = 3.14159,  
      digits = 2  
      )
```

- `x` ist hier das Objekt
- `digits` ist die Option

Die Argumente einer Funktion folgen einer bestimmten Reihenfolge:

```
round(3.14159, 2)
```

```
## [1] 3.14
```

```
round(2, 3.14159)
```

```
## [1] 2
```

Offensichtlich nutzt R das erste Argument als Objekt und das zweite als Option.

Die Reihenfolge der Argumente kann verändert werden, wenn die Argumente benannt werden.

```
round(x = 3.14159,  
      digits = 2  
    )
```

```
## [1] 3.14
```

```
round(digits = 2,  
      x = 3.14159  
    )
```

```
## [1] 3.14
```

Hinweis

Denkt dran, dass der Code auch nach langer Zeit noch lesbar sein soll bzw. andere sollen euren Code auch lesen können. Daher gehört es zur guten Praxis, die Argumente zu benennen.

Niemand muss die Bedeutung der Argumente einer Funktion oder ihre Default-Werte auswendig lernen. Für jede Funktion existiert eine sogenannte **Documentation**. Um diese aufzurufen kann nach folgendem Schema aufgerufen werden:

```
?function
```

Ist jedoch auch der konkrete Funktionsaufruf unbekannt, kann innerhalb der Documentation nach der Funktion gesucht werden. Die Documentation wird dann nach dem übergebenen Schlagwort durchsucht:

```
??average
```

Objekte an Namen binden

“Names have objects; objects don’t have names.”

- Hardley Wickham

Mit dem Operator `<-` lässt sich ein referenzierbares Objekt erstellen. Das bedeutet, dass das Objekt an einen Namen gebunden wird:

```
x_test <- 3
```

Nun kann das Objekt `x_test` überall dort verwendet werden, wo der Wert 3 gebraucht wird.

```
x_test
```

```
## [1] 3
```

Strukturierte Datentypen

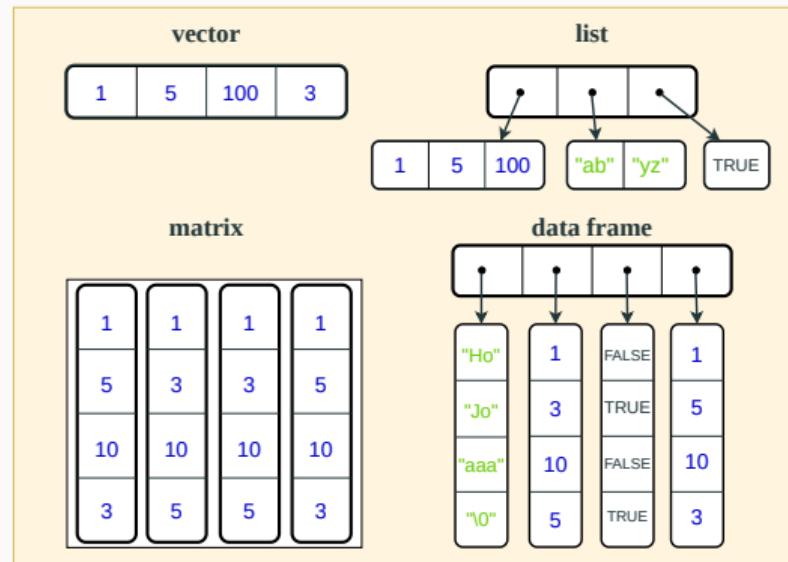
Basierend auf den einfachen Datentypen existieren in R die folgenden grundlegenden Datenstrukturen.

| Datenstruktur | Beschreibung |
|---------------|---|
| vector | Sequenz gleicher Datentypen |
| matrix | Verallg. Vektor in 2 Dimensionen |
| array | Verallg. Vektor mit beliebigen Dimensionen |
| list | Sequenz ungleicher Datentypen |
| data frame | Spezielle Liste mit Vektoren gleicher Länge |

Schema:

| Dimension | Homogen | Heterogen |
|-----------|---------|------------|
| 1-Dim. | vector | list |
| 2-Dim. | matrix | data frame |
| k-Dim. | array | |

Vereinfachte Darstellung Datentypen



Vektoren

Vektoren sind sequentiell geordnete Folgen von Werten *gleichen Typs*. Vektoren können auf ganz unterschiedliche Art generiert werden.

Beispiel: Bedeutende Forscher*innen zu Dual Process Ansätzen. Mit `c()` (kurz für ‘concatenate’ / ‘combine’)

```
dpt.Alter <- c(86, 92, 80, 68)
dpt_name <- c("Jonathan Evans", "Daniel Kahneman",
            "Ann Swidler", "Karen Cerulo")
dpt_psych <- c(T, T, F, F)

dpt.Alter
## [1] 86 92 80 68

dpt_name
## [1] "Jonathan Evans"  "Daniel Kahneman" "Ann Swidler"      "Karen Cerulo"

dpt_psych
```

Einfache Zahlenfolgen

Mit dem Colon-Operator : lassen sich leicht Folgen mit Inkrement 1/-1 erzeugen

`1:4`

```
## [1] 1 2 3 4  
countdown <- 10:0  
countdown
```

```
## [1] 10 9 8 7 6 5 4 3 2 1 0
```

`-1.2:5`

```
## [1] -1.2 -0.2  0.8  1.8  2.8  3.8  4.8
```

Allgemeinere Folgen kann man mit seq() erzeugen

```
seq(from = 1,  
    to = 3,  
    by = 0.5  
)  
  
## [1] 1.0 1.5 2.0 2.5 3.0
```

Auch sehr nützlich:

```
rep(x = c("Nein!", "Doch!"),  
    times = 3  
)  
  
## [1] "Nein!" "Doch!" "Nein!" "Doch!" "Nein!" "Doch!"
```

Abfrage von Werten durch Subsetting von Vektoren

Mit dem Operator [] können einzelne Elemente eines Vektors abgefragt werden.

```
dpt_alter[2]
```

```
## [1] 92
```

Der Operator ist aber tatsächlich ein "Subset"-Operator und damit wesentlich flexibler und sehr mächtig verwendbar:

```
dpt_name[c(1, 3)]
```

```
## [1] "Jonathan Evans" "Ann Swidler"
```

```
dpt_name[-c(1, 3)]
```

```
## [1] "Daniel Kahneman" "Karen Cerulo"
```

Man sollte also [] wirklich immer als einen Operator verstehen, welcher von links auf einen Vektor angewendet wird!

Übergibt man [] einen logischen Vektor (passender Länge), so werden alle korrespondierenden "wahren" Elemente eines Vektors ermittelt.

Dies ist tatsächlich eine der häufigsten Verwendungen von []:

```
dpt_name  
## [1] "Jonathan Evans"  "Daniel Kahneman" "Ann Swidler"      "Karen Cerulo"  
dpt_psych  
## [1] TRUE  TRUE FALSE FALSE  
dpt_name[dpt_psych]  
## [1] "Jonathan Evans"  "Daniel Kahneman"
```

Alle atomaren Datentypen sind tatsächlich Vektoren der Länge 1

```
zahl <- 1
p <- TRUE
name <- "Tom"

length(zahl)

## [1] 1

length(p)

## [1] 1

length(name)

## [1] 1
```

Vektorisierte Operation

Analog agieren dann (fast) alle Operationen, welche für atomare Datentypen definiert sind als vektorisierte Operation, d.h. elementweise:

```
x <- c(1, 2, 3)
y <- c(2, 1, 3)

x + y

## [1] 3 3 6
x + 1

## [1] 2 3 4
log(x)

## [1] 0.0000000 0.6931472 1.0986123
dpt.Alter < 70

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

Operationen auf Vektoren

Abhängig vom Datentyp eines Vektors existieren verschiedene Funktionen, welche als Argument einen Vektor nehmen und auf ein Resultat abbilden.

| Operation | Beschreibung |
|-----------|---|
| length() | Länge eines Vektors |
| mean() | Durchschnittswert eines num. Vektors |
| max() | Maximum eines num. Vektors |
| any() | Ist irgendein Wert eines log. Vektors wahr? |

```
mean(dpt_alter)
```

```
## [1] 81.5
```

```
any(dpt_psych)
```

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

Matrizen

Matrix

Die Datenstruktur einer Matrix verallgemeinert das Konzept eines Vektors in zwei Dimensionen. Eine Matrix kann z.B. durch einen Vektor mit dem Befehl `matrix()` generiert werden:

```
M <- matrix(data = 1:9,  
            nrow = 3,  
            ncol = 3,  
            byrow = TRUE  
          )
```

M

```
##      [,1] [,2] [,3]  
## [1,]    1    2    3  
## [2,]    4    5    6  
## [3,]    7    8    9
```

Indizierung ist ähnlich zu Vektoren:

```
M[1, 2]
```

```
## [1] 2
```

Abrufen einer Zeile

```
M[2, ]
```

```
## [1] 4 5 6
```

Abrufen einer Spalte

```
M[, 1]
```

```
## [1] 1 4 7
```

Benennung von Spalten

Mit `colnames()` lassen sich einerseits die Spalten (`columns`) einer Matrix benennen/ändern, als auch abrufen.

```
colnames(M) <- c("A", "B", "C")
M

##      A B C
## [1,] 1 2 3
## [2,] 4 5 6
## [3,] 7 8 9

colnames(M)

## [1] "A" "B" "C"
```

Die Funktion `rownames()` macht dann gleiches für die Zeilen einer Matrix.

Indizierung über Spaltennamen

Über die Spaltennamen können wir nun auch auf die Elemente der Matrix zugreifen.

```
M[, "B"]
```

```
## [1] 2 5 8
```

Zwei Vektoren können mit der Funktion `cbind()` in eine Matrix überführt werden.

```
D <- cbind(dpt_alter, dpt_psych)
```

```
D
```

```
##          dpt_alter dpt_psych
## [1,]         86         1
## [2,]         92         1
## [3,]         80         0
## [4,]         68         0
```

Analog können mit `rbind()` zwei Vektoren als Zeilen zu einer Matrix gebunden werden.

Listen

Eine Liste ist ein “verallgemeinerter Vektor” und lässt als Elemente beliebige Werte oder Datenstrukturen zu:

```
profil_marie <- list(Name = "Marie",
                      Freund_innen = c("Daphne", "Peer"),
                      Alter = 24
)
profil_marie

## $Name
## [1] "Marie"
##
## $Freund_innen
## [1] "Daphne" "Peer"
##
## $Alter
## [1] 24
```

```
profil_marie$name  
## [1] "Marie"  
profil_marie$freund_innen  
## [1] "Daphne" "Peer"
```

Indizierung von Listen

```
profil_marie[2]  
  
## $Freund_innen  
## [1] "Daphne" "Peer"  
  
profil_marie[[2]][1]  
  
## [1] "Daphne"  
  
profil_marie$Freund_innen[1]  
  
## [1] "Daphne"
```

Data Frames

Die für statistische Zwecke häufigste und wichtigste Datenstruktur ist die einer Datentabelle, ein sogenanntes `dataframe`.

Erzeugung aus Vektoren:

```
df_scholar <- data.frame(Name = dpt_name,  
                         Alter = dpt_alter  
                         )  
  
df_scholar
```

```
##           Name Alter  
## 1  Jonathan Evans    86  
## 2 Daniel Kahneman    92  
## 3     Ann Swidler    80  
## 4   Karen Cerulo    68
```

Analog zu Matrizen:

```
df_scholar[3, ]  
  
##           Name Alter  
## 3 Ann Swidler     80  
  
df_scholar[, 2]  
  
## [1] 86 92 80 68
```

Analog zu Listen:

```
df_scholar$name  
  
## [1] "Jonathan Evans"  "Daniel Kahneman" "Ann Swidler"      "Karen Cerulo"
```

Aufname weiterer Variablen/Spalten

Binden analog zu Matrizen:

```
df_scholar <- cbind(df_scholar, dpt_psych)  
df_scholar
```

```
##           Name Alter dpt_psych  
## 1  Jonathan Evans    86      TRUE  
## 2 Daniel Kahneman    92      TRUE  
## 3   Ann Swidler     80     FALSE  
## 4 Karen Cerulo      68     FALSE
```

Aufname weiterer Variablen/Spalten

```
df_scholar$Geburtsjahr <- 2024 - dpt_alter  
df_scholar  
  
##           Name Alter dpt_psych Geburtsjahr  
## 1 Jonathan Evans    86      TRUE     1938  
## 2 Daniel Kahneman   92      TRUE     1932  
## 3 Ann Swidler       80     FALSE     1944  
## 4 Karen Cerulo       68     FALSE     1956
```

Analog zu Vektoren:

```
mean(df_scholar$Alter)
```

```
## [1] 81.5
```

Ausblick

Neben der Basisfunktionalität von R existieren zahlreiche Packages, mit denen der Funktionsumfang quasi beliebig erweiter werden kann. Das wohl wichtigste Package ist die Packagesammlung *tidyverse*, die die klassische Grammatik für statistische Analysen darstellt.

```
tibble(Name = c("Anna", "Berta", "Cora"),
       Alter = c(23, 25, 22)
     ) %>%
  mutate(Geburtsjahr = 2024 - Alter,
        Bildung = c("Low", "Middle", "High")
      ) %>%
  filter(Alter > 22) %>%
  arrange(desc(Bildung)) %>%
  ggplot(mapping = aes(x = Name,
                        y = Alter
                      )
        ) +
  geom_bar(stat = "identity")
```

Fragen?

Danke für eure Aufmerksamkeit! :-)