Computational Social Science Workshop

A Gentle Introduction to R: Basics

R User Group

05 11 2018

Grundlagen

Zunächst einmal die Basics zu Rmarkdown.

Das ist ein Chunk:

```
print("Hello World!")
```

```
## [1] "Hello World!"
```

Klick doch einfach mal auf den grünen "Play" Button.

R als Taschenrechner

- `+` addieren
- `-` subtrahieren
- `*` multiplizieren
- `/` dividieren
- `^` exponieren

Aufgaben:

1. 34+77
2. (500 geteilt durch 125) plus 3 mal 6 hoch 2
3. Berechne die Differenz zwischen 2018 und dem Jahr, an dem du begonnen hast zu studieren und dividiere das durch die Differenz zwischen 2018 und dem Jahr, in dem du geboren wurdest. Multipliziere dies mit 100, um den Prozentsatz deines Lebens zu erhalten, den du an die Uni verbracht hast.

```
34  + 77
```

```
## [1] 111
```

```
(500 / 125) + 3 * 6^2
```

```
## [1] 112
```

```
(2018 - 2013)/(2018 - 1991)
```

```
## [1] 0.1851852
```

Objekte und Zuweisung

In R kann man je nach Rechenleistung zehntausende Datensätze auf einmal laden und verarbeiten. Das wird möglich da jeder Datensatz seinen eigenen Namen besitzt. Dieser muss mit einem Buchstaben beginnen, kann aber durchaus auch . oder _ oder Zahlen enthalten. Um einem Objekt einen Namen zuzuweisen wird das Zeichen <- (Zuweisungspfeil) eingesetzt. Objekte können einzelne Zahlen, Variablen oder auch Datensätze sein. Mit den Objektennamen können Daten jeglicher Art abgespeichert und abgerufen werden.

Führe einfach mal den folgenden Chunk aus!

```r
x <- 2   #definiere x als 1

x
```

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

Aufgaben:

4. Erstelle ein neues Objekt y mit deiner Lieblingszahl. Adddiere x und y und speichere das Ergebnis in z. Gebe z aus!

```r
y <- 21

z <- x + y

z
```

```
## [1] 23
```

Wichtige Operatoren

Boolean

- `&` (logisch) und
- `|` (logisch) oder
- `!` (logisch) nicht

Weitere Operatoren

- `==` (logisch) ist gleich
- `!=` (logisch) ist ungleich
- `%in%` (string) ist gleich
- `>` größer als
- `<` kleiner als
- `>=` größer gleich
- `<=` kleiner gleich
- `is.na()` ist gleich NA (fehlender Wert)
- `!is.na()` ist ungleich NA

Beispiele

```r
3 == 3    #ist 3 gleich 3?
```

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

```r
4 > 5     #ist 4 größer als 5?
```

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

```r
"albert" == "albert"   #ist albert gleich albert vor?
```

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

R gibt hier entweder `TRUE` oder `FALSE` aus, abhängig davon ob der benutzte logische Operator zutrifft oder nicht.

Vektoren I

Vektoren in R sind einfach mehrere Werte die aneinander gebunden werden. Diese können Zahlen aber auch strings (Buchstabenfolgen) sein. Hilfreich dabei ist die `c()` Funktion (c steht für *concatenate* = verketten auf

Deutsch).

Beispiele:

```
c(4, 1, 3, 4, 8)
```

```
## [1] 4 1 3 4 8
```

```
vektor <- c(4, 1, 3, 4, 8) #definiere Vektor mit den Werten 4, 1, 3, 4 und 8

vektor
```

```
## [1] 4 1 3 4 8
```

Operatoren können auch ganz einfach mit Vektoren angewandt werden.

```
vektor * vektor     #vektor mit sich selbst mal nehmen
```

```
## [1] 16  1  9 16 64
```

```
vektor == vektor    #sind alle Werte von vektor gleich alle Werte von vektor?
```

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

Aufgabe

5. Bilde einen Vektor mit dem Namen `cm` und den folgenden Größen in Zentimeter:

- 190
- 152
- 174

Teile dann `cm` durch Hundert um die Größen in Meter zu erhalten!

```
cm <- c(190, 152, 174)

cm / 100
```

```
## [1] 1.90 1.52 1.74
```

Wollen wir auf einen bestimmten Wert in unserem Vektor zugreifen, dann benutzen wir **eckige Klammern** `[...]` auf die folgende Art und Weise:

```
vektor
```

```
## [1] 4 1 3 4 8
```

```
vektor[3]  #gibt uns das dritte Element von vektor aus
```

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

```
vektor[5]  #gibt uns das fünfte Element von vektor aus
```

```
## [1] 8
```

```
vektor[1] / vektor[4] #teile die erste Stelle von vektor durch die fünfte Stelle von vektor
```

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

Funktionen

Das Ziel einer Funktion ist es Daten zu verarbeiten. Dazu gibt es einen *input* und einen *output*.

Definieren wir doch mal eine Funktion, welche Werte die man als *input* eingibt wieder quadriert als *output* gibt. Dazu benutzen wir `function()` und bestimmen dabei die Argumente welche wir jeweils eintippen wollen. Die Funktion selber wird in geschweiften Klammern `{}` codiert.

Beispiele:

```
quadrieren <- function(input) {
  output <- input^2        #nehme den input hoch 2 und speichere ihn in output
  return(output)                    #gebe output aus
}

#Funktion ausprobieren!
quadrieren(9)
```

```
## [1] 81
```

Wir können auch zwei oder mehrere Argumente zu einer Funktion hinzufügen. Probieren wir das gleiche doch mal mit einer Funktion die jeder aus dem Physik - Unterricht kennen sollte:

Distanz = Geschwindigkeit * Zeit

$$Distanz = Geschwindigkeit \times Zeit$$

bzw.

$$s = v * t$$

s = v * t

```
distanz <- function(v, t) {
 s <- v * t       #v mal t und definiere deren Ergebnis als s
 s                #gebe s aus
}

#Funktion ausprobieren!
distanz(v = 50, t = 6)
```

```
## [1] 300
```

Nun müssen wir allerdings nicht immer selber Funktionen definieren. Im Gegenteil! `Base R` und die vielen Packages haben unzählige Funktionen, die Datenbearbeitung einfach machen. Beispielsweise gibt `mean()` den Mittelwert eines Vektors aus und `sd()` gibt die Standardabweichung.

```
vektor2 <- 1:5
mean(vektor2)   #gib den Mittelwert von vektor aus
```

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

```
sd(vektor2)     #gib die Standardabweichung von vektor aus
```

```
## [1] 1.581139
```

Vektoren II

Nun eine kleine Übersicht über die Arten von Vektoren:

Es gibt drei Haupttypen

- **Character:** aka “String” oder ‘String’, ist einfach nur Text.
- **Factor:** Factors können eine Reihenfolge haben sogennante *Levels*.
- **Numeric:** Jeglicher Zahleninput mit dem man rechnen kann.

```
char <- c("Männlich", "Weiblich")
fac <- factor(c("Männlich", "Weiblich"))
num <- c(1, 2, 3)

class(char)
```

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

```
class(fac)
```

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

```
class(num)
```

```
## [1] "numeric"
```

Tranformation zwischen den Datentypen

- `as.character()`
- `as.factor()`
- `as.numeric()`

```
as.character(num)
```

```
## [1] "1" "2" "3"
```

```
as.factor(char)
```

```
## [1] Männlich Weiblich
## Levels: Männlich Weiblich
```

```
as.numeric(fac)
```

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