Basics

22. November 2019

1 Grundoperationen in R

In R lassen sich Vektoren[1] subtrahieren, addieren, multiplizieren, dividieren, potenzieren, logarithmieren und natürlich lassen sich auch Wurzeln ziehen.

Im Folgenden sind einige einfache Rechnungen aufgeführt.

```
3 + 4 * 5
```

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

```
14 - 2 * 7
```

```
## [1] 0
```

```
3^3
```

```
## [1] 27
```

```
(3^3)^2
```

```
## [1] 729
```

```
sqrt(36)
```

```
## [1] 6
```

```
sqrt(4^2)
```

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

```
log(10)
```

```
## [1] 2.3
```

Wir können problemlos auch Vektoren mit mehr als einem Element erstellen. Man erreicht dies durch die c()-Funktion, wobei das "c" für "combine" steht. Beispielsweise könnte ein Vektor so aussehen:

```
c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

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

[1] Zahlen werden von R als Vektoren interpretiert, hier also Vektoren mit jeweils einem Element.

Als nächstes wollen wir einen Vektor mit den Zahlen 1:100 erstellen. Man kann sich hierbei jedoch die Tipparbeit sparen, da es in R einen Sequenzoperator, den Doppelpunkt, gibt, der uns die Arbeit abnimmt. Der Doppelpunkt steht für "von bis".

```
c(1:100)
```

Dass hier das Ergebnis nicht angezeigt wird sei mir verziehen. Überprüfen Sie es gerne selbst, Sie werden sehen, es funktioniert. Die Taschenrechner-Funktionen von R beschränken sich nicht nur auf Zahlen (Vektoren mit einem Element). Genauso lassen sich auch die verschiedenen Rechenoperationen mit "längeren"Vektoren durchführen. Hier einige Beispiele um zu erkennen, wie R arbeitet:

```
c(1, 2, 3) * 3

## [1] 3 6 9

c(2, 2) + c(3, 4)

## [1] 5 6

c(1, 2, 3) + c(20, 200)

## [1]  21 202  23
```

Im letzten Beispiel gibt R eine Warnmeldung aus, weil die Länge des ersten Vektors kein Vielfaches der Länge des zweiten Vektors ist. Nachdem die ersten beiden Rechnungen (1+20) und (2+200) durchgeführt wurden, „springt“ R wieder an den Anfang des zweiten Vektors um zur Zahl 3 die 20 zu addieren. Die Berechnung wird beendet, obwohl R noch nicht das Ende des zweiten Vektors erreicht hat. Es folgt eine Warnmeldung.

Der Vollständigkeit halber hier der Code ohne Warnung:

```
c(1, 2, 3, 4) + c(20, 200)

## [1]  21 202  23 204
```

Variablen

Wie in jeder anderen Programmiersprache ist auch in R die Zuweisung von Objekten zu Variablen möglich. Wie der Name schon sagt, sind die Inhalte einer Variablen veränderbar. Der Wert einer neuen Variable kann selbst wieder eine Variable sein. Der Zuweisungsoperator in R (im Engl. “assignment operator”) ist ein Pfeil (“<-”).

```
x <- 1
x

## [1] 1

(y <- x/2)

## [1] 0.5

(z <- y + x)

## [1] 1.5
```

Im ersten Beispiel wird dem Symbol[2] x der Wert "1" zugewiesen. Im zweiten Beispiel wird dem Symbol "y" der Wert der Variable "x", dividiert durch "2", zugewiesen.[3] Im dritten Beispiel werden die beiden Variablen "x" und "y" addiert und in die Variable "z" gespeichert.

1.1 Datentypen

Bisher haben wir gesehen, wie mit Zahlen und Vektoren gerechnet werden kann und wie sich Werte Variablen zuweisen lassen. Im folgenden Abschnitt werden weitere wichtige Objekte[4] vorgestellt. Außerdem wird erklärt, wie auf einzelne Elemente innerhalb der verschiedenen Objekte zugegriffen werden kann (auch als "Indexing" bezeichnet).

1.1.1 Vektoren

Die einfachsten Objekte in R sind die Vektoren. Wir haben sie schon in 1 kennengelernt. Es gibt jedoch nicht nur numerische Vektoren, sondern auch Vektoren, die aus Buchstaben bzw. Wörtern (sog. character) oder logischen Operatoren (TRUE/FALSE) bestehen. Sie werden durch ein c() erstellt. Die einzelnen Werte innerhalb des Vektors werden durch Kommas voneinander getrennt. Nachkommastellen werden in R mit einem Punkt gekennzeichnet.

```
# ein Vektor mit Zahlen (numerisch)
c(1.2, 2.4, 3.6, 4.8, 5)

## [1] 1.2 2.4 3.6 4.8 5.0

# ein Vektor mit Wörtern (character)
c("Apfel", "Birne", "Banane")

## [1] "Apfel"  "Birne"  "Banane"

# ein Vektor mit logischen Operatoren (logical)
c("TRUE", "TRUE", "FALSE", "FALSE")

## [1] "TRUE"  "TRUE"  "FALSE" "FALSE"

# ein 'gemischter' Vektor
c(24, "Stunden")

## [1] "24"      "Stunden"
```

1.1.2 Matrizen

Wie Sie vermutlich schon wissen, lassen sich verschiedene Verfahren auch mittels matrix-algebraischer Herangehensweise lösen. Zwar ist die Berechnung per Hand äußerst mühsam, aus Sicht der Programmierung aber vorteilhaft. Das "Allgemeine Lineare Modell" bietet einen Bezugsrahmen, durch den sich die Unterschiede zwischen regressions- und varianzanalytischen Ansätzen weitgehend aufheben[5]. Weil dies so ist, beschäftigen wir uns im folgenden mit Matrizen. Matrizen besitzen im Gegensatz zu Vektoren

[2] In R wird der Variablenname als "Symbol" bezeichnet

[3] Hinweis: die Klammern in Beispiel 1&2 dienen sowohl dem Speichern wie dem sofortigen Darstellen des Inhalts der Variable

[4] Sie werden in den seltensten Fällen falsch liegen, wenn Sie etwas inR als „Objekt" bezeichnen.

[5] Im ALM ist die ANOVA eine bestimmte Form der Regression

eine zwei-dimensionale Datenstruktur. Wie werden Matrizen erstellt? Hierfür gibt es die matrix Funktion. Hier einige Beispiele, die den Aufbau verdeutlichen:

```
matrix(1:6)
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
## [4,]    4
## [5,]    5
## [6,]    6
```

```
# zwei Zeilen:
matrix(1:6, nrow = 2)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
# 'by row' 'wahr' machen
matrix(1:6, nrow = 2, byrow = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
# Spalten- und Zeilennamen noch:
matrix(1:6, nrow = 2, byrow = TRUE, dimnames = list(c("A", "B"), c("X", "Y", "Z")))
```

```
##   X Y Z
## A 1 2 3
## B 4 5 6
```

Im letzten Beispiel wurden den entsprechenden Zeilen und Spalten der Matrix Namen gegeben. Es fällt auf, dass hier eine bislang unbekannte Funktion, nämlich list(), verwendet wurde. Eine Liste ist, wie so vieles in R, ein Objekt. In 1.1.4 wird erklärt was Listen sind und wie diese erstellt werden.

In manchen Fällen kann es sein, dass R als Input einer bestimmten Funktion den Datentyp Matrix erwartet. Nicht weiter schlimm - unser Objekt wird durch die Funktion as.matrix() in eine Matrix transformiert.

Im nächsten Abschnitt widmen wir uns einem weiteren wichtigen Objekttyp in R, den Faktoren. Bevor wir dies jedoch tun, wollen wir einen anderen Objekttyp, die arrays, nicht unerwähnt lassen, auch wenn Sie mit diesem Typ in den Anfängen wenig bis keinen Kontakt haben werden. Arrays sind mehrdimensionale Vektoren und lassen sie durch die array-Funktion erstellen. Ein kleines Beispiel soll hier genügen. Wir erzeugen einen „2*2*2-array“:

```
array(1:8, dim = c(2, 2, 2))
```

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

1.1.3 Faktoren

Bei der Datenauswertung, sagen wir eines Fragebogens, finden sich i.d.R. eine oder mehrere diskrete/kategoriale Variablen. Kategoriale Variablen liegen in Nominal- oder Ordinalskalenniveau vor.[6] Wir müssen diese Variablen in R als solche zunächst kennzeichnen (entweder als nominal oder ordinal), um mit ihnen weiterarbeiten zu können. Ein Beispiel: Wir haben von vier Personen eine Angabe über das Geschlecht. Die Daten liegen in Form eines numerischen Vektors vor. Also so:

```
c(1, 1, 1, 2)
```

Diesen Vektor wandeln wir nun in einen Faktor um:

```
factor(c(1, 1, 1, 2))

## [1] 1 1 1 2
## Levels: 1 2
```

Der einzige Unterschied zwischen dem Ouput des Vektors und einem Faktor ist die zusätzliche Angabe der Levels. Anschaulicher wird es, wenn die Levels noch umbenannt werden, sodass wir sehen was sich hinter den beiden Zahlen „vebirgt".[7]

```
factor(c(1, 1, 1, 2), labels = c("weiblich", "männlich"))

## [1] weiblich weiblich weiblich männlich
## Levels: weiblich männlich
```

Nun ein weiteres Beispiel um zu verdeutlichen, wie sich eine ordinalskalierte Variable als solche kennzeichnen lässt. Wir haben von vier Personen eine Angabe über die Höhe ihres Einkommens, und gehen von einer Dreifach-Stufung aus (hoch, mittel, niedrig). Hier die Daten:

```
c(1, 3, 2, 2)

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

Der Befehl in diesem Fall lautet ordered. Statt factor verwenden wir im Falle einer ordinalen Variable also den Befehl ordered(). Im Folgenden die "Umwandlung":

```
ordered(c(1, 3, 2, 2), labels = c("hoch", "mittel", "niedrig"))

## [1] hoch    niedrig mittel  mittel
## Levels: hoch < mittel < niedrig
```

Alternativ kann der Befehl factor() durch die Option ordered=TRUE ergänzt werden. Beides führt zu demselben Resultat.

1.1.4 Listen

Listen stellen eine geordnete Sammlung von Objekten dar. Listen können dabei unterschiedliche Objekttypen beinhalten. Es ist also z.B. möglich einer Liste einen numerischen Vektor, einen "character-Vektor" und eine Matrix zu übermitteln. Ein kleines Beispiel zur Veranschaulichung einer Liste in R:

[6] Auch viele metrisch skalierte Variablen haben nur endlich viele Ausprägungen (z.B. Items in einem Fragebogen); sie werden aber meist nicht als kategorial im engeren Sinne bezeichnet

[7] dem Befehl labels entspricht die Zuweisung von Wertelabels in SPSS

```
list(Personen = 3, Altersangaben = c(23, 26, 19), Augenfarbe = c("grün", "blau", "rot"))

## $Personen
## [1] 3
##
## $Altersangaben
## [1] 23 26 19
##
## $Augenfarbe
## [1] "grün" "blau" "rot"
```

1.1.5 Data Frames

Data Frames sind in gewisser Weise eine Erweiterung von Matrizen. Data Frames enthalten wie Matrizen verschiedene Spalten und Reihen, denen Namen gegeben werden können. Der Unterschied ist, dass die Spalten eines Data Frames verschiedene Modi (numeric, character, factor) darstellen können. Daher eignet sich dieser Objekttyp auch sehr gut zum Darstellen von Datensätzen, wie man sie üblicherweise nach einer Erhebung vorliegen hat. Bei einer klassischen Fragebogenuntersuchung bekommen die Spalten die Nummern des jeweiligen Items (z.B. Item1, Item2 u.s.w.) und die Reihen repräsentieren die Teilnehmer (Personen, die den Fragebogen beantwortet haben). Theoretisch ließe sich ein derartiger Datensatz auch in eine Matrix überführen[8], doch üblicherweise wird zusätzlich zu den Antworten zumindest noch das Geschlecht der Probanden interessieren. Wie wir bereits wissen, muss R mitgeteilt werden, dass es sich bei der Variable "Geschlecht" um eine nominale Variable handelt. Dies erreicht man durch das faktorisieren (siehe 1.1.3).

```
data.frame(Geschlecht = factor(c("w", "m", "w")), Item1 = c(3, 4, 1))

##   Geschlecht Item1
## 1          w     3
## 2          m     4
## 3          w     1
```

1.1.6 Indexing

In diesem Abschnitt geht es darum, wie einzelne oder mehrere Elemente aus den im vorherigen Kapitel beschriebenen Objekten ausgewählt werden können. Wir erstellen wir einen Vektor "x" mit den ersten fünf Buchstaben des Alphabets und greifen auf das dritte Element, sprich das "c" zu:

```
v <- c("A", "B", "C", "D", "E")
v[3]

## [1] "C"
```

Der Zugriff auf ein Element erfolgt durch die Angabe des Symbols (des Namens der Variable) gefolgt von einer eckigen Klammer, in welcher die Stelle des Elements innerhalb des Vektors angegeben wird.

Mehrere Elemente können wie folgt ausgewählt werden:

```
v[c(2, 4, 5)]

## [1] "B" "D" "E"
```

[8] Schließlich hätten wir in einem solchen Fall nur numerische Vektoren

```r
# bei aufeinanderfolgenden Elementen:
v[3:5]

## [1] "C" "D" "E"
```

Weiterhin ist es nicht nur möglich, bestimmte Elemente auszuwählen, sondern auch, bestimmte Elemente „auszuschließen“:

```r
v[-1]

## [1] "B" "C" "D" "E"

v[-1:-3]

## [1] "D" "E"

v[-c(2, 5)]

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

Denken Sie daran, dass sich die so ausgewählten Elemente ohne weiteres in neue Variablen abspeichern ließen. Alternativ wäre es natürlich auch möglich die Variable, aus welcher die Elemente ausgewählt werden, selbst zu verändern. Etwa so:

```r
(v <- v[c(1, 5)])

## [1] "A" "E"
```

Nachdem wir wissen, wie Elemente aus Vektoren extrahiert werden, wollen wir jetzt zeigen, wie auf Elemente aus Matrizen zugegriffen werden kann. Zur Erinnerung, eine Matrix sieht z.B. so aus:

```r
(m <- matrix(101:108, nrow = 2))

##      [,1] [,2] [,3] [,4]
## [1,]  101  103  105  107
## [2,]  102  104  106  108
```

Zugriff auf Element 3:

```r
m[3]

## [1] 103
```

Zugriff auf Element 7:

```r
m[7]

## [1] 107
```

Zugriff auf Element 3 bis 7:

```r
m[3:7]

## [1] 103 104 105 106 107
```

Zugriff auf den zweiten Wert in Spalte 3:

```
m[2, 3]

## [1] 106
```

Hier wird ersichtlich, dass die erste Angabe in der eckigen Klammer die Reihe bzw. Zeile betrifft, und die zweite Angabe (von der ersten durch ein Komma getrennt), die Spalte. Wie aber eine ganze Spalte oder eine ganze Reihe auswählen? Ganz einfach - hier der Beweis:

```
m[2, ]

## [1] 102 104 106 108

m[, 3]

## [1] 105 106
```

Was meinen Sie, wie lassen sich die Spalten 1,2 und 4 aus der Matrix „m“ auswählen?

```
m[, c(1, 2, 4)]

##      [,1] [,2] [,3]
## [1,]  101  103  107
## [2,]  102  104  108
```

Vektoren, Matrizen, jetzt fehlen uns noch Listen und Data Frames[9]. Weiter geht es mit Listen.

```
(l <- list(Zahl = 5, Vektor = c(1, 2, 3), String = "Das ist ein String"))

## $Zahl
## [1] 5
##
## $Vektor
## [1] 1 2 3
##
## $String
## [1] "Das ist ein String"
```

Sie werden gleich sehen, dass es viele Parallelen zu dem Zugriff auf Elemente in Vektoren und Matrizen gibt. Allerdings auch einige kleine „Neuheiten“. Jedes Element der Liste besitzt einen Namen (hier: Zahl, Vektor und String). Wir können auf einzelne Elemente durch Angabe der Liste und des Namens, getrennt durch ein $-Zeichen, zugreifen. Im Folgenden auf das Element mit dem Namen „Vektor“, also den Vektor c(1,2,3):

```
l$Vektor

## [1] 1 2 3
```

Alternativ können dasselbe durch eine doppelte eckige Klammer erreichen:

```
l[[2]]

## [1] 1 2 3
```

Zugriff auf „Zahl“ und „String“:

[9] und wie ist es bei Faktoren? - Der Zugriff ist äquivalent zu dem der Vektoren

```
l[c("Zahl", "String")]

## $Zahl
## [1] 5
##
## $String
## [1] "Das ist ein String"
```

Zugriff auf Element 2 des Elements 2 (auf die Zahl „2“ aus dem Element „Vektor“):

```
l$Vektor[2]

## [1] 2

# oder so:
l[[c(2, 2)]]

## [1] 2
```

Zuletzt noch etwas zu Data Frames. Die Zugriffsoptionen unterscheiden sich nicht wesentlich von denen der Matrizen. Lediglich lassen sich bei Data Frames einzelne Spalten (wie bei den Listen) durch das $-Zeichen und den Namen der jeweiligen Spalte aufrufen. Ansonsten funktioniert der Zugriff identisch.