Programmieren mit R

Wasilios Hariskos

14 November 2019

Um später Datenanalysen durchführen zu können, haben wir in diesem Modul wichtige Datenstrukturen (Objekte) wie Vektoren, Matrizen, Faktoren, Datensätze und Listen kennengelernt.

Das angefertigte R Skript gibt es [hier](https://github.com/wasilios-hariskos/r4ds-erfurt19/blob/master/00_script.R).

# Einführung in die Basics

Wir haben gelernt wie wir die Konsole von R als Taschenrechner benutzen können und wie wir einer Variablen ein Objekt wie eine Zahl zuordnen können.

# Vektoren

Wir haben gelernt wie wir

* Vektoren in R erstellen können,
* diese benennen können,
* Elemente auswählen können und
* zwei Vektoren vergleichen können.

Ein Vektor kann in R mit der “combine” Funktion c() erstellt werden.

Wir haben weitere Funktionen kennengelernt:

* names()
* sum()
* class()

Mit der Funktion help() können wir mehr über eine Funktion herausfinden.

Für die Zuweisung eines Objekts zu einer Variablen haben wir den Operator <-benutzt. Es folgen drei unterschiedliche Arten und Weisen, um eine Variable auszugeben:

a <- 4  
  
print(a)

## [1] 4

a

## [1] 4

(b <- 5)

## [1] 5

Die Eingabe des Vektors c(1, 2, 3, 4, 5) können wir mit 1:5 abkürzen. Die zweite Schreibweise ist besonders nützlich, wenn der Vektor lang ist.

Beispiel 1:

1:100

## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17  
## [18] 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34  
## [35] 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51  
## [52] 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68  
## [69] 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85  
## [86] 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

Beispiel 2:

200:101

## [1] 200 199 198 197 196 195 194 193 192 191 190 189 188 187 186 185 184  
## [18] 183 182 181 180 179 178 177 176 175 174 173 172 171 170 169 168 167  
## [35] 166 165 164 163 162 161 160 159 158 157 156 155 154 153 152 151 150  
## [52] 149 148 147 146 145 144 143 142 141 140 139 138 137 136 135 134 133  
## [69] 132 131 130 129 128 127 126 125 124 123 122 121 120 119 118 117 116  
## [86] 115 114 113 112 111 110 109 108 107 106 105 104 103 102 101

Beispiel 3:

-100:100

## [1] -100 -99 -98 -97 -96 -95 -94 -93 -92 -91 -90 -89 -88 -87  
## [15] -86 -85 -84 -83 -82 -81 -80 -79 -78 -77 -76 -75 -74 -73  
## [29] -72 -71 -70 -69 -68 -67 -66 -65 -64 -63 -62 -61 -60 -59  
## [43] -58 -57 -56 -55 -54 -53 -52 -51 -50 -49 -48 -47 -46 -45  
## [57] -44 -43 -42 -41 -40 -39 -38 -37 -36 -35 -34 -33 -32 -31  
## [71] -30 -29 -28 -27 -26 -25 -24 -23 -22 -21 -20 -19 -18 -17  
## [85] -16 -15 -14 -13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3  
## [99] -2 -1 0 1 2 3 4 5 6 7 8 9 10 11  
## [113] 12 13 14 15 16 17 18 19 20 21 22 23 24 25  
## [127] 26 27 28 29 30 31 32 33 34 35 36 37 38 39  
## [141] 40 41 42 43 44 45 46 47 48 49 50 51 52 53  
## [155] 54 55 56 57 58 59 60 61 62 63 64 65 66 67  
## [169] 68 69 70 71 72 73 74 75 76 77 78 79 80 81  
## [183] 82 83 84 85 86 87 88 89 90 91 92 93 94 95  
## [197] 96 97 98 99 100

# Matrizen

Wir haben gelernt wie wir in R Matrizen erstellen können und wir haben verstanden wie man grundlegende Berechnungen mit diesen durchführen kann.

Eine Matrix haben wir definiert als eine (zwei-dimensionale) Kollektion von Elementen (des gleichen Datentyps, zum Beispiel “numeric”, “logical” oder “character”), welche in einer festen Anzahl an Zeilen und Spalten angeordnet sind.

Eine Matrix kann in R mit der Funktion matrix() erstellt werden.

Wir haben weitere Funktionen kennengelernt:

* rownames() und colnames()
* rowSums() und colSums()
* rbind() und cbind()

Wenn wir mehr über eine Funktion wissen wollen, können wir statt help(matrix) auch ?matrix schreiben.

# Faktoren

Sehr oft können Daten einer endlichen Anzahl von Kategorien zugeordnet werden. Zum Beispiel, das Geschlecht einer Versuchsperson kann als männlich oder weiblich kategorisiert werden.

In R werden kategoriale Daten als “factor” gespeichert. Wir haben gelernt wie wir ungeordnete Faktoren (wie Geschlecht) sowie geordnete Faktoren (wie Temperatur) erstellen und vergleichen können.

Ein Faktor kann in R mit der Funktion factor() erstellt werden.

Wir haben zudem folgende Funktionen kennengelernt:

* levels()
* summary()

# Data Frames

Die meisten Datensätze werden in R in Data Frames gespeichert. Wir haben gelernt wie man

* Data Frame erstellt,
* Teile des Data Frames auswählt und
* den Data Frame nach einer Variablen ordnet.

In einem Data Frame sind die Variablen eines Datensatzes als Spalten und die Beobachtugen als Zeilen angeordnet.

Spalten korrepondieren zu Fragen:

* “Logical”: Bist du verheiratet? oder andere Ja/Nein Fragen
* “Numeric”: Wie alt bist du?
* “Character”: Was hälst du von diesem Produkt? oder andere offene Fragen

Wichtig: Im Vergleich zu einer Matrix müssen die Elemente eines Data Frames nicht vom selben Typ sein (nur die Spalten müssen vom selben Typ sein).

Wir haben zwei Datensätze, die schon in R gespeichert sind, genutzt:

* mtcars
* airquality

Dabei war die Hilfefunktion help() oder? nützlich, um mehr Infos über die Datensätze zu erhalten.

Ein neuer Data Frame kann in R mit der Funktion data.frame() erstellt werden.

Wir haben zudem folgende Funktionen kennengelernt:

* head() und tail()
* str()
* subset()
* order()

# Listen

Listen, können im Vergleich zu Vektoren, Elemente unterschiedlichen Typs enthalten. Wir haben gelernt wie man eine Liste erstellt, benennt und ein weiteres Element hinzufügt.

Eine Liste kann in R mit der Funktion list() erstellt werden.

# Operatoren

|  |  |
| --- | --- |
| Operator | Beschreibung |
| +, - | Addition, Subtraktion |
| \*, ^ oder \*\*, %\*% | Multiplikation, Potenz, Matrizenmultiplikation |
| /, %/%, %% | Division, Ganzzahlige Division, Modulo Division |
|  |  |
| ==, != | gleich, ungleich |
| >, >= | größer, größer gleich |
| <, <= | kleiner, kleiner gleich |
|  |  |
| ! | nicht (Negation) |
| &, && | und |
| |, || | oder |
| xor() | entweder oder (ausschließend) |
|  |  |

## Was passiert in diesen Fällen?

"r" == "R" # R unterscheidet zwischen Groß- und Kleinschreibung

## [1] FALSE

"Guten Morgen" < "Guten Tag" # lexikographisch

## [1] TRUE

FALSE < TRUE # wird als 0 < 1 interpretiert

## [1] TRUE

## Die Negation

!TRUE

## [1] FALSE

!FALSE

## [1] TRUE

## Das logische UND

Nur WAHR, wenn beide Aussagen wahr sind; sonst FALSCH.

TRUE & TRUE

## [1] TRUE

TRUE & FALSE

## [1] FALSE

FALSE & TRUE

## [1] FALSE

FALSE & FALSE

## [1] FALSE

## Das logische ODER (einschließend)

Nur WAHR, wenn mindestens eine Aussage WAHR ist; sonst FALSCH.

TRUE | TRUE

## [1] TRUE

TRUE | FALSE

## [1] TRUE

FALSE | TRUE

## [1] TRUE

FALSE | FALSE

## [1] FALSE

## Das logische ENTWEDER ODER (ausschließend)

Nur WAHR, wenn genau eine Aussage WAHR ist; sonst FALSCH.

xor(TRUE, TRUE)

## [1] FALSE

xor(TRUE, FALSE)

## [1] TRUE

xor(FALSE, TRUE)

## [1] TRUE

xor(FALSE, FALSE)

## [1] FALSE

## Vektorwertiges &, | vs nicht vektorwertiges &&, ||

FALSE && TRUE # nur der erste Ausdruck wird ausgewertet

## [1] FALSE

TRUE && FALSE # auch der zweite Ausdruck wird ausgewertet

## [1] FALSE

c(FALSE, FALSE, TRUE) & c(FALSE, TRUE, TRUE)

## [1] FALSE FALSE TRUE

c(FALSE, FALSE, TRUE) && c(FALSE, TRUE, TRUE)

## [1] FALSE

!c(FALSE, FALSE, TRUE) & !c(FALSE, TRUE, TRUE)

## [1] TRUE FALSE FALSE

!c(FALSE, FALSE, TRUE) && !c(FALSE, TRUE, TRUE)

## [1] TRUE

c(FALSE, FALSE, TRUE) | c(FALSE, TRUE, TRUE)

## [1] FALSE TRUE TRUE

c(FALSE, FALSE, TRUE) || c(FALSE, TRUE, TRUE)

## [1] FALSE

c(TRUE, FALSE, TRUE) | c(FALSE, TRUE, TRUE)

## [1] TRUE TRUE TRUE

c(TRUE, FALSE, TRUE) || c(FALSE, TRUE, TRUE)

## [1] TRUE

# Bedingte Anweisungen (Conditional Statements)

Beispiel 1 (if)

x <- -3 # [change to 5 and run again, set back to -3]  
  
if (x < 0) { # Bedingung  
 print("x ist eine negative Zahl") # Ausdruck  
}

## [1] "x ist eine negative Zahl"

Beispiel 2 (if, else)

x <- -3 # [change to 5 and run again, set back to -3]  
  
if (x < 0) { # Bedingung  
 print("x ist eine negative Zahl") # Ausdruck1  
} else {  
 print("x ist eine nicht-negative Zahl") # Ausdruck2  
}

## [1] "x ist eine negative Zahl"

Beispiel 3 (if, else if, else)

x <- -3 # [change to 0 and 5 and run again, set back to -3]  
  
if (x < 0) { # Bedingung1  
 print("x ist eine negative Zahl") # Ausdruck1  
} else if (x == 0) { # Bedingung2  
 print("x ist gleich Null") # Ausdruck2  
 } else {  
 print("x ist eine nicht-negative Zahl") # Ausdruck3  
}

## [1] "x ist eine negative Zahl"

# Schleifen (loops)

## while loop

Beispiel 1 (while loop)

count <- 1  
  
while (count <= 3) {   
 print(paste("Der Count ist gleich", count))  
 count <- count + 1  
}

## [1] "Der Count ist gleich 1"  
## [1] "Der Count ist gleich 2"  
## [1] "Der Count ist gleich 3"

Beispiel 2 (while loop with break)

count <- 1  
  
while (count <= 3) {  
 if (count == 2) {  
 break  
 }  
   
 print(paste("Der Count ist gleich", count))  
 count <- count + 1  
}

## [1] "Der Count ist gleich 1"

## for loop

Beispiel 1 (for loop over vector)

new\_states <- c("Brandenburg",  
 "Mecklenburg-Vorpommern",  
 "Sachsen",  
 "Sachsen-Anhalt",  
 "Thüringen")  
  
for (state in new\_states) {  
 print(state)  
}

## [1] "Brandenburg"  
## [1] "Mecklenburg-Vorpommern"  
## [1] "Sachsen"  
## [1] "Sachsen-Anhalt"  
## [1] "Thüringen"

Beispiel 2 (for loop over list)

new\_states <- list("Brandenburg",  
 "Mecklenburg-Vorpommern",  
 "Sachsen",  
 "Sachsen-Anhalt",  
 "Thüringen")  
  
for (state in new\_states) {  
 print(state)  
}

## [1] "Brandenburg"  
## [1] "Mecklenburg-Vorpommern"  
## [1] "Sachsen"  
## [1] "Sachsen-Anhalt"  
## [1] "Thüringen"

Beispiel 3 (for loop with break)

new\_states <- list("Brandenburg",  
 "Mecklenburg-Vorpommern",  
 "Sachsen",  
 "Sachsen-Anhalt",  
 "Thüringen")  
  
for (state in new\_states) {  
 if (nchar(state) < 10) {  
 break  
 }  
   
 print(state)  
}

## [1] "Brandenburg"  
## [1] "Mecklenburg-Vorpommern"

Beispiel 4 (for loop with next)

new\_states <- list("Brandenburg",  
 "Mecklenburg-Vorpommern",  
 "Sachsen",  
 "Sachsen-Anhalt",  
 "Thüringen")  
  
for (state in new\_states) {  
 if (nchar(state) < 10) {  
 next  
 }  
   
 print(state)  
}

## [1] "Brandenburg"  
## [1] "Mecklenburg-Vorpommern"  
## [1] "Sachsen-Anhalt"

Beispiel 5 (for loop with index)

new\_states <- c("Brandenburg",  
 "Mecklenburg-Vorpommern",  
 "Sachsen",  
 "Sachsen-Anhalt",  
 "Thüringen")  
  
for (i in 1:length(new\_states)) {  
 print(paste(new\_states[i],  
 "ist auf Position",  
 i,  
 "im new\_states Vektor")  
 )  
}

## [1] "Brandenburg ist auf Position 1 im new\_states Vektor"  
## [1] "Mecklenburg-Vorpommern ist auf Position 2 im new\_states Vektor"  
## [1] "Sachsen ist auf Position 3 im new\_states Vektor"  
## [1] "Sachsen-Anhalt ist auf Position 4 im new\_states Vektor"  
## [1] "Thüringen ist auf Position 5 im new\_states Vektor"

# Funktionen

## Eine Funktion in R aufrufen

grades <- c(1, 4, 2, 2, 3, 1, NA) # die letzte Note steht nicht fest  
  
mean\_grades <- mean(grades) # mittelwert berechnen  
  
mean\_grades # mittelwert ausgeben

## [1] NA

## Die Dokumentation einer Funktion aufrufen

help(mean)

## starting httpd help server ... done

?NA

## Wie kann man den Mittelwert bei fehlenden Werten berechnen?

grades <- c(1, 4, 2, 2, 3, 1, NA) # die letzte Note steht nicht fest  
  
mean\_grades <- mean(x = grades,  
 na.rm = TRUE) # mittelwert ohne NAs berechnen  
  
mean\_grades # mittelwert ausgeben

## [1] 2.166667

## Eine Funktion selber schreiben

grades <- c(1, 4, 2, 2, 3, 1)  
  
mittelwert <- function(x) {  
 n <- length(x)  
 total <- sum(x)  
 total / n  
}  
  
mittelwert(grades)

## [1] 2.166667

## Eine selbstgeschriebene Funktion verbessern

grades <- c(1, 4, 2, 2, 3, 1, NA)  
  
help(NA) # go to see also, try na.omit  
  
mittelwert <- function(x) {  
 x <- na.omit(x) # position matters  
 n <- length(x)  
 total <- sum(x)  
 return(total / n)  
}  
  
mittelwert(grades)

## [1] 2.166667