Einführung in R

Clemens Brunner

14.-15.2.2019

Tabellarische Daten

Vektoren werden in R verwendet, um eindimensionale Daten abzubilden. Häufig sind Daten aber zweidimensional strukturiert, also in Form einer Tabelle. Für tabellarische Daten gibt es in R zwei gängige Datentypen, nämlich Matrizen (Einzahl Matrix) und sogenannte Data Frames. Matrizen können genau wie Vektoren nur Elemente eines einzigen Datentyps enthalten (sie sind also homogene Datentypen). Im Gegensatz dazu können Data Frames unterschiedliche Datentypen enthalten (jede Spalte kann ein anderer Datentyp sein).

Matrizen

Zusammenhang mit Vektoren

Matrizen sind lediglich Vektoren mit einem speziellen Dimensionsattribut, welches man mit der Funktion `dim` abfragen und setzen kann.

```
v <- 1:20
v
```

```
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
dim(v)
```

```
NULL
```

```
length(v)
```

```
[1] 20
```

Ein Vektor hat kein Dimensionsattribut, daher wird hier `NULL` angezeigt (sehr wohl hat ein Vektor aber eine Länge, was der Anzahl der Elemente entspricht).

Nun kann man für einen Vektor das Dimensionsattribut auf die gewünschte Anzahl an Zeilen und Spalten setzen (das Produkt von Zeilen und Spalten muss mit der Gesamtanzahl an Elementen im Vektor übereinstimmen):

```
dim(v) <- c(4, 5)  # 4 Zeilen, 5 Spalten
dim(v)
```

```
[1] 4 5
```

```
attributes(v)
```

```
$dim
[1] 4 5
```

```
v
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

```
class(v)
```

```
[1] "matrix"
```

Dieses Beispiel zeigt, dass sich die zugrundeliegenden Daten nicht ändern - sie werden jediglich anders dargestellt bzw. interpretiert.

Erstellen von Matrizen

Eine Matrix kann nicht nur aus einem bereits vorhandenen Vektor, sondern direkt mit der Funktion `matrix` erzeugt werden:

```
m <- matrix(1:20, 4, 5)
m
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

Das erste Argument sind die Daten (ein Vektor), die man in die Matrix schreiben möchte. Das zweite Argument ist die Anzahl der Zeilen, und das dritte Argument ist die Anzahl der Spalten der Matrix. Wie man sieht, werden die Daten spaltenweise in der Matrix angeordnet. Möchte man die Daten zeilenweise anordnen, kann man das Argument `byrow=TRUE` setzen, also im Beispiel:

```
matrix(1:20, 4, 5, byrow=TRUE)
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
```

Benennen von Zeilen und Spalten

So wie man einen Vektor mit benannten Elementen erstellen kann, kann man in einer Matrix Zeilen- und Spaltennamen vergeben:

```
rownames(m) <- c(21, 13, 37, 45)
colnames(m) <- c("A", "B", "C", "D", "E")
m
```

```
   A B  C  D  E
21 1 5  9 13 17
13 2 6 10 14 18
37 3 7 11 15 19
45 4 8 12 16 20
```

Indizieren

Das Herausgreifen einzelner Elemente funktioniert im Prinzip wie bei Vektoren. Der Hauptunterschied ist, dass man bei Matrizen zwei Indizes innerhalb der eckigen Klammern angibt: der erste entspricht den Zeilen und der zweite entspricht den Spalten. Wenn der erste Index weggelassen wird, werden alle Zeilen herausgegriffen. Wenn der zweite Index weggelassen wird, werden alle Spalten herausgegriffen. Zu beachten ist jedoch, dass das zur Trennung beider Indizes verwendete Komma verwendet werden sollte - auch wenn ein Index weggelassen wird. Die folgenden Beispiele illustrieren dies anschaulich.

```
m[1, 4]  # 1. Zeile, 4. Spalte
```

```
[1] 13
```

```
m[, 3]  # 3. Spalte
```

```
21 13 37 45
 9 10 11 12
```

```
m[3,]  # 3. Zeile
```

```
 A  B  C  D  E
 3  7 11 15 19
```

```
m[c(2, 4),]  # 2. und 4. Zeile
```

```
   A B  C  D  E
13 2 6 10 14 18
45 4 8 12 16 20
```

```
m[c(1, 3), c(1, 2, 5)]
```

```
   A B  E
21 1 5 17
37 3 7 19
```

```
m[, "C"]  # Spalte C
```

```
21 13 37 45
 9 10 11 12
```

```
m[m[, "A"] > 2,]  # Zeilen, in denen die Spalte A > 2 ist
```

```
   A B  C  D  E
37 3 7 11 15 19
45 4 8 12 16 20
```

Zwang (Coercion)

Wenn man jetzt z.B. eine neue Spalte vom Typ `character` hinzufügen möchte, dann funktioniert das mit einer Matrix nicht wie gewünscht, denn die numerischen Elemente werden automatisch in Zeichenketten "gezwungen".

```
subjects <- c("Hans", "Birgit", "Ferdinand", "Johanna")
cbind(subjects, m)
```

```
   subjects    A   B   C    D    E
21 "Hans"      "1" "5" "9"  "13" "17"
13 "Birgit"    "2" "6" "10" "14" "18"
37 "Ferdinand" "3" "7" "11" "15" "19"
45 "Johanna"   "4" "8" "12" "16" "20"
```

Anhand dieses Beispiels sieht man auch eine weitere Möglichkeit, wie man Matrizen erstellen bzw. erweitern kann. Die Funktion `cbind` hängt Vektoren (oder Matrizen) spaltenweise zusammen, während analog dazu die Funktion `rbind` Objekte zeilenweise zusammenfügt.

Rechnen mit Matrizen

Genau wie bei Vektoren werden Rechenoperationen mit Matrizen elementweise durchgeführt. Zusätzlich gibt es noch zwei praktische Funktionen, mit denen man die Zeilen- bzw. Spaltensummen einer Matrix berechnen kann: `rowSums` und `colSums`.

```
rowSums(m)
```

```
21 13 37 45
45 50 55 60
```

```
colSums(m)
```

```
 A  B  C  D  E
10 26 42 58 74
```

Eine Matrix ist, wie man anhand der obigen Beispiele erkennt, eigentlich nur für rein numerische Daten geeignet. Oft will man aber auch nicht-numerische Spalten wie z.B. Namen oder Gruppenzugehörigkeit hinzufügen - dies funktioniert wie oben gezeigt mit Matrizen praktisch nicht.

Data Frames

Data Frames sind ebenso wie Matrizen zweidimensionale Datenstrukturen (sie bestehen aus Zeilen und Spalten). Im Gegensatz zu Matrizen können Spalten aber unterschiedliche Datentypen beinhalten (z.B. kann eine Spalte numerisch sein, eine andere Spalte kann Zeichenketten beinhalten, und so weiter). Innerhalb einer Spalte müssen aber alle Werte homogen sein. Man kann sich die Spalten in einem Data Frame daher als Vektoren vorstellen.

Erstellen von Data Frames

Mit der Funktion `data.frame` kann man ein Data Frame aus Vektoren und/oder Matrizen erzeugen (die einzelnen Argumente werden spaltenweise aneinandergehängt):

```
df <- data.frame(subjects, m)
df
```

```
    subjects A B  C  D  E
21      Hans 1 5  9 13 17
13    Birgit 2 6 10 14 18
37 Ferdinand 3 7 11 15 19
45   Johanna 4 8 12 16 20
```

Wie bei Matrizen kann man mit der Funktion `colnames` die Spaltennamen lesen bzw. setzen.

```
colnames(df)
```

```
[1] "subjects" "A"        "B"        "C"        "D"        "E"
```

```
colnames(df) <- c("patient", "age", "weight", "bp", "rating", "test")
df
```

```
     patient age weight bp rating test
21      Hans   1      5  9     13   17
13    Birgit   2      6 10     14   18
37 Ferdinand   3      7 11     15   19
45   Johanna   4      8 12     16   20
```

Oft möchte man ein Data Frame spaltenweise aus einzelnen Vektoren zusammensetzen. Auch dies ist mit der Funktion `data.frame` einfach möglich:

```
data.frame(x=1:5, id=c("X", "c1", "V", "RR", "7G"), value=c(12, 18, 19, 3, 8))
```

```
  x id value
1 1  X    12
2 2 c1    18
3 3  V    19
4 4 RR     3
5 5 7G     8
```

Hier ist hervorzuheben, dass die Spaltennamen automatisch auf die Argumentnamen gesetzt werden.

Anzeigen von Data Frames

Eine schnelle Übersicht über ein Data Frame bekommt man mit den Funktionen `str`, `head` und `tail`. Die Funktion `str` stellt die Struktur eines Objektes knapp zusammengefasst dar:

```
str(df)
```

```
'data.frame':   4 obs. of  6 variables:
 $ patient: Factor w/ 4 levels "Birgit","Ferdinand",..: 3 1 2 4
 $ age    : int  1 2 3 4
 $ weight : int  5 6 7 8
 $ bp     : int  9 10 11 12
 $ rating : int  13 14 15 16
 $ test   : int  17 18 19 20
```

Hier sieht man, dass der Datentyp der Spalte `patient` nicht wie vielleicht erwartet `character` ist, sondern `factor` (wir könnten diese Spalte aber mit der Funktion `as.character` in den gewünschten Datentyp umwandeln).

Die Funktion `head` gibt die ersten sechs Zeilen am Bildschirm aus, während `tail` die letzten sechs Zeilen ausgibt. Es gibt mit dem Argument `n` auch die Möglichkeit, die Anzahl der angezeigten Zeilen anzupassen.

```
l <- data.frame(a=rnorm(5000), b=rpois(5000, 2), x=rep(letters, length.out=5000))
head(l)
```

```
            a b x
1 -1.021039947 4 a
2 -0.278569499 0 b
3  0.793029181 2 c
4  0.449149889 2 d
5  1.274568672 2 e
6  0.007168437 3 f
```

```
tail(l, n=4)
```

```
              a b x
4997 -0.06128952 1 e
4998 -0.25784771 2 f
4999 -1.42996147 1 g
5000  0.74074565 1 h
```

```
str(l)
```

```
'data.frame':   5000 obs. of  3 variables:
 $ a: num  -1.021 -0.279 0.793 0.449 1.275 ...
 $ b: int  4 0 2 2 2 3 2 4 1 1 ...
 $ x: Factor w/ 26 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10 ...
```

Indizieren

Auf einzelne Spalten eines Data Frames kann man mit `$` gefolgt vom Spaltennamen zugreifen:

```
df$patient
```

```
[1] Hans      Birgit    Ferdinand Johanna
Levels: Birgit Ferdinand Hans Johanna
```

```
df$rating
```

```
[1] 13 14 15 16
```

Diese Schreibweise kann man auch anwenden, wenn man dem Data Frame eine neue Spalte hinzufügen will. Dazu gibt man einen Spaltennamen an, der noch nicht vorhanden ist und weist diesem einen entsprechenden Vektor zu:

```
df$new <- c("yes", "no", "no", "yes")
df
```

```
     patient age weight bp rating test new
21      Hans   1      5  9     13   17 yes
13    Birgit   2      6 10     14   18  no
37 Ferdinand   3      7 11     15   19  no
45   Johanna   4      8 12     16   20 yes
```

Alternativ kann man wie bei Matrizen auch `rbind` bzw. `cbind` zum Hinzufügen neuer Zeilen bzw. Spalten benutzen.

Eine Spalte aus einem Data Frame kann man löschen, indem man der Spalte den Wert `NULL` zuweist:

```
df$new <- NULL
df
```

```
     patient age weight bp rating test
21      Hans   1      5  9     13   17
13    Birgit   2      6 10     14   18
37 Ferdinand   3      7 11     15   19
45   Johanna   4      8 12     16   20
```

Wir können auch einzelne Spalten in einen anderen Datentyp umwandeln - z.B. können wir die `patient`-Spalte vom Typ `factor` in den Typ `character` konvertieren, indem wir diese Spalte mit neuen Werten überschreiben (R wertet immer zuerst die rechte Seite der Zuweisung aus und führt erst dann die Zuweisung durch):

```
df$patient <- as.character(df$patient)
str(df)
```

```
'data.frame':   4 obs. of  6 variables:
 $ patient: chr  "Hans" "Birgit" "Ferdinand" "Johanna"
 $ age    : int  1 2 3 4
 $ weight : int  5 6 7 8
 $ bp     : int  9 10 11 12
 $ rating : int  13 14 15 16
 $ test   : int  17 18 19 20
```

Zeilen und Spalten kann man auch per Indizierung mit eckigen Klammern herausgreifen. Dies funktioniert wie bei Matrizen. Die erste Zahl in den eckigen Klammern steht für die Zeile, die zweite Zahl für die Spalte. Wenn ein Index weggelassen wird, werden alle Zeilen bzw. Spalten herausgegriffen.

```
df[1,]
```

```
   patient age weight bp rating test
21    Hans   1      5  9     13   17
```

```
df[2:3,]
```

```
     patient age weight bp rating test
13    Birgit   2      6 10     14   18
37 Ferdinand   3      7 11     15   19
```

Spalten können somit mit dem zweiten Index indiziert werden.

```
df[, 1]
```

```
[1] "Hans"      "Birgit"    "Ferdinand" "Johanna"
```

```
df[, 4]
```

```
[1]  9 10 11 12
```

Spalten (oder auch Zeilen) kann man anstelle ihrer Indizes auch mit ihren Namen ansprechen:

```
df[, "patient"]
```

```
[1] "Hans"      "Birgit"    "Ferdinand" "Johanna"
```

```
df[, "bp"]
```

```
[1]  9 10 11 12
```

So kann auch gezielt ein bestimmter Bereich herausgegriffen werden:

```
df[1:2, c(1, 3:4)]
```

```
   patient weight bp
21    Hans      5  9
13  Birgit      6 10
```

Eine andere Möglichkeit bestimmte Zeilen auszuwählen bietet die Funktion `subset`:

```
subset(df, weight < 7)
```

```
   patient age weight bp rating test
21    Hans   1      5  9     13   17
13  Birgit   2      6 10     14   18
```

Diese Funktion funktioniert mit Vektoren, Matrizen und Data Frames und es ist manchmal übersichtlicher, anstelle von eckigen Klammern die Funktion `subset` zu benutzen.

Alternative: Tibbles

Mit den Bordmitteln von R (d.h. mit den eingebauten Datentypen wie Data Frames sowie dazugehörigen Funktionen) kann man hervorragend Daten analysieren. Manchmal sind diese Konstrukte aber umständlich zu verwenden, und daher bietet sich die in letzter Zeit immer populärer werdende Paketsammlung namens Tidyverse an, diverse Dinge zu modernisieren bzw. zu vereinfachen. Man installiert alle notwendigen Pakete aus dem Tidyverse mit dem Paket `tidyverse`. Darin enthalten ist das Paket `tibble`, welches eine moderne Alternative zu Data Frames darstellt. Insbesondere ist die Konvertierung/Erstellung von Tibbles nachvollziehbarer, und die Darstellung von Tibbles am Bildschirm ist wesentlich übersichtlicher. Überall wo man Data Frames verwenden kann, kann man auch Tibbles benutzen (Tibbles sind eigentlich nichts anderes als erweiterte/verbesserte Data Frames).

Bevor man Tibbles verwenden kann, muss man entweder `tibble` oder `tidyverse` aktivieren. Letzteres aktiviert gleich alle Pakete aus dem Tidyverse.

```
library(tibble)
```

Mit dem Befehl `tibble` kann man nun ein neues Tibble erstellen, ganz analog zur Funktion `data.frame`:

```
t <- tibble(subjects=c("Hans", "Birgit", "Ferdinand", "Johanna"), A=1:4, B=5:8, C=9:12, D=13:16, E=17:2
t
```

```
# A tibble: 4 x 6
  subjects      A     B     C     D     E
```

```
  <chr>     <int> <int> <int> <int> <int>
1 Hans          1     5     9    13    17
2 Birgit        2     6    10    14    18
3 Ferdinand     3     7    11    15    19
4 Johanna       4     8    12    16    20
```

Wenn man ein Tibble am Bildschirm ausgibt, werden für jede Spalte automatisch die Datentypen angeführt. Eine weitere Verbesserung gegenüber Data Frames ist, dass Spalten mit Character-Vektoren nicht automatisch in Faktoren konvertiert werden, sondern Character-Spalten bleiben. Man erspart sich in diesen Fällen die Umwandlung mittels `as.character`.

Bei längeren Tabellen zeigt sich ein weiterer Vorteil von Tibbles. Gibt man große Tibbles am Bildschirm aus, wird die Darstellung automatisch so angepasst, dass nicht alle Daten ausgegeben werden, sondern nur so viel wie möglich um noch einen guten Überblick über die Daten zu gewährleisten. Dies sieht man beispielsweise am Datensatz `iris`, welcher als Data Frame automatisch in R vorhanden ist:

```
iris
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
1            5.1         3.5          1.4         0.2     setosa
2            4.9         3.0          1.4         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
4            4.6         3.1          1.5         0.2     setosa
5            5.0         3.6          1.4         0.2     setosa
6            5.4         3.9          1.7         0.4     setosa
7            4.6         3.4          1.4         0.3     setosa
8            5.0         3.4          1.5         0.2     setosa
9            4.4         2.9          1.4         0.2     setosa
10           4.9         3.1          1.5         0.1     setosa
11           5.4         3.7          1.5         0.2     setosa
12           4.8         3.4          1.6         0.2     setosa
13           4.8         3.0          1.4         0.1     setosa
14           4.3         3.0          1.1         0.1     setosa
15           5.8         4.0          1.2         0.2     setosa
16           5.7         4.4          1.5         0.4     setosa
17           5.4         3.9          1.3         0.4     setosa
18           5.1         3.5          1.4         0.3     setosa
19           5.7         3.8          1.7         0.3     setosa
20           5.1         3.8          1.5         0.3     setosa
21           5.4         3.4          1.7         0.2     setosa
22           5.1         3.7          1.5         0.4     setosa
23           4.6         3.6          1.0         0.2     setosa
24           5.1         3.3          1.7         0.5     setosa
25           4.8         3.4          1.9         0.2     setosa
26           5.0         3.0          1.6         0.2     setosa
27           5.0         3.4          1.6         0.4     setosa
28           5.2         3.5          1.5         0.2     setosa
29           5.2         3.4          1.4         0.2     setosa
30           4.7         3.2          1.6         0.2     setosa
31           4.8         3.1          1.6         0.2     setosa
32           5.4         3.4          1.5         0.4     setosa
33           5.2         4.1          1.5         0.1     setosa
34           5.5         4.2          1.4         0.2     setosa
35           4.9         3.1          1.5         0.2     setosa
36           5.0         3.2          1.2         0.2     setosa
37           5.5         3.5          1.3         0.2     setosa
```

38	4.9	3.6	1.4	0.1	setosa
39	4.4	3.0	1.3	0.2	setosa
40	5.1	3.4	1.5	0.2	setosa
41	5.0	3.5	1.3	0.3	setosa
42	4.5	2.3	1.3	0.3	setosa
43	4.4	3.2	1.3	0.2	setosa
44	5.0	3.5	1.6	0.6	setosa
45	5.1	3.8	1.9	0.4	setosa
46	4.8	3.0	1.4	0.3	setosa
47	5.1	3.8	1.6	0.2	setosa
48	4.6	3.2	1.4	0.2	setosa
49	5.3	3.7	1.5	0.2	setosa
50	5.0	3.3	1.4	0.2	setosa
51	7.0	3.2	4.7	1.4	versicolor
52	6.4	3.2	4.5	1.5	versicolor
53	6.9	3.1	4.9	1.5	versicolor
54	5.5	2.3	4.0	1.3	versicolor
55	6.5	2.8	4.6	1.5	versicolor
56	5.7	2.8	4.5	1.3	versicolor
57	6.3	3.3	4.7	1.6	versicolor
58	4.9	2.4	3.3	1.0	versicolor
59	6.6	2.9	4.6	1.3	versicolor
60	5.2	2.7	3.9	1.4	versicolor
61	5.0	2.0	3.5	1.0	versicolor
62	5.9	3.0	4.2	1.5	versicolor
63	6.0	2.2	4.0	1.0	versicolor
64	6.1	2.9	4.7	1.4	versicolor
65	5.6	2.9	3.6	1.3	versicolor
66	6.7	3.1	4.4	1.4	versicolor
67	5.6	3.0	4.5	1.5	versicolor
68	5.8	2.7	4.1	1.0	versicolor
69	6.2	2.2	4.5	1.5	versicolor
70	5.6	2.5	3.9	1.1	versicolor
71	5.9	3.2	4.8	1.8	versicolor
72	6.1	2.8	4.0	1.3	versicolor
73	6.3	2.5	4.9	1.5	versicolor
74	6.1	2.8	4.7	1.2	versicolor
75	6.4	2.9	4.3	1.3	versicolor
76	6.6	3.0	4.4	1.4	versicolor
77	6.8	2.8	4.8	1.4	versicolor
78	6.7	3.0	5.0	1.7	versicolor
79	6.0	2.9	4.5	1.5	versicolor
80	5.7	2.6	3.5	1.0	versicolor
81	5.5	2.4	3.8	1.1	versicolor
82	5.5	2.4	3.7	1.0	versicolor
83	5.8	2.7	3.9	1.2	versicolor
84	6.0	2.7	5.1	1.6	versicolor
85	5.4	3.0	4.5	1.5	versicolor
86	6.0	3.4	4.5	1.6	versicolor
87	6.7	3.1	4.7	1.5	versicolor
88	6.3	2.3	4.4	1.3	versicolor
89	5.6	3.0	4.1	1.3	versicolor
90	5.5	2.5	4.0	1.3	versicolor
91	5.5	2.6	4.4	1.2	versicolor

92	6.1	3.0	4.6	1.4	versicolor
93	5.8	2.6	4.0	1.2	versicolor
94	5.0	2.3	3.3	1.0	versicolor
95	5.6	2.7	4.2	1.3	versicolor
96	5.7	3.0	4.2	1.2	versicolor
97	5.7	2.9	4.2	1.3	versicolor
98	6.2	2.9	4.3	1.3	versicolor
99	5.1	2.5	3.0	1.1	versicolor
100	5.7	2.8	4.1	1.3	versicolor
101	6.3	3.3	6.0	2.5	virginica
102	5.8	2.7	5.1	1.9	virginica
103	7.1	3.0	5.9	2.1	virginica
104	6.3	2.9	5.6	1.8	virginica
105	6.5	3.0	5.8	2.2	virginica
106	7.6	3.0	6.6	2.1	virginica
107	4.9	2.5	4.5	1.7	virginica
108	7.3	2.9	6.3	1.8	virginica
109	6.7	2.5	5.8	1.8	virginica
110	7.2	3.6	6.1	2.5	virginica
111	6.5	3.2	5.1	2.0	virginica
112	6.4	2.7	5.3	1.9	virginica
113	6.8	3.0	5.5	2.1	virginica
114	5.7	2.5	5.0	2.0	virginica
115	5.8	2.8	5.1	2.4	virginica
116	6.4	3.2	5.3	2.3	virginica
117	6.5	3.0	5.5	1.8	virginica
118	7.7	3.8	6.7	2.2	virginica
119	7.7	2.6	6.9	2.3	virginica
120	6.0	2.2	5.0	1.5	virginica
121	6.9	3.2	5.7	2.3	virginica
122	5.6	2.8	4.9	2.0	virginica
123	7.7	2.8	6.7	2.0	virginica
124	6.3	2.7	4.9	1.8	virginica
125	6.7	3.3	5.7	2.1	virginica
126	7.2	3.2	6.0	1.8	virginica
127	6.2	2.8	4.8	1.8	virginica
128	6.1	3.0	4.9	1.8	virginica
129	6.4	2.8	5.6	2.1	virginica
130	7.2	3.0	5.8	1.6	virginica
131	7.4	2.8	6.1	1.9	virginica
132	7.9	3.8	6.4	2.0	virginica
133	6.4	2.8	5.6	2.2	virginica
134	6.3	2.8	5.1	1.5	virginica
135	6.1	2.6	5.6	1.4	virginica
136	7.7	3.0	6.1	2.3	virginica
137	6.3	3.4	5.6	2.4	virginica
138	6.4	3.1	5.5	1.8	virginica
139	6.0	3.0	4.8	1.8	virginica
140	6.9	3.1	5.4	2.1	virginica
141	6.7	3.1	5.6	2.4	virginica
142	6.9	3.1	5.1	2.3	virginica
143	5.8	2.7	5.1	1.9	virginica
144	6.8	3.2	5.9	2.3	virginica
145	6.7	3.3	5.7	2.5	virginica

```
146          6.7         3.0          5.2         2.3  virginica
147          6.3         2.5          5.0         1.9  virginica
148          6.5         3.0          5.2         2.0  virginica
149          6.2         3.4          5.4         2.3  virginica
150          5.9         3.0          5.1         1.8  virginica
```

Diese Darstellung von allen Zeilen ist alles andere als übersichtlich, deswegen lässt man sich hier am besten eine Zusammenfassung mittels `str`, `head` oder `tail` ausgeben. Bei Tibbles ist das nicht notwendig (die Funktion `as_tibble` wandelt vorhandene Data Frames in Tibbles um):

```
iris_tibble <- as_tibble(iris)
iris_tibble
```

```
# A tibble: 150 x 5
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          <dbl>       <dbl>        <dbl>       <dbl> <fct>
 1          5.1         3.5          1.4         0.2 setosa
 2          4.9         3            1.4         0.2 setosa
 3          4.7         3.2          1.3         0.2 setosa
 4          4.6         3.1          1.5         0.2 setosa
 5          5           3.6          1.4         0.2 setosa
 6          5.4         3.9          1.7         0.4 setosa
 7          4.6         3.4          1.4         0.3 setosa
 8          5           3.4          1.5         0.2 setosa
 9          4.4         2.9          1.4         0.2 setosa
10          4.9         3.1          1.5         0.1 setosa
# ... with 140 more rows
```

Übungen

Übung 1

Erstellen Sie einen Vektor `u` mit den ganzen Zahlen von 50 bis 99 und einen Vektor `v` mit den ganzen Zahlen von 0 bis -49. Wandeln Sie dann beide Vektoren in Matrizen mit jeweils 10 Zeilen um. Fügen Sie anschließend `u` und `v` (in dieser Reihenfolge) spaltenweise zusammen und speichern Sie das Ergebnis in der Variable `r` ab. Geben Sie dann `r` am Bildschirm aus. Welche Klasse hat das Objekt `r`?

Übung 2

Beantworten Sie folgende Fragen zur Matrix `r` aus Übung 1:

- Welche Dimension hat `r`?
- Wie viele Elemente beinhaltet `r` insgesamt?
- Wie lautet das Element in der 7. Zeile und 9. Spalte?
- Wie lauten die Zeilenmittelwerte bzw. die Spaltenmittelwerte?
- Wie lautet der Mittelwert der Elemente in den Zeilen 3-7 und Spalten 4-6?

Übung 3

Erstellen Sie ein Data Frame `df` mit 10 Zeilen und 3 Spalten wie folgt:

- Die erste Spalte soll `name` heißen und die Werte Ben, Emma, Luis, Mia, Paul, Hanna, Lukas, Sophia, Jonas und Emilia beinhalten.
- Die zweite Spalte `gender` soll das Geschlecht der Personen beinhalten, d.h. entweder den Buchstaben `m` oder `f`.
- Die dritte Spalte `value` soll 10 beliebige Zahlen zwischen 1 und 100 beinhalten.

Geben Sie abschließend `df` am Bildschirm aus. Welche Datentypen haben die drei Spalten?

Übung 4

Erstellen Sie ein Tibble mit den Spalten wie in Übung 3 beschrieben (verwenden Sie dazu die Funktion `tibble` aus dem Paket `tibble`). Speichern Sie dieses Tibble in der Variable `tf` ab und geben Sie es am Bildschirm aus. Unterscheiden sich die Spaltentypen des Tibbles von jenen des Data Frames?

Übung 5

Wandeln Sie das in Übung 3 erstellte Data Frame `df` in ein Tibble um.

Übung 6

Erstellen Sie ein neues Data Frame `df_f`, welches die Zeilen aller weiblichen Personen aus `df` enthält, und geben Sie dieses am Bildschirm aus. *Hinweis:* Verwenden Sie für die notwendige Vergleichsoperation in der Funktion `subset` oder in den eckigen Indizierungsklammern den richtigen Vergleichsoperator `==` (und nicht `=`).

Übung 7

Wie können Sie im eben erstellten Data Frame `df_f` auf die erste Spalte `name` zugreifen? Geben Sie drei Möglichkeiten an!