# Fundamental object types in R III: Factors, matrices, and data frames

Claudius Gräbner-Radkowitsch

2022-03-02

- TOC
- Picture with overview

## Faktoren

Faktoren werden verwendet um ordinale oder kategoriale Daten darzustellen. Ein Faktor kann nur einen von mehreren vorher definierten Werten annehmen, so genannten *Levels*. Faktoren werden über die Funktion `factor()` erstellt. Sie nimmt als erstes Argument die Werte für den Faktor:

```
x <- c("Frau", "Mann", "Frau")
x <- factor(c("Frau", "Mann", "Frau"))
x
```

```
## [1] Frau Mann Frau
## Levels: Frau Mann
```

Wenn wir Levels definieren wollen, die aber aktuell noch keine Ausprägung haben können wir dies mit dem Argument `levels` bewerkstelligen:

```
x <- c("Frau", "Mann", "Frau")
x <- factor(c("Frau", "Mann", "Frau"),
            levels=c("Divers","Frau", "Mann"))
x
```

```
## [1] Frau Mann Frau
## Levels: Divers Frau Mann
```

Wenn wir das Argument `levels` verwenden werden dort nicht genannte Ausprägungen den Wert `NA` erhalten:

```
x <- c("Frau", "Mann", "Frau")
x <- factor(c("Frau", "Mann", "Frau", "Divers"),
            levels=c("Frau", "Mann"))
x
```

```
## [1] Frau Mann Frau <NA>
## Levels: Frau Mann
```

Die Reihenfolge der einzelnen Levels spielt meist keine Rolle. Bei ordinalen Daten möchten wir aber eine sinnvolle Wertigkeit der Ausprägungen sicherstellen. Das geht mit der Funktion `factor()` und dem Argument `ordered`:

```
x <- c("Hoch", "Hoch", "Gering", "Hoch")
x <- factor(x,
            levels = c("Gering", "Mittel", "Hoch"),
```

```
            ordered = TRUE)
x
```

```
## [1] Hoch   Hoch   Gering Hoch
## Levels: Gering < Mittel < Hoch
```

Häufig handelt es sich bei den Ausprägungen von Faktoren um Wörter, also Objekte vom Type `character`. Technisch gesehen werden Faktoren aber als `integer` gespeichert: um Speicherplatz zu sparen wird jedem Level auf dem Computer eine ganze Zahl zugewiesen, die dann auf den eigentlichen Wert gemapt wird. Gerade wenn die Ausprägungen als solche große Zahlen oder lange Wörter sind spart das Speicher, weil diese Ausprägungen nur einmal gespeichert werden müssen, und jedes Element des Fakors nur noch eine einfache Zahl ist. Daher gibt `typeof()` für Faktoren auch `integer` aus:

```
x <- factor(c("Frau", "Mann", "Frau"),
            levels=c("Mann", "Frau", "Divers"))
typeof(x)
```

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

Um zu überprüfen ob es sich bei einem Objekt um einen Faktor handelt verwenden wir die Funktion `is.factor()`:

```
is.factor(x)
```

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

Manche Operationen, die für `integer` definiert sind, funktionieren bei Faktoren aber nicht, z.B. Addition:

```
x[1] + x[2]
```

```
## Warning in Ops.factor(x[1], x[2]): '+' not meaningful for factors
```

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

Dafür können wir andere nützliche Dinge mit Faktoren anstellen, z.B. die absoluten Häufigkeiten über die Funktion `table()` anzeigen:

```
table(x)
```

```
## x
##   Mann   Frau Divers
##      1      2      0
```

Faktoren werden vor allem in der Arbeit mit ordinalen und kategorialen Daten verwendet (siehe Kapitel @ref(data)).

# Matrizen

Bei Matrizen handelt es sich um zweidimensionale Objekte mit Zeilen und Spalten, bei denen es sich jeweils um atomare Vektoren handelt.

### Erstellen von Matrizen

Matrizen werden mit der Funktion `matrix()` erstellt. Diese Funktion nimmt als erstes Argument die Elemente der Matrix und dann die Spezifikation der Anzahl von Zeilen (`nrow`) und/oder der Anzahl von Spalten (`ncol`):

```
m_1 <- matrix(11:20, nrow = 5)
m_1
```

```
##      [,1] [,2]
## [1,]   11   16
```

```
## [2,]   12   17
## [3,]   13   18
## [4,]   14   19
## [5,]   15   20
```

Wir können die Zeilen und Spalten sowie einzelne Werte folgendermaßen extrahieren und gegebenenfalls Ersetzungen vornehmen:

```
m_1[,1] # Erste Spalte
```

```
## [1] 11 12 13 14 15
```

```
m_1[1,] # Erste Zeile
```

```
## [1] 11 16
```

```
m_1[2,2] # Element [2,2]
```

```
## [1] 17
```

> **Optionaler Hinweis:** Matrizen sind weniger 'fundamental' als atomare Vektoren. Entsprechend gibt uns `typeof()` für eine Matrix auch den Typ der enthaltenen atomaren Vektoren an:

```
typeof(m_1)
```

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

> Um zu testen ob es sich bei einem Objekt um eine Matrix handelt verwenden wir entsprechend `is.matrix()`:

```
is.matrix(m_1)
```

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

```
is.matrix(2.0)
```

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

Die Grundlagen der Matrizenalgebra und ihre Implementierung in R wird später in Kapitel @ref(formalia) erläutert. Zudem gibt es im Internet zahlreiche gute Überblicksartikel zum Thema Matrizenalgebra in R, z.B. hier oder in größerem Umfang hier.

## Data Frames

Der `data.frame` ist eine besondere Art von Liste und ist ein in der Datenanalyse regelmäßig auftretender Datentyp. Im Gegensatz zu einer normalen Liste müssen bei einem `data.frame` alle Elemente die gleiche Länge aufweisen. Das heißt man kann sich einen `data.frame` als eine rechteckig angeordnete Liste vorstellen.

Wegen der engen Verwandschaft können wir einen `data.frame` direkt aus einer Liste erstellen indem wir die Funktion `as.data.frame()` verwenden:

```
l_3 <- list(
  "a" = 1:3,
  "b" = 4:6,
  "c" = 7:9
)
df_3 <- as.data.frame(l_3)
```

Wenn wir R nach dem Typ von `df_3` fragen, sehen wir, dass es sich weiterhin um eine Liste handelt:

```
typeof(df_3)
```

```
## [1] "list"
```

Allerdings können wir testen ob `df_3` ein `data.frame` ist indem wir `is.data.frame` benutzen:

```
is.data.frame(df_3)
```

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

```
is.data.frame(l_3)
```

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

Wenn wir `df_3` ausgeben sehen wir unmittelbar den Unterschied zur klassischen Liste:

```
l_3
```

```
## $a
## [1] 1 2 3
##
## $b
## [1] 4 5 6
##
## $c
## [1] 7 8 9
```

```
df_3
```

```
##   a b c
## 1 1 4 7
## 2 2 5 8
## 3 3 6 9
```

Die andere Möglichkeit einen `data.frame` zu erstellen ist direkt über die Funktion `data.frame()`, wobei es hier in der Regel ratsam ist das optionale Argument `stringsAsFactors` auf `FALSE` zu setzen, da sonst Wörter in so genannte Faktoren umgewandelt werden:[1]

```
df_4 <- data.frame(
  "gender" = c(rep("male", 3), rep("female", 2)),
  "height" = c(189, 175, 180, 166, 150),
  stringsAsFactors = FALSE
)
df_4
```

```
##   gender height
## 1   male    189
## 2   male    175
## 3   male    180
## 4 female    166
## 5 female    150
```

Data Frames sind das klassische Objekt um eingelesene Daten zu repräsentieren. Wenn Sie sich z.B. Daten zum BIP in Deutschland aus dem Internet runterladen und diese Daten dann in R einlesen, werden diese Daten zunächst einmal als `data.frame` repräsentiert.[2] Diese Repräsentation erlaubt dann eine einfache Analyse und Manipulation der Daten.

[1] Zur Geschichte dieses wirklich ärgerlichen Verhaltens siehe diesen Blog. Zwar wurde das Standardverhalten mit R 4.0 umgestellt, allerdings empfiehlt sich die explizite Setzung von `stringsAsFactors=F` trotzdem, damit der Code auch mit älteren Versionen gut funktioniert.

[2] Das ist nicht ganz korrekt, weil es mittlerweilse Erweiterungen gibt, welche den `data.frame` mit effizienteren Objekten ersetzen, z.B. dem `tibble` oder dem `data.table`. Der Umgang mit diesen Objekten ist jedoch sehr ähnlich zum `data.frame`.

Zwar gibt es ein eigenes Kapitel zur Bearbeitung von Daten (siehe Kapitel @ref(data)), wir wollen aber schon hier einige zentrale Befehle im Zusammenhang von Data Frames einführen.

An dieser Stelle sei schon angemerkt, dass um Zeilen, Spalten oder einzelne Elemente auszuwählen die gleichen Befehle wie bei Matrizen verwendet werden können:

```
df_4[, 1] # Werte der ersten Spalte
```

```
## [1] "male"   "male"   "male"   "female" "female"
```

```
df_4[, 2] # Werte der zweiten Spalte
```

```
## [1] 189 175 180 166 150
```

Die Abfrage funktioniert nicht nur mit Indices, sondern auch mit Spaltennamen:[3]

```
df_4[["gender"]]
```

```
## [1] "male"   "male"   "male"   "female" "female"
```

Wenn wir `[` anstatt von `[[` verwenden erhalten wir als Output einen (reduzierten) Data Frame:

```
df_4["gender"]
```

```
##   gender
## 1   male
## 2   male
## 3   male
## 4 female
## 5 female
```

Es können auch mehrere Zeilen ausgewählt werden:

```
df_4[1:2, ] # Die ersten beiden Zeilen
```

```
##   gender height
## 1   male    189
## 2   male    175
```

Oder einzelne Werte:

```
df_4[2, 2] # Zweiter Wert der zweiten Spalte
```

```
## [1] 175
```

Dies können wir uns zu Nutze machen um den Typ der einzelnen Spalten herauszufinden:

```
typeof(df_4[["gender"]])
```

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

Gerade bei sehr großen Data Frames möchte man oft nur die ersten paar Zeilen inspizieren. Das ist mit der Funktion `head()` möglich. Das erste Argument ist immer der Name des Data Frames. Das zweite (optionale) Argument ist ein `integer`, der die Anzahl der anzuzeigenden Zeilen angibt (Standardwert: `5`):

```
head(df_4, 2) # gibt die ersten zwei Zeilen aus
```

```
##   gender height
## 1   male    189
## 2   male    175
```

[3] Anstelle von `[[` kann auch der Shortcut `$` verwendet werden. Das werden wir aufgrund der größeren Transparenz von `[[` hier jedoch nicht verwenden.