0_1_datentypen_vektoren

March 17, 2023

1 Datenstrukturen

Es gibt verschiedene Datenstrukturen in R:

- Vektoren
- Matrix / Matrizen
- Listen
- Arrays
- Data Frames
- Faktoren

Es gibt 6 Datentypen in R:

- Integer
- Double
- Logical
- Character
- Complex
- Raw

1.1 Vektoren

Vektoren stellen die einfachste Form der Datenstrukturen dar. Sie bestehen aus einem bis vielen Elementen desselben Datentyps.

1.1.1 Integer

Es gibt verschiedene Theorien, warum Integers mit L gekennzeichnet sind. Eine lautet, dass L in C für *32 bit long integer* steht.

[1]:
```
# Integer
v <- c(1L, 2L, 3L)
v
```

1. 1 2. 2 3. 3

[2]:
```
typeof(v)
```

'integer'

[3]:
```
str(v)
```

```
 int [1:3] 1 2 3
```

1.1.2 Double

[4]:
```
# Double
v <- c(1.0, 2.0, 3.0)
v
```

1. 1 2. 2 3. 3

[5]:
```
typeof(v)
```

'double'

[6]:
```
str(v)
```

```
 num [1:3] 1 2 3
```

1.1.3 Logical

[7]:
```
# Logical
v <- c(TRUE, FALSE, FALSE, TRUE)
v
```

1. TRUE 2. FALSE 3. FALSE 4. TRUE

[8]:
```
typeof(v)
```

'logical'

[9]:
```
str(v)
```

```
 logi [1:4] TRUE FALSE FALSE TRUE
```

1.1.4 Character

Ein String als Datenstruktur besteht aus Charactern.

Ein Vektor als Datenstruktur kann aus mehreren Strings bestehen.

[10]:
```
# Character
v <- c("Anton", "Berta", "Christine")
v
```

1. 'Anton' 2. 'Berta' 3. 'Christine'

[11]:
```
typeof(v)
```

'character'

[12]:
```
str(v)
```

```
 chr [1:3] "Anton" "Berta" "Christine"
```

1.1.5 Complex

[13]:
```
# Einfache komplexe Zahl.
complex_value <- 3 + 2i
complex_value
```

3+2i

[14]:
```
typeof(complex_value)
```

'complex'

[15]:
```
str(complex_value)
```

```
 cplx 3+2i
```

[16]:
```
# Vektor von komplexen Zahlen.
complex_vector <- complex(real = c(1, 2, 3), imaginary = c(3, 2,1))
complex_vector
```

1. 1+3i 2. 2+2i 3. 3+1i

[17]:
```
typeof(complex_vector)
```

'complex'

[18]:
```
str(complex_vector)
```

```
 cplx [1:3] 1+3i 2+2i 3+1i
```

1.1.6 Raw

Raw Data sind byteweise Darstellungen.

Die Strings sind vom Grundsatz UTF-8 encodiert. Das Ö hat die Darstellung in zwei Bytes c3 96.

[19]:
```
# Raw Daten sind byteweise Darstellungen.
v <- charToRaw(c("Hallo!"))
v
v1 <- charToRaw(c("ÖPNV"))
v1
v2 <- charToRaw(c("OPNV"))
v2
```

```
[1] 48 61 6c 6c 6f 21
```

```
[1] c3 96 50 4e 56
```

```
[1] 4f 50 4e 56
```

[20]:
```
typeof(v)
```

'raw'

[21]:
```
str(v)
```

```
raw [1:6] 48 61 6c 6c …
```

1.2 Arbeiten mit Vektoren

Wir erzeugen verschiedene Vektoren.

[22]:
```
# seq() erzeugt eine Sequenz von Elementen mit
v1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
v1
v1 <- seq(1:10)
v1
v2 <- seq(1, 10, by = 0.5)
v2
v3 <- seq(0, 100, by = 10)
v3
# rep() wiederholt kompletten Vektor oder elementweise
v4 <- c("rot", "gelb", "grün", "blau", "schwarz", "weiß")
v4
v5 <- rep(v4, times = 2)
v5
v6 <- rep(v4, each = 2)
v6
```

1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10

1. 1 2. 2 3. 3 4. 4 5. 5 6. 6 7. 7 8. 8 9. 9 10. 10

1. 1 2. 1.5 3. 2 4. 2.5 5. 3 6. 3.5 7. 4 8. 4.5 9. 5 10. 5.5 11. 6 12. 6.5 13. 7 14. 7.5 15. 8 16. 8.5 17. 9 18. 9.5 19. 10

1. 0 2. 10 3. 20 4. 30 5. 40 6. 50 7. 60 8. 70 9. 80 10. 90 11. 100

1. 'rot' 2. 'gelb' 3. 'grün' 4. 'blau' 5. 'schwarz' 6. 'weiß'

1. 'rot' 2. 'gelb' 3. 'grün' 4. 'blau' 5. 'schwarz' 6. 'weiß' 7. 'rot' 8. 'gelb' 9. 'grün' 10. 'blau' 11. 'schwarz' 12. 'weiß'

1. 'rot' 2. 'rot' 3. 'gelb' 4. 'gelb' 5. 'grün' 6. 'grün' 7. 'blau' 8. 'blau' 9. 'schwarz' 10. 'schwarz' 11. 'weiß' 12. 'weiß'

Wir selektieren auf unterschiedliche Art und Weise

[23]:
```
# Elemente auswählen nach Position
v4
v4[3] # das dritte Element
v4[-v3] # alle außer dem dritten Element
v4[3:5] # Elemente 3 bis 5
v4[-(3:5)] # alle außer den Elementen 3 bis 5
v4[c(2, 4)] # Element 2 und Element 4
v4[length(v4)] # Letztes Element, eine von vielen Möglichkeiten
```

1. 'rot' 2. 'gelb' 3. 'grün' 4. 'blau' 5. 'schwarz' 6. 'weiß'

'grün'

1. 'rot' 2. 'gelb' 3. 'grün' 4. 'blau' 5. 'schwarz' 6. 'weiß'

1. 'grün' 2. 'blau' 3. 'schwarz'

1. 'rot' 2. 'gelb' 3. 'weiß'

1. 'gelb' 2. 'blau'

'weiß'

[24]:
```
# Elemente auswählen nach Wert
v2
v2[v2 == 4] # Element gleich 4
v2[v2 < 4] # Elemente kleiner als 4
mychoice <- c(2, 4, 6, 8, 10, 12)
v2[v2 %in% mychoice]
```

1. 1 2. 1.5 3. 2 4. 2.5 5. 3 6. 3.5 7. 4 8. 4.5 9. 5 10. 5.5 11. 6 12. 6.5 13. 7 14. 7.5 15. 8 16. 8.5 17. 9 18. 9.5 19. 10

4

1. 1 2. 1.5 3. 2 4. 2.5 5. 3 6. 3.5

1. 2 2. 4 3. 6 4. 8 5. 10

1.3 Weitere Vektorfunktionen

[25]:
```
# Vektor sortieren
v5
sort(v5)
order(v5)
```

1. 'rot' 2. 'gelb' 3. 'grün' 4. 'blau' 5. 'schwarz' 6. 'weiß' 7. 'rot' 8. 'gelb' 9. 'grün' 10. 'blau' 11. 'schwarz' 12. 'weiß'

1. 'blau' 2. 'blau' 3. 'gelb' 4. 'gelb' 5. 'grün' 6. 'grün' 7. 'rot' 8. 'rot' 9. 'schwarz' 10. 'schwarz' 11. 'weiß' 12. 'weiß'

1. 4 2. 10 3. 2 4. 8 5. 3 6. 9 7. 1 8. 7 9. 5 10. 11 11. 6 12. 12

[26]:
```
# Im Vektor die Reihenfolge der Elemente umkehren
rev(v5)
rev(sort(v5))
```

1. 'weiß' 2. 'schwarz' 3. 'blau' 4. 'grün' 5. 'gelb' 6. 'rot' 7. 'weiß' 8. 'schwarz' 9. 'blau' 10. 'grün' 11. 'gelb' 12. 'rot'

1. 'weiß' 2. 'weiß' 3. 'schwarz' 4. 'schwarz' 5. 'rot' 6. 'rot' 7. 'grün' 8. 'grün' 9. 'gelb' 10. 'gelb' 11. 'blau' 12. 'blau'

[27]:
```
# Eindeutige Elemente ausgeben / Dubletten entfernen
unique(v5)
sort(unique(v5))
```

1. 'rot' 2. 'gelb' 3. 'grün' 4. 'blau' 5. 'schwarz' 6. 'weiß'

1. 'blau' 2. 'gelb' 3. 'grün' 4. 'rot' 5. 'schwarz' 6. 'weiß'

[28]:
```
# Einfache Häufigkeitstabelle
table(v5)
```

```
v5
   blau    gelb    grün     rot schwarz    weiß
      2       2       2       2       2       2
```

[29]:
```
# Einfache Stichprobe mit Häufigkeitstabelle
set.seed(123)
n <- 1000
s1 <- sample(x = c("rot", "gelb", "grün"), size = n, replace = TRUE, prob = c(1/
 ↪4, 1/4, 1/2))
table(s1)
dfs1 <- data.frame(table(s1))

dfs1$Pct <- dfs1$Freq/n*100
dfs1
```

```
s1
gelb grün  rot
 245  507  248
```

A data.frame: 3 × 3

s1 <fct>	Freq <int>	Pct <dbl>
gelb	245	24.5
grün	507	50.7
rot	248	24.8

[ ]:

[ ]: