



Programmierung und Deskriptive Statistik

BSc Psychologie WiSe 2023/24

Belinda Fleischmann

Inhalte basieren auf Programmierung und Deskriptive Statistik von Dirk Ostwald, lizenziert unter CC BY-NC-SA 4.0

Datum	Einheit	Thema
11.10.23	Einführung	(1) Einführung
18.10.23	R Grundlagen	(2) R und Visual Studio Code
25.10.23	R Grundlagen	(2) R und Visual Studio Code
01.11.23	R Grundlagen	(3) Vektoren
08.11.23	R Grundlagen	(4) Matrizen
15.11.23	R Grundlagen	(5) Listen und Dataframes
22.11.23	R Grundlagen	(6) Datenmanagement
29.11.23	Deskriptive Statistik	(7) Häufigkeitsverteilungen
06.12.23	Deskriptive Statistik	(8) Verteilungsfunktionen und Quantile
13.12.23	Deskriptive Statistik	(9) Maße der zentralen Tendenz
20.12.23	Deskriptive Statistik	(10) Maße der Datenvariabilität
20.12.23	<i>Leistungsnachweis Teil 1</i>	
	Weihnachtspause	
10.01.24	Deskriptive Statistik	(11) Anwendungsbeispiel (Deskriptive Statistik)
17.01.24	Inferenzstatistik	(12) Anwendungsbeispiel (Parameterschätzung, Konfidenzintervalle)
24.01.24	Inferenzstatistik	(13) Anwendungsbeispiel (Hypothesentest)
25.01.24	<i>Leistungsnachweis Teil 2</i>	

(5) Listen und Dataframes

Listen

Dataframes

Übungen und Selbstkontrollfragen

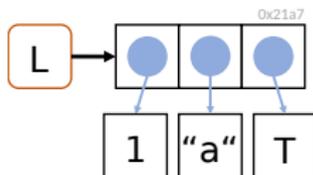
Listen

Dataframes

Übungen und Selbstkontrollfragen

- Listen sind geordnete Folgen von R Objekten.
- Listen sind rekursiv, können also Objekte verschiedenen Datentyps enthalten.
- Defacto enthalten Listen keine Objekte, sondern Referenzen zu Objekten.

```
L <- list(1, "a", T)
```



- Listen sind ein wesentlicher Baustein von Dataframes.

Erzeugung

Direkte Konkatenation von Listenelementen mit list()

```
L <- list(c(1, 4, 5), # Liste mit einem Vektor,  
         matrix(1:8, nrow = 2), # einer Matrix und  
         exp) # einer Funktion  
print(L)
```

```
[[1]]  
[1] 1 4 5
```

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

```
[[3]]  
function (x) .Primitive("exp")
```

Listen können Elemente von Listen sein

```
L <- list(list(1)) # Liste mit Element 1 in einer Liste  
print(L)
```

```
[[1]]  
[[1]][[1]]  
[1] 1
```

c() kann zum Verbinden von Listen genutzt werden

```
L <- c(list(pi), list("a")) # Konkatenation zweier Listen  
print(L)
```

```
[[1]]  
[1] 3.141593
```

```
[[2]]  
[1] "a"
```

Charakterisierung

Der Datentyp von Listen ist `list`

```
L <- list(1:2, "a", log)      # Erzeugung einer Liste
typeof(L)                   # Typenbestimmung
```

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

`length()` gibt die Anzahl der Toplevel Listenelemente aus

```
L <- list(1:2, list("a", pi), exp) # Liste mit drei Toplevel-Elementen
length(L)                         # length() ignoriert Elementinhalte, length() von L ist also 3
```

```
[1] 3
```

Die Dimension, Zeilen-, und Spaltenanzahl von Listen ist `NULL`

```
L <- list(1:2, "a", sin)      # eine Liste
dim(L)                       # Die Dimension von Listen ist NULL
```

```
NULL
```

```
nrow(L)                      # Die Zeilenanzahl von Listen ist NULL
```

```
NULL
```

```
ncol(L)                      # Die Spaltenanzahl von Listen ist NULL
```

```
NULL
```

Listenelement vs. Listenelementinhalt

Einfache eckige Klammern [] indizieren **Listenelemente als Listen**

```
L <- list(1:3, "a", exp)      # eine Liste  
l1 <- L[1]                  # Indizierung eines Listenelements  
print(l1)
```

```
[[1]]  
[1] 1 2 3  
typeof(l1)                  # Typbestimmung von l1
```

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

Doppelte eckige Klammern [[]] indizieren den **Inhalt von Listenelementen**

```
L <- list(1:3, "a", exp)      # eine Liste  
i2 <- L[[2]]                 # Indizierung des Listenelementinhalts  
print(i2)
```

```
[1] "a"  
typeof(i2)                  # Typbestimmung von i2
```

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

Ersetzen von Listenelement(inhalt)en

Ersetzen eines Listenelements über einfache eckige Klammer []

```
L      <- list(1:3, "a", exp)      # eine Liste
L[1]   <- 4:6                     # Ersetzung des 1. Listenelementes mit gewünschtem Inhalt
```

Warning in L[1] <- 4:6: number of items to replace is not a multiple of replacement length

```
print(L[1])                      # -> keine Typkonversion vector zu list; erzeugt stattdessen warning message
                                     # und übernimmt nur erstes Element des Vektors
```

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

```
L[1]   <- list(4:6)              # Ersetzung des 1. Listenelementes mit einer Liste des gewünschten Inhalts
print(L[1])
```

```
[[1]]
[1] 4 5 6
```

Ersetzen des Listenelementinhalts über doppelte eckige Klammern [[]]

```
L[[1]] <- 7:9                    # Ersetzung des 1. Listenelementinhaltes mit gewünschtem Inhalt
L[[3]] <- "c"                    # Ersetzung des 3. Listenelementinhaltes mit gewünschtem Inhalt
print(L)
```

```
[[1]]
[1] 7 8 9
```

```
[[2]]
[1] "a"
```

```
[[3]]
[1] "c"
```

Prinzipien der Listenindizierung

Die Prinzipien der Listenindizierung sind analog zur Vektorindizierung.

Vektoren positiver Zahlen adressieren entsprechende Elemente

```
L <- list(1:3, "a", pi)           # eine Liste
print(L[c(1, 3)])                # 1. und 3. Listenelement
```

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

```
[[2]]
[1] 3.141593
```

Vektoren negativer Zahlen adressieren komplementäre Elemente

```
print(L[-c(1, 3)])              # 2. Listenelement
```

```
[[1]]
[1] "a"
```

Logische Vektoren adressieren Elemente mit TRUE.

```
print(L[c(TRUE, TRUE, FALSE)])  # 1. und 2. Listenelement
```

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

```
[[2]]
[1] "a"
```

Attribute

Listenelementen können bei Erzeugung Namen gegeben werden

```
L <- list(greta = 1:3,          # eine Liste mit benannten Elementen
          luisa = "a",
          carla = exp)
print(L)
```

```
$greta
[1] 1 2 3
```

```
$luisa
[1] "a"
```

```
$carla
function (x) .Primitive("exp")
```

Namen werden automatisch als Attribut `names` gespeichert und können mit `attributes()` angezeigt werden

```
print(attributes(L))
```

```
$names
[1] "greta" "luisa" "carla"
```

Listenelementen können auch mit `names()` Namen gegeben werden

```
K <- list(1:2, TRUE)          # eine unbenannte Liste
names(K) <- c("Frodo", "Sam") # Namensgebung mit names()
print(K)
```

```
$Frodo
[1] 1 2
```

```
$Sam
[1] TRUE
```

Indizierung über Namen und dem \$-Operator

Mit Namen können **Listenelemente** ([]) und **Listenelementinhalte** ([[]]) indiziert werden.

```
L <- list(greta = 1:3,      # eine Liste mit benannten Elementen
         luisa = "a",
         carla = exp)
L["carla"]                # Listenelementindizierung
```

```
$carla
function (x) .Primitive("exp")
L[["carla"]]              # Listenelementinhaltsindizierung
```

```
function (x) .Primitive("exp")
```

Mit dem \$-Operator können **Listenelementinhalte** indiziert werden.

```
L <- list(greta = 1:3,      # eine Liste mit benannten Elementen
         luisa = "a",
         carla = exp)
L$greta                   # Listenelementinhalt
```

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

```
L$luisa                  # Listenelementinhalt
```

```
[1] "a"
```

```
L$carla                  # Listenelementinhalt
```

```
function (x) .Primitive("exp")
```

Listenarithmetik ist nicht definiert, da Listenelemente unterschiedlichen Typs sein können

```
L1 <- list(1:3, "a")      # eine Liste
L2 <- list(TRUE, exp)    # eine Liste
L1+L2                    # Versuch der Listenaddition
```

Error in L1 + L2: non-numeric argument to binary operator

Listenelementinhalte können bei Passung jedoch arithmetisch verknüpft werden

```
L1 <- list(1:3, 1)      # eine Liste
L2 <- list(4:6, exp)    # eine Liste
L1[[1]] + L2[[1]]      # Addition der 1. Listenelementinhalte, [1+4, 2+5, 3+6]
```

```
[1] 5 7 9
```

```
L2[[2]](L1[[2]])      # Anwendung des 2. Listenelementinhalts auf den 1. Listenelementinhalt -> exp(1)
```

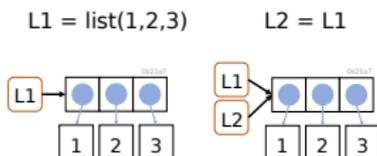
```
[1] 2.718282
```

Copy-on-modify

Wie bei Vektoren gilt bei Listen das Copy-on-Modify Prinzip.

“Shallow copy”: Listenobjekt wird kopiert, aber nicht die gebundenen Objekte.

`lobstr::ref()` erlaubt es, dieses Verhalten zu studieren.



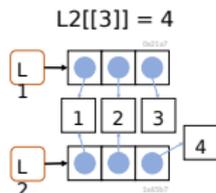
```
L1 <- list(1, 2, 3) # Erzeugen einer Liste als Objekt (z.B. 0x1a3)
L2 <- L1           # L2 wird das selbe listenobject wie L1 zugewiesen
lobstr::ref(L1, L2) # Ausgabe der Referenzen beider Listen
```

```
[1:0x55da32102868] <list>
[2:0x55da30ac78b0] <dbl>
[3:0x55da30ac78e8] <dbl>
[4:0x55da30ac7920] <dbl>
```

```
[1:0x55da32102868]
```

Copy-on-modify

Änderung nur eines Listenelements



```
L2[[3]] <- 4 # Copy-on-Modify mit shallow Objekt Kopie
lobstr::ref(L1, L2) # Ausgabe der Referenzen beider Listen
```

```
[1:0x55da32102868] <list>
[2:0x55da30ac78b0] <dbl>
[3:0x55da30ac78e8] <dbl>
[4:0x55da30ac7920] <dbl>
```

```
[5:0x55da334bb948] <list>
[2:0x55da30ac78b0]
[3:0x55da30ac78e8]
[6:0x55da32c6dd88] <dbl>
```

Anmerkung: Die Referenzen der Objekte beziehen sich auf die adressierten Speicherzellen im Arbeitsspeicher. Da bei jeder Neuerstellung von Objekten, neue Speicherzellen vergeben werden, variieren die genauen Adressen bei jeder Neuerstellung von Objekten. Entsprechend werden die mit `lobstr::ref()` ausgegebenen Referenzen nicht mit denen in der Abbildung oder bei Replikation zu Hause übereinstimmen.

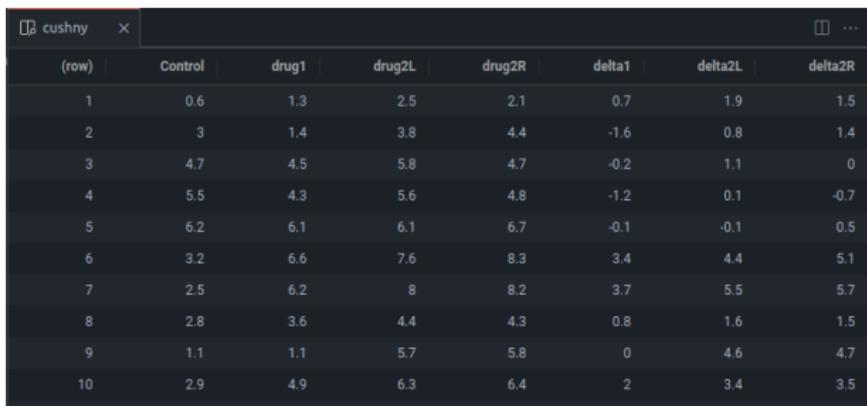
Listen

Dataframes

Übungen und Selbstkontrollfragen

Übersicht

- Dataframes sind die zentrale Datenstruktur in R.
- Dataframes stellt man sich am besten als Tabelle vor.
- Die Zeilen und Spalten der Tabelle haben Namen.



The screenshot shows an R console window with a data frame named 'cushny'. The data frame has 10 rows and 8 columns. The columns are labeled (row), Control, drug1, drug2L, drug2R, delta1, delta2L, and delta2R. The rows are numbered 1 through 10. The values in the columns are as follows:

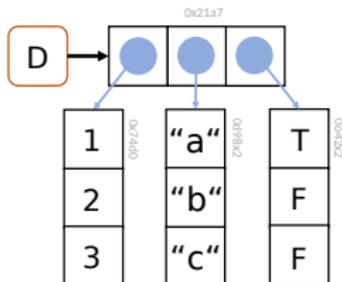
(row)	Control	drug1	drug2L	drug2R	delta1	delta2L	delta2R
1	0.6	1.3	2.5	2.1	0.7	1.9	1.5
2	3	1.4	3.8	4.4	-1.6	0.8	1.4
3	4.7	4.5	5.8	4.7	-0.2	1.1	0
4	5.5	4.3	5.6	4.8	-1.2	0.1	-0.7
5	6.2	6.1	6.1	6.7	-0.1	-0.1	0.5
6	3.2	6.6	7.6	8.3	3.4	4.4	5.1
7	2.5	6.2	8	8.2	3.7	5.5	5.7
8	2.8	3.6	4.4	4.3	0.8	1.6	1.5
9	1.1	1.1	5.7	5.8	0	4.6	4.7
10	2.9	4.9	6.3	6.4	2	3.4	3.5

Anmerkung: Der Datensatz `cushny` ist im R Paket `psychTools` enthalten und kann nach Installation (`install.packages()`) und Laden (`library()`) mit dem Befehl `data(cushny)` geladen werden. Mehr Details zu `cushny` und weiteren Datensätzen [hier](#).

Übersicht

- Formal ist ein Dataframe eine Liste, deren Elemente Vektoren gleicher Länge sind.
- Die Listenelemente entsprechen den Spalten einer Tabelle.
- Die Vektorelemente gleicher Position entsprechen den Zeilen einer Tabelle.

```
D <- data.frame(c(1,2,3),  
               c("a", "b", "c"),  
               c(T,F,F))
```



Erzeugung

`data.frame()` erzeugt einen Dataframe

```
D <- data.frame(x = letters[1:4], # 1. Spalte mit Name x
               y = 1:4,         # 2. Spalte mit Name y
               z = c(T, T, F, T)) # 3. Spalte mit Name z
print(D)
```

```
  x y    z
1 a 1 TRUE
2 b 2 TRUE
3 c 3 FALSE
4 d 4 TRUE
```

Die Spalten des Dataframes müssen gleiche Länge haben

```
D <- data.frame(x = letters[1:4], # 1. Spalte mit Name x
               y = 1:4,         # 2. Spalte mit Name y
               z = c(T, T, F))  # 3. Spalte mit Name z
```

Error in `data.frame(x = letters[1:4], y = 1:4, z = c(T, T, F))`: arguments imply differing number of rows:

Die Spalten eines Dataframes können offenbar unterschiedlichen Typs sein

⇒ rekursive Datenstruktur.

Charakterisierung

Ein Dataframe hat `names()`, `rownames()`, `colnames()`

```
D <- data.frame(age = c(30, 35, 40, 45), # 1. Spalte
               height = c(178, 189, 165, 171), # 2. Spalte
               weight = c(67, 76, 81, 92)) # 3. Spalte
names(D) # names gibt die Spaltennamen aus
```

```
[1] "age" "height" "weight"
```

```
colnames(D) # colnames entspricht names
```

```
[1] "age" "height" "weight"
```

```
rownames(D) # default rownames sind 1,2,...
```

```
[1] "1" "2" "3" "4"
```

Ein Dataframe `nrow()` Zeilen und `length()` bzw. `ncol()` Spalten

```
nrow(D) # Zeilenanzahl
```

```
[1] 4
```

```
ncol(D) # Spaltenanzahl
```

```
[1] 3
```

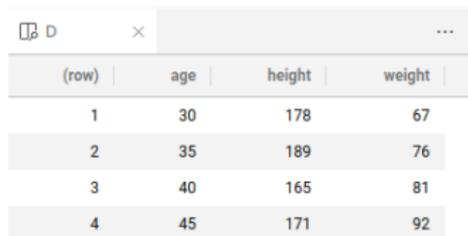
```
length(D) # Länge ist die Spaltenanzahl
```

```
[1] 3
```

Charakterisierung

`View()` öffnet den Data Viewer.

```
View(D)
```



(row)	age	height	weight
1	30	178	67
2	35	189	76
3	40	165	81
4	45	171	92

`str()` zeigt in kompakter Form wesentliche Aspekte eines Dataframes an.

```
str(D)
```

```
'data.frame':  4 obs. of  3 variables:
 $ age   : num  30 35 40 45
 $ height: num  178 189 165 171
 $ weight: num  67 76 81 92
```

Allgemein zeigt `str()` in kompakter Form wesentliche Aspekte eines R Objektes an.

Attribute

Dataframes sind Listen mit Attributen für (column) names und row.names. Dataframes haben class "data.frame".

```
typeof(D)
```

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

```
attributes(D)
```

```
$names
```

```
[1] "age"    "height" "weight"
```

```
$class
```

```
[1] "data.frame"
```

```
$row.names
```

```
[1] 1 2 3 4
```

ab dataframes übersicht müssen auch noch slides formate angepasst werden

Indizierung - Mit einem Index

Die Prinzipien der Indizierung für Vektoren und Matrizen gelten auch für Dataframes

⇒ Bei **einem** Index verhalten sich Dataframes wie **Listen**

```
D <- data.frame(x = letters[1:4], # 1. Spalte mit Name x
               y = 1:4,         # 2. Spalte mit Name y
               z = c(T, T, F, T)) # 3. Spalte mit Name z
print(D)
```

```
  x y    z
1 a 1 TRUE
2 b 2 TRUE
3 c 3 FALSE
4 d 4 TRUE
```

```
class(D) # D ist ein Dataframe
```

```
[1] "data.frame"
```

```
v <- D[1] # 1. Listenelement als Dataframe
v
```

```
  x
1 a
2 b
3 c
4 d
```

```
class(v) # v ist ein Dataframe
```

```
[1] "data.frame"
```

Indizierung - Mit einem Index

```
w <- D[[1]]          # Inhalt des 1. Listenelements  
w
```

```
[1] "a" "b" "c" "d"
```

```
class(w)            # w ist ein character vector
```

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

```
y <- D$y            # $ zur Indizierung der y Spalte  
y
```

```
[1] 1 2 3 4
```

```
class(y)            # y ist ein Vektor vom Typ "integer" (!)
```

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

Indizierung - Mit zwei Indices

Die Prinzipien der Indizierung für Vektoren und Matrizen gelten auch für Dataframes

⇒ Bei **zwei** Indices verhalten sich Dataframes wie **Matrizen**

```
D <- data.frame(x = letters[1:4], # 1. Spalte mit Name x
               y = 1:4,         # 2. Spalte mit Name y
               z = c(T, T, F, T)) # 3. Spalte mit Name z
D[2:3,-2]                       # 1. Index für Zeilen, 2. Index für Spalten
```

```
  x      z
2 b TRUE
3 c FALSE
```

```
D[c(T, F, T, F),]                # 1. Index für Zeilen, 2. Index für Spalten
```

```
  x y      z
1 a 1 TRUE
3 c 3 FALSE
```

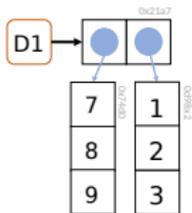
```
D[,c("x", "z")]                  # 1. Index für Zeilen, 2. Index für Spalten
```

```
  x      z
1 a TRUE
2 b TRUE
3 c FALSE
4 d TRUE
```

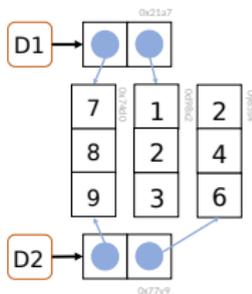
Copy-on-modify

- Die Copy-on-Modify Prinzipien für Listen gelten auch für Dataframes
- Modifikation einer Spalte führt zur Kopie der entsprechenden Spalte
- Modifikation einer Zeile führt zur Kopie des gesamten Dataframes

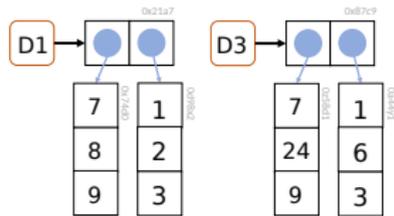
```
D1 <- data.frame(  
  c(7,8,9),c(1,2,3))
```



```
D2 <- D1  
D2[2,] <- D2[2,] * 2
```



```
D3 <- D1  
D3[2,] <- D3[2,] * 3
```



Listen

Dataframes

Übungen und Selbstkontrollfragen

Übungen und Selbstkontrollfragen

1. Dokumentieren Sie die in dieser Einheit eingeführten R Befehle.
2. Beschreiben Sie in einer Übersicht die R Datenstruktur "List".
3. Erzeugen Sie eine Liste mit vier Elementen.
4. L sei eine Liste. Benennen und demonstrieren Sie den Unterschied zwischen `L[1]` und `L[[1]]`.
5. Erzeugen Sie zwei Listen und fügen Sie diese zusammen.
6. L sei eine Liste. Was gibt `length(L)` an?
7. L sei eine Liste. Was bedeutet dann `L$Student`?
8. Erläutern Sie den Begriff "Shallow Copy" einer Liste.
9. Beschreiben Sie in einer Übersicht die R Datenstruktur "Dataframe".
10. Erzeugen Sie einen Dataframe mit vier Spalten.
11. D sei ein Dataframe. Was geben `rownames(D)` und `colnames(D)` an?
12. D sei ein Dataframe. Was ist der Unterschied zwischen `D[1]` und `D[[1]]`?
13. D sei ein Dataframe. Was ist der Unterschied zwischen `D[1]` und `D[1,1]`?
14. D sei ein Dataframe. Was bedeutet dann `D$Student`?
15. Erläutern Sie das Copy-on-modify Prinzip für Dataframes.