



# Programmierung und Deskriptive Statistik

BSc Psychologie WiSe 2022/23

Belinda Fleischmann

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

## (5) Listen und Dataframes

---

Listen

Dataframes

Übungen und Selbstkontrollfragen

---

**Listen**

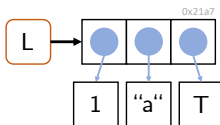
Dataframes

Übungen und Selbstkontrollfragen

## Übersicht

- 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.

# Listen

## Erzeugung

Direkte Konkatenation von Listenelementen mit `list()`

```
L = list(c(1,4,5),  
        matrix(1:8, nrow = 2),  
        exp)  
[[1]]  
[1] 1 4 5  
[[2]]  
  [,1] [,2] [,3] [,4]  
[1,]   1   3   5   7  
[2,]   2   4   6   8  
[[3]]  
function (x) .Primitive("exp")
```

*# Liste mit einem Vektor,  
# einer Matrix und  
# einer Funktion  
# 1. Listenelement  
# 2. Listenelement  
# 3. Listenelement*

Listen können Elemente von Listen sein

```
L = list(list(1))  
[[1]]  
[[1]][[1]]  
[1] 1
```

*# Liste mit Element 1 in einer Liste*

`c()` kann zum Verbinden von Listen genutzt werden

```
L = c(list(pi), list("a"))  
[[1]]  
[1] 3.141593  
[[2]]  
[1] "a"
```

*# Konkatenation zweier Listen  
# 1. Listenelement  
# 2. Listenelement*

# Listen

## 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 Toplevelelementen
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
```

# Listen

## Indizierung

Einfache eckige Klammern [ ] indizieren Listenelemente als Listen

```
L = list(1:3, "a", exp)      # eine Liste
l1 = L[1]                   # Indizierung eines Listenelements
[[1]]                       # das Listenelement l1
[1] 1 2 3                   # der Inhalt von Listenelement l1
typeof(l1)                  # Typbestimmung von l1
[1] "list"                  # L[] gibt eine Liste aus
```

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

```
L = list(1:3, "a", exp)      # eine Liste
i2 = L[[2]]                 # Indizierung des Listenelementinhalts
[1] "a"                     # der Inhalt von Listenelement L[2]
typeof(i2)                  # Typbestimmung von i2
[1] "character"             # L[[ ]] gibt den Listenelementinhalt aus
```

Ersetzen von Listenelement(inhalt)en

```
L      = list(1:3, "a", exp)  # eine Liste
L[1]   = 4:6                  # keine Typkonversion, Fehlermeldung
L[1]   = list(4:6)            # Ersetzung des 1. Listenelementes
L[[3]] = "c"                  # Ersetzung des 3. Listenelementinhaltes
```



# Listen

## Indizierung

Die Prinzipien der Listenindizierung sind analog zur Vektorindizierung

Vektoren positiver Zahlen adressieren entsprechende Elemente

```
L = list(1:3, "a", pi)           # eine Liste
l = L[c(1,3)]                   # 1. und 3. Listenelement
[[1]]                           # 1. Listenelement
[1] 1 2 3
[[2]]                           # 2. Listenelement
[1] 3.141593
```

Vektoren negativer Zahlen adressieren komplementäre Elemente

```
L = list(1:3, "a", pi)           # eine Liste
l = L[-c(1,3)]                  # 2. Listenelement
[1] "a"
```

Logische Vektoren adressieren Elemente mit TRUE.

```
L = list(1:3, "a", pi)           # eine Liste
l = L[c(T,T,F)]                 # 1. und 2. Listenelement
[[1]]                           # 1. Listenelement
[1] 1 2 3
[[2]]                           # 2. Listenelement
[1] "a"
```

## Indizierung

Listenelementen können bei Erzeugung Namen gegeben werden

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

$greta          # 1. Listenelement
[1] 1 2 3

$luisa          # 2. Listenelement
[1] "a"

$carla          # 3. Listenelement
function (x) .Primitive("exp")
```

Listenelementen können mit names() Namen gegeben werden

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

$Frodo          # 1. Listenelement
[1] 1 2

$Sam            # 2. Listenelement
[1] TRUE
```

# Listen

## Indizierung

Listenelemente und Listenelementinhalte können mit Namen indiziert werden

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

Listenelementinhalte können mit dem \$ Operator 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")
```

## Arithmetik

Listenarithmetik ist nicht definiert, da Listenelemente unterschiedlichen Typs sein können

```
L1 = list(1:3, "a" )      # eine Liste
L2 = list(T, 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, pi )      # 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]](1)              # Anwendung des 2. Listenelementinhalts, exp(1)
```

```
> [1] 2.72
```

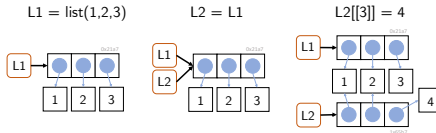
# Listen

## 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         # L1 und L2 referenzieren das Objekt (z.B. 0x1a3)
L2[[3]] = 4     # Copy-on-Modify mit shallow Objekt Kopie
lobstr::ref(L1,L2) # Ausgabe der Referenzen
```

```
> [1:0x557b6a7a2558] <list>
> [2:0x557b6c5974f0] <dbl>
> [3:0x557b6c5974b8] <dbl>
> [4:0x557b6c597480] <dbl>
>
> [5:0x557b6b2cc568] <list>
> [2:0x557b6c5974f0]
> [3:0x557b6c5974b8]
> [6:0x557b6c597368] <dbl>
```

Anmerkung: Die Referenzen werden bei jeder Neuerstellung der Objekte anders sein. Entsprechend werden die mit `lobstr::ref()` ausgegebenen Referenzen nicht mit denen in der Abbildung übereinstimmen.

---

Listen

**Dataframes**

Übungen und Selbstkontrollfragen

## Übersicht

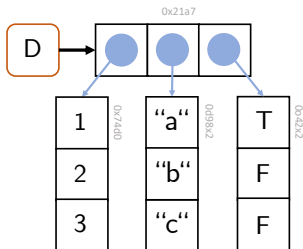
- Dataframes sind die zentrale Datenstruktur in R.
- Dataframes stellt man sich am besten als Tabelle vor.
- Die Zeilen und Spalten der Tabelle haben Namen.

	Control	drug1	drug2L	drug2R	delta1	delta2L	delta2R
1	0.6	1.3	2.5	2.1	0.7	1.9	1.5
2	3.0	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	0.0
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	8.0	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	0.0	4.6	4.7
10	2.9	4.9	6.3	6.4	2.0	3.4	3.5

## 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.

# Dataframes

## 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
```

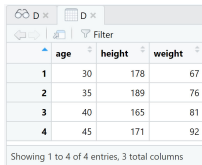
```
> [1] 3
```

# Dataframes

## Charakterisierung

`View()` öffnet den RStudio Data Viewer.

`View(D)`



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

Showing 1 to 4 of 4 entries, 3 total columns

`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
```

# Dataframes

## Indizierung

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
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"
```

# Dataframes

## Indizierung

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

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

# Dataframes

## Indizierung

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 fuer Zeilen, 2. Index fuer Spalten
```

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

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

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

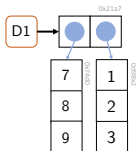
```
D[,c("x", "z")]                          # 1. Index fuer Zeilen, 2. Index fuer Spalten
```

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

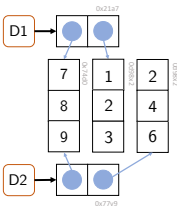
## Copy-on-modify

- 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

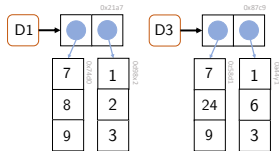
```
D1 = data.frame(  
  c(7,8,9),c(1,2,3))
```



```
D2 = D1  
D2[,1] = D2[,1] * 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. Was ist der 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$Student`?
8. Erläutern Sie den Begriff "Shallow Copy" einer Liste.
9. Beschreiben Sie in einer Übersicht die R Datenstruktur "Dataframe".
10. Erzeugen Sie einen Dataframe mit vier Spalten.
11. D sei ein Dataframe. Was geben `rownames(D)` und `colnames(D)` an?
12. D sei ein Dataframe. Was ist der Unterschied zwischen `D[1]` und `D[1,1]`?
13. D sei ein Dataframe. Was bedeutet dann `D$Student`?
14. Erläutern Sie das Copy-on-modify Prinzip für Dataframes.