LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Statistische Software (R-Vertiefung)

Paul Fink, M.Sc.

Institut für Statistik
Ludwig-Maximilians-Universität München

Vektoren, Matrizen, Arrays, Listen und Data Frames

Datentypen in R

Das einfachste Datenobjekt ist ein Vektor mit Elementen des Typs

- **numeric**: ganzzahlige oder Gleitkomma-Werte,
- **character**: beliebige Zeichen,
- **logical**: die Zustände `TRUE` und `FALSE`,
- **list**: ein Objekt beliebigen Typs, auch wieder eine Liste (rekursive Datenstrukturen!).

Jeder Vektor hat eine Länge (`length()`) und der Typ kann mittels `mode()` festgestellt werden.

Vektoren unterschiedleichen Typs

Vektor vom mit `mode` **numeric**.

```
> num1 <- c(1.64, 1.96, 2.71, 3.14, 3.84)
> num1
[1] 1.64 1.96 2.71 3.14 3.84
```

Vektor vom mit `mode` **character**.

```
> char1 <- c("Hallo", "Welt")
> char1
[1] "Hallo" "Welt"
```

Vektor vom mit `mode` **logical**. Bemerkung: Logische Vektoren können auch in arithmetischen Operationen verwendet werden, dabei wird der Wert `TRUE` in 1 und der Wert `FALSE` in 0 umgewandelt.

```
> logic1 <- c(TRUE, FALSE, FALSE, TRUE)
> logic1
[1]  TRUE FALSE FALSE  TRUE
> sum(logic1)
[1] 2
```


Vektoren unterschiedleichen Typs

Ein Objekt mit `mode` **list** kann einfache Vektoren, wie auch komplexere Objekte unterschiedlicher Klassen als Elemente enthalten.

```
> l1 <- list(numeric = num1, character = char1,
+   logical = logic1)
> lapply(l1, mode)
$numeric
[1] "numeric"

$character
[1] "character"

$logical
[1] "logical"
```

Vektoren unterschiedleichen Typs

```
> l2 <- list(numeric = num1, character = char1,
+   logical = logic1, list = l1)
> lapply(l2, mode)
$numeric
[1] "numeric"

$character
[1] "character"

$logical
[1] "logical"

$list
[1] "list"
```


Grundlegende Operatoren und Funktionen

Aufruf der Hilfeseiten zu grundlegende Operatoren und Funktionen:

`?Arithmetic`	Grundlegende Operatoren für numerische Vektoren
`?Logic`	Operatoren für logische Vektoren
`?log`	Logarithmus und Exponens
`?Trig`	Trigonnometrische Funktionen
`?Special`	Z.B. Binomialkoeffizienten, Fakultät, etc.

Konstanten

Übersicht einiger Konstanten:

`pi`	Die Zahl π
`Inf`, `-Inf`	$\infty, -\infty$
`NaN`	Not a Number: z.B. 0/0[1] NaN
`NA`	Not available: fehlende Werte
`NULL`	leere Menge

Fünf Arten des Zugriffs auf Vektorelemente

1. Vektor von positiven Zahlen (`letters` und `LETTERS` liefern die 26 Klein– bzw. Großbuchstaben des Alphabets zurück)

```
> letters[1:3]
[1] "a" "b" "c"
> letters[ c(2,4,6) ]
[1] "b" "d" "f"
```

Fünf Arten des Zugriffs auf Vektorelemente

2. Logischer Vektor

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x[ (x>5) ]
[1]  6  7  8  9 10
> x[ (x%%2==0) ]
[1]  2  4  6  8 10
> x[(x%%2==1)]
[1] 1 3 5 7 9
> x[5] <- NA
> y <- x[ !is.na(x) ]
> y
[1]  1  2  3  4  6  7  8  9 10
> mean(x)
[1] NA
> mean(y)
[1] 5.555556
```


Fünf Arten des Zugriffs auf Vektorelemente

3. Vektor von negativen Zahlen

```
> x <- 1:10
> x[-(1:5)]
[1]  6  7  8  9 10
```

4. Vektor von Zeichenketten

   Die Elemente eines Vektors kann man mit Namen versehen. Mittels dieses Namens kann auf die Elemente zugegriffen werden.

```
> x <- c(Wasser=1, Saft=2, Limonade=3 )
> names(x)
[1] "Wasser"   "Saft"     "Limonade"
> x["Saft"]
Saft
   2
```


Fünf Arten des Zugriffs auf Vektorelemente

5. Leerer Index. Diesen haben wir bereits ständig verwendet!

```
> x <- 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> x[]
 [1]  1  2  3  4  5  6  7  8  9 10
```


Rechnen mit Vektoren

Wie in Linearer Algebra komponentenweise Addition und Subtraktion

```
> x <- 1:4
> y <- c(4,10,2,0)
> x + y
[1]  5 12  5  4
```

Achtung: Multiplikation/Division auch komponentenweise!!

```
> x * y
[1]  4 20  6  0
```

Rechnen mit Vektoren

R erlaubt auch Rechnen mit Vektoren unterschiedlicher Länge.

```
> x
[1] 1 2 3 4
> x + c(1, 2)
[1] 2 4 4 6
```

entspricht

```
> x + c(1, 2, 1, 2)
[1] 2 4 4 6
```

Fehlende Werte werden aus bestehenden „recycled“.
Funktioniert auch wenn Vektorlängen nicht Vielfache sind, allerdings mit Warnung

```
> x + c(1, 2, 4) # x + c(1, 2, 4, 1)
[1] 2 4 7 5

Warning message:
In x + c(1, 2, 4) :
  longer object length is not a multiple of shorter object length
```


Faktoren

Nominale oder ordinale Daten werden in R „Faktoren“ genannt. Intern ist dies ein ganzzahliger Vektor, wo jeder Zahl ein „Label“ zugeordnet ist. Zeichen-Vektoren sind keine nominale Variablen, können aber mit der Funktion `factor()` leicht in eine solche verwandelt werden. Eine kategoriale Variable zeichnet sich dadurch aus, dass sie (hier: endlich) viele Stufen, die sog. Faktorstufen besitzt. Vom Skalenniveau her kann es sich um eine nominale oder eine ordinale Variable handeln.

```
> x <- factor( c("Saft", "Saft", "Limonade", "Saft", "Wasser") )
> x
[1] Saft     Saft     Limonade Saft     Wasser
Levels: Limonade Saft Wasser
```

Die einzelnen Stufen werden dabei gemäß der lexikographischen Ordnung angelegt:

Limonade $<$ Saft $<$ Wasser

Faktoren

Den einzelnen Stufen werden Zahlenwerte zugeordnet, wie der `unclass` Befehl zeigt:

```
> unclass(x)
[1] 2 2 1 2 3
attr(,"levels")
[1] "Limonade" "Saft"     "Wasser"
```

Soll die Zuordnung anders geschehen, so kann das durch den Parameter `levels` erfolgen:

```
> x <- factor( c("Saft", "Saft", "Limonade", "Saft", "Wasser"),
+   levels=c("Saft", "Wasser", "Limonade") )
> x
[1] Saft     Saft     Limonade Saft     Wasser
Levels: Saft Wasser Limonade
> unclass(x)
[1] 1 1 3 1 2
attr(,"levels")
[1] "Saft"     "Wasser"   "Limonade"
> levels(x)
[1] "Saft"     "Wasser"   "Limonade"
```


Faktoren

Möchte man ein spezielles Level als erstes verwenden, so geht das mittels `relevel`

```
> x <- relevel(x, "Wasser")
> unclass(x)
[1] 2 2 3 2 1
attr(,"levels")
[1] "Wasser"   "Saft"     "Limonade"
```

Ein Vektor kann in einen Faktor mittels `as.factor` umgewandelt werden:

```
> x <- c(4,5,1,2,3,3,4,4,5,6)
> x <- as.factor(x)
> x
 [1] 4 5 1 2 3 3 4 4 5 6
Levels: 1 2 3 4 5 6
```

Geordnete Faktoren

Beispiel für einen geordneten Faktor:

```
> einkommen <- ordered( c("hoch", "hoch", "niedrig", "mittel", "mittel"),
+   levels=c("niedrig", "mittel", "hoch") )
> einkommen
[1] hoch    hoch    niedrig mittel  mittel
Levels: niedrig < mittel < hoch
> unclass(einkommen)
[1] 3 3 1 2 2
attr(,"levels")
[1] "niedrig" "mittel"  "hoch"
```

Es ist wichtig, nominale und ordinale Variablen als solche zu codieren, da viele Statistik-Funktionen diese von Zeichen-Vektoren unterscheiden. (Beim Datenimport wird dies für nominale Größen aber ohnedies fast immer automatisch erledigt.)

Folgen

Der Befehl `seq()`. Beispiele:

- Absteigende Sequenz mit gleicher Schrittweite

```
> seq(from=3, to=-2, by=-0.5)
 [1]  3.0  2.5  2.0  1.5  1.0  0.5  0.0 -0.5 -1.0 -1.5 -2.0
```

- Standardschrittweite ist $+1$ oder -1:

```
> seq(from=2, to=4)
[1] 2 3 4
> seq(from=4, to=2)
[1] 4 3 2
```

- Sequenzen mit vorgegebener Länge

```
> seq(to=10, length=10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(from=10, length=10)
 [1] 10 11 12 13 14 15 16 17 18 19
> seq(from=10, length=10, by=0.1)
 [1] 10.0 10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8 10.9
```


Folgen

- Zuorden eines Indexvektors

```
> x <- c(9,8,7,6)
> ind <- seq(along=x)
> ind
[1] 1 2 3 4
> x[ ind[2] ]
[1] 8
```


Wiederholungen

Der Befehl `rep()`.

- $n-$malige Wiederholung eines Objekts

```
> rep(3.5, times=10)
 [1] 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5 3.5
> x <- matrix(nrow=2, ncol=2, data=1:4, byrow=T)
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> rep(x, 3)
 [1] 1 3 2 4 1 3 2 4 1 3 2 4
> rep(1:4, 2)
[1] 1 2 3 4 1 2 3 4
```

- Jedes Objekt wird mehrmals hintereinander wiederholt:

```
> rep(1:4, each = 2)
[1] 1 1 2 2 3 3 4 4
> rep(1:4, each = 2, times = 3)
 [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
```

Wiederholungen

- Jedes Obkjekt wird mit variabler Anzahl wiederholt:

```
> rep(1:4, 2:5)
 [1] 1 1 2 2 2 3 3 3 3 4 4 4 4 4
> anz <- seq(from=2, to=8, by=2)
> anz
[1] 2 4 6 8
> rep(1:4, anz)
 [1] 1 1 2 2 2 2 3 3 3 3 3 3 4 4 4 4 4 4 4 4
```


Matrizen

In R wird eine 4×2-Matrix X z.B. durch folgenden Befehl erzeugt:

```
> x <- matrix( nrow = 4, ncol = 2,
+   data = c(1, 2, 3, 4, 5, 6, 7, 8) )
> x
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    7
[4,]    4    8
```

Auf ein einzelnes Element kann mittels der Notation `x[i,j]` zugegriffen werden:

```
> x[3, 2]
[1] 7
```


Matrizen

Wir erkennen:

- Der Parameter nrow legt die Zahl der Zeilen der Matrix fest.
- Der Paramater ncol legt die Zahl der Spalten der Matrix fest.
- Der Parameter data erlaubt die Vorbelegung der Matrix mit bestimmten Werten für die Elemente.
- Die Werte aus dem Parameter data werden spaltenweise in die Matrix geschrieben.

Matrizen

Wir können nun bestimmte *Eigenschaften* der Matrix abfragen:

```
> dim(x)
[1] 4 2
> nrow(x)
[1] 4
> ncol(x)
[1] 2
> mode(x)
[1] "numeric"
> attributes(x)
$dim
[1] 4 2
```

D.h., neben den Elementen selbst werden zusätzliche, abfragbare Eigenschaften im *Objekt* Matrix abgelegt (Dimension, Anzahl der Zeilen, Anzahl der Spalten, Typ der Elemente).

Matrizen

Der Hilfe zum Befehl `matrix` entnehmen wir, dass man durch das Argument `byrow = TRUE` eine zeilenweise Belegung der Matrix mit den Daten erreicht (beachte die andere Variante für das Argument `data`)

```
> x <- matrix( nrow = 4, ncol = 2,
+   data = 1:8, byrow=TRUE )
> x
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
```


Matrizen

Spaltenweise Vektoren und Matrizen verbinden mit `cbind`

```
> y <- c(12, 3, 4, 1)
> cbind(x, y)
          y
[1,] 1 2 12
[2,] 3 4  3
[3,] 5 6  4
[4,] 7 8  1
> cbind(y, y)
      y  y
[1,] 12 12
[2,]  3  3
[3,]  4  4
[4,]  1  1
```


Matrizen

Zeilenweise Vektoren und Matrizen verbinden mit `rbind`

```
> rbind(c(100, 0), x)
     [,1] [,2]
[1,]  100    0
[2,]    1    2
[3,]    3    4
[4,]    5    6
[5,]    7    8
> rbind(x, x)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    1    2
[6,]    3    4
[7,]    5    6
[8,]    7    8
```


Mehrdimensionale Arrays

Der Befehl `array()` erlaubt die Definition mehrdimensionaler Arrays mit Dimensionen auch größer als 2.

- Ein dreidimensionaler Array:

```
> x <- array(data = 1:12, dim = c(3, 2, 2) )
```

Mehrdimensionale Arrays

```
> x
, , 1

     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

, , 2

     [,1] [,2]
[1,]    7   10
[2,]    8   11
[3,]    9   12
> x[2, 2, 1]
[1] 5
> x[3, 1, 2]
[1] 9
```


Mehrdimensionale Arrays

- Es können Dimensionen "verschwinden", daher gibt es den `drop` Befehl:

```
> x[3, 1, ]
[1] 3 9
> x[3, 1, , drop=F]
, , 1

     [,1]
[1,]    3

, , 2

     [,1]
[1,]    9
```


Mehrdimensionale Arrays

```
> x[3, , ]
     [,1] [,2]
[1,]    3    9
[2,]    6   12
> x[3, , , drop=F]
, , 1

     [,1] [,2]
[1,]    3    6

, , 2

     [,1] [,2]
[1,]    9   12
```


Listen

Wie bereits erwähnt zeichnen sich Listen zunächst dadurch aus, dass ihre Elemente nicht vom gleichen Objekttyp sein müssen. Aber auch Matrizen und Arrays, wie wir sie bereits kennen gelernt haben, können als Elemente beispielsweise logische Werte (FALSE, TRUE), Zeichen oder Zeichenketten enthalten. Allerdings funktionieren dann natürlich Operatoren nicht mehr, die den numerischen Typ voraussetzen. Beispiel:

```
> x1 <- matrix(nrow = 2, ncol = 2, data = 1:4, byrow = TRUE)
> x2 <- matrix(nrow = 2, ncol = 2, data = 5:8, byrow = TRUE)
> x1 + x2
     [,1] [,2]
[1,]    6    8
[2,]   10   12
> x1[2, 1] <- "Hallo"
> x1
     [,1]    [,2]
[1,] "1"     "2"
[2,] "Hallo" "4"

> x1 + x2
```

Listen

```
Error in x1 + x2 : non-numeric argument to binary operator
```

Listen können beliebige Objekte enthalten, auch Objekte verschiedenen Typs. Beispielsweise können Listen als Objekte Matrizen enthalten:

```
> x1 <- matrix(nrow = 2, ncol = 2, data = 1:4, byrow = TRUE)
> x2 <- matrix(nrow = 2, ncol = 2, data = 5:8, byrow = TRUE)
> matlist <- list( x1, x2)
> matlist[[1]]
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> matlist[[2]]
     [,1] [,2]
[1,]    5    6
[2,]    7    8
```


Listen

Dies erlaubt zum Beispiel auch die Rückgabe mehrerer Ergebnisse in Funktionen:

```
> multi_matrix_function <- function(x1, x2){
+   s <- x1 + x2
+   d <- x1 - x2
+   return( list(Summe = s, Differenz = d) )
+   }
> m <- multi_matrix_function(x1, x2)
> m$Summe
     [,1] [,2]
[1,]    6    8
[2,]   10   12
> m$Differenz
     [,1] [,2]
[1,]   -4   -4
[2,]   -4   -4
```


Listen

Die Funktion liefert die Summe und Differenz zweier Matrizen zurück. Die Liste, `list()`, wird dazu verwendet, beide Ergebnisse zurückzugeben. Dabei kann jedes Element einer Liste mit einem Namen versehen werden (hier "Summe" und "Differenz"), mittels dessen auf die Elemente der Liste u.a. zugegriffen werden kann. Manchmal geht auch

```
> m[1]
$Summe
     [,1] [,2]
[1,]    6    8
[2,]   10   12
> m[2]
$Differenz
     [,1] [,2]
[1,]   -4   -4
[2,]   -4   -4
```

Hinweis: besser sollte in diesem Fall der `[[]]` Operator verwendet werden, siehe unten.

Listen

Beispiel einer Liste,die verschiedene Objekte enthält:

```
> l1 <- list( c("Wasser", "Saft", "Limonade"), rep(1:4, each = 2),
+   matrix(data = 5:8, nrow = 2, ncol = 2, byrow=TRUE) )
> l1
[[1]]
[1] "Wasser"   "Saft"     "Limonade"

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

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

Listen

Der Zugriff auf die Elemente einer Liste sollte über den [[]] Operator erfolgen. Zwar liefert

```
> l1[1]
[[1]]
[1] "Wasser"   "Saft"     "Limonade"
```

noch das gewünschte Ergebnis, aber

```
> l1[1][2]
[[1]]
NULL
```

gibt NULL statt "Saft" zurück, während

```
> l1[[1]][2]
[1] "Saft"
```

schließlich das gewünschte Ergebnis liefert.

Datenmatrix

Die wohl wichtigste Struktur zur Haltung von Daten im üblichen Rechteckschema, wo die Beobachtungen in den Zeilen und die Variablen in den Spalten dargestellt werden, ist die Datenmatrix. In R wird diese `data.frame` genannt.

Data frames sind spezielle Listen, deren Elemente wiederum Vektoren gleicher Länge sind. Data frames sind *die* typische Datenstruktur in R. Data frames können komplette Datensätze aufnehmen, die (meist) mit anderen Programmen erstellt wurden (Spreadsheet– Dateien, SPSS–Dateien, tab–delimited ASCII Dateien, etc.).

Datenmatrix

Ein data.frame *painters* existiert in der library MASS (hier nur ein Ausschnitt des Datensatzes)

```
> library(MASS)
> painters
```

	Composition	Drawing	Colour	Expression	School
Da Udine	10	8	16	3	A
Da Vinci	15	16	4	14	A
Del Piombo	8	13	16	7	A
Del Sarto	12	16	9	8	A
Fr. Penni	0	15	8	0	A

Datenmatrix

- Die Namen der Maler dienen in diesem Fall der Identifikation der Zeilen, d.h. jeder Zeile wird der Namen des entsprechenden Malers zugeordnet. Diese Namen stellen aber keine Variable des Datensatzes dar! Hier ein Ausschnitt dieser Zeilennamen:

```
> rownames(painters)
 [1] "Da Udine"        "Da Vinci"        "Del Piombo"      "Del Sarto"
 [5] "Fr. Penni"       "Guilio Romano"   "Michelangelo"    "Perino del Vaga"
 [9] "Perugino"        "Raphael"         "F. Zucarro"      "Fr. Salviata"
[13] "Parmigiano"      "Primaticcio"     "T. Zucarro"      "Volterra"
[17] "Barocci"         "Cortona"         "Josepin"         "L. Jordaens"
[21] "Testa"           "Vanius"          "Bassano"         "Bellini"
[25] "Giorgione"       "Murillo"         "Palma Giovane"   "Palma Vecchio"
[29] "Pordenone"       "Tintoretto"      "Titian"          "Veronese"
[33] "Albani"          "Caravaggio"      "Corregio"        "Domenichino"
[37] "Guercino"        "Lanfranco"       "The Carraci"     "Durer"
[41] "Holbein"         "Pourbus"         "Van Leyden"      "Diepenbeck"
[45] "J. Jordaens"     "Otho Venius"     "Rembrandt"       "Rubens"
[49] "Teniers"         "Van Dyck"        "Bourdon"         "Le Brun"
[53] "Le Suer"         "Poussin"
```

Datenmatrix

- Der Datensatz enthält vier numerische Variablen (Composition, Drawing, Colour und Expression), sowie eine Faktorvariable (School).

```
> is.numeric(painters$School)
[1] FALSE
> is.numeric(painters$Drawing)
[1] TRUE
> is.factor(painters$School)
[1] TRUE
> is.factor(painters$Drawing)
[1] FALSE
> colnames(painters)
[1] "Composition" "Drawing"     "Colour"      "Expression"  "School"
```


Datenmatrix

- Mittels der `summary` Funktion kann man eine schnelle Übersicht über deskriptive Maße der einzelnen Variablen gewinnen:

```
> summary(painters)
  Composition       Drawing          Colour        Expression        School
 Min.   : 0.00   Min.   : 6.00   Min.   : 0.00   Min.   : 0.000   A      :10
 1st Qu.: 8.25   1st Qu.:10.00   1st Qu.: 7.25   1st Qu.: 4.000   D      :10
 Median :12.50   Median :13.50   Median :10.00   Median : 6.000   E      : 7
 Mean   :11.56   Mean   :12.46   Mean   :10.94   Mean   : 7.667   G      : 7
 3rd Qu.:15.00   3rd Qu.:15.00   3rd Qu.:16.00   3rd Qu.:11.500   B      : 6
 Max.   :18.00   Max.   :18.00   Max.   :18.00   Max.   :18.000   C      : 6
                                                                  (Other): 8
```

Es fällt auf: Die Kategorien `F` und `H` der Variablen School, welche jeweils 4 mal vorkommen, werden unter der Kategorie `Other` mit der entsprechenden Häufigkeit angegeben. D.h. für die Variable School werden (nur) die 6 häufigsten Ausprägungen angegeben.

Datenmatrix

- Einzelne Variablen kann man direkt über ihren Namen (Erinnerung: Die Namen sind abfragbar über `colnames()`) ansprechen (hier nur Ausschnitt)

```
> painters$School

 [1] A A A A A A A A A A B B B B B B C C C C C C D D D D D D ...
Levels: A B C D E F G H
```

Die `summary` Funktion liefert hier auch eine ausführliche Häufigkeitstabelle:

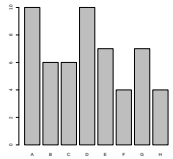
```
> summary(painters$School)
 A  B  C  D  E  F  G  H
10  6  6 10  7  4  7  4
```


Datenmatrix

Zur Auflockerung:

```
> plot(painters$School)
```

liefert



- Test, ob ein data frame vorliegt:

```
> is.data.frame(painters)
[1] TRUE
```

Datenmatrix

- Der `attach()` Befehl erlaubt, einen Datensatz in den Suchpfad mitaufzunehmen. Man kann dann direkt auf die Variablen zugreifen, also ohne das Präfix `painters$`. Mit `detach()` wird der data frame wieder aus dem Suchpfad enfernt.

Datenmatrix

```
> search()
 [1] ".GlobalEnv"        "package:MASS"      "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
> attach(painters)
> search()
 [1] ".GlobalEnv"        "painters"          "package:MASS"
 [4] "package:stats"     "package:graphics"  "package:grDevices"
 [7] "package:utils"     "package:datasets"  "package:methods"
[10] "Autoloads"         "package:base"
> summary(School)
 A  B  C  D  E  F  G  H
10  6  6 10  7  4  7  4
> detach(painters)
> search()
 [1] ".GlobalEnv"        "package:MASS"      "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"

> summary(School)

Error in summary(School) : object 'School' not found
```


Datenmatrix

- Teilmengen eines data frame erhält man mit `subset()` oder dem darauf folgenden äquivalenten Befehl:

Datenmatrix

```
> subset(painters, School == 'F')
          Composition Drawing Colour Expression School
Durer               8      10     10          8      F
Holbein             9      10     16         13      F
Pourbus             4      15      6          6      F
Van Leyden          8       6      6          4      F
> painters[ painters[["School"]] == "F", ]
          Composition Drawing Colour Expression School
Durer               8      10     10          8      F
Holbein             9      10     16         13      F
Pourbus             4      15      6          6      F
Van Leyden          8       6      6          4      F
> subset(painters, Composition <= 6)
             Composition Drawing Colour Expression School
Fr. Penni              0      15      8          0      A
Perugino               4      12     10          4      A
Bassano                6       8     17          0      D
Bellini                4       6     14          0      D
Murillo                6       8     15          4      D
Palma Vecchio          5       6     16          0      D
Caravaggio             6       6     16          0      E
Pourbus                4      15      6          6      F
```

Datenmatrix

- Uninteressante Spalten können in der Teilmenge ausgeschlossen werden:

```
> subset(painters, School == "F", select = c(-3, -5) )
          Composition Drawing Expression
Durer               8      10          8
Holbein             9      10         13
Pourbus             4      15          6
Van Leyden          8       6          4
```


Datenmatrix

Die dritte und die fünfte Spalte (Colour und School) werden nicht selektiert.

- Der Operator `%in%` erlaubt komplexe Abfragen. Beispiel:

```
> subset(painters, Drawing %in% c(6, 7, 8, 9) )
             Composition Drawing Colour Expression School
Da Udine              10       8     16          3      A
Bassano               6        8     17          0      D
Bellini               4        6     14          0      D
Giorgione             8        9     18          4      D
Murillo               6        8     15          4      D
Palma Giovane        12        9     14          6      D
Palma Vecchio         5        6     16          0      D
Caravaggio            6        6     16          0      E
Van Leyden            8        6      6          4      F
J. Jordaens          10        8     16          6      G
Rembrandt            15        6     17         12      G
Bourdon              10        8      8          4      H
```


Datenmatrix

- Der Befehl `split` erlaubt das Aufteilen eines Datensatzes gemäß der Ausprägungen einer Variablen. Diese sollte möglichst eine Faktorvariable sein. Beispiel:

```
> splitted <- split(painters, painters$School)
> splitted
```


Datenmatrix

```
$A
             Composition Drawing Colour Expression School
Da Udine              10       8     16          3      A
Da Vinci              15      16      4         14      A
Del Piombo             8      13     16          7      A
Del Sarto             12      16      9          8      A
Fr. Penni              0      15      8          0      A
Guilio Romano         15      16      4         14      A
Michelangelo           8      17      4          8      A
Perino del Vaga       15      16      7          6      A
Perugino               4      12     10          4      A
Raphael               17      18     12         18      A

$B
             Composition Drawing Colour Expression School
F. Zucarro            10      13      8          8      B
Fr. Salviata          13      15      8          8      B
Parmigiano            10      15      6          6      B
Primaticcio           15      14      7         10      B
T. Zucarro            13      14     10          9      B
Volterra              12      15      5          8      B

$C
...
```

Datenmatrix

Bemerkung: man muss hier `painters$School` verwenden, wenn der Datensatz nicht "attached" ist. Die Objekte `splitted$A` bis `splitted$H` sind jetzt selbst data frames:

```
> is.data.frame(splitted$A)
[1] TRUE
```


Aufgaben

Laden Sie den data.frame painters aus der library MASS.

1. Nehmen Sie die Malernamen als Variable in den Datensatz auf und geben Sie dieser einen sinnvollen Namen.

2. Splitten Sie den Datensatz nach der Variablen `School`.

3. Es gibt auch einen `merge()` Befehl. Fassen Sie alle Maler mit `School` $\in$ ("B","C","D"), deren `Composition` Variable größer ist als 10, zu einem neuen data.frame zusammen.

4. Verwenden Sie statt `merge()` einen anderen, schon bekannten Befehl.