Data Structures II

Matrices

Matrices are comparable to vectors in that they are only allowed to contain elements of the same mode. In contrast to vectors, they have an additional dimensionality attribute. A matrix is a special case of an array (see below) having exactly two dimensions, i.e., having rows and columns.

Matrix creation and description

The simplest way to create a matrix is by using the `matrix()` command. Matrices are column-oriented:

```r
(X <- matrix(1:10, ncol = 2)) # matrix is filled column-wise, number of rows is
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
```

```r
#determined automatically (-> 5x2 matrix)
(Y <- matrix(1:8, ncol = 3)) # recycling rule
```

```
## Warning in matrix(1:8, ncol = 3): data length [8] is not a sub-multiple or
## multiple of the number of rows [3]
```

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

```r
(X <- matrix(1:10, ncol = 2, byrow = TRUE)) # elements row-wise
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
## [5,]    9   10
```

```r
matrix(1:10, ncol = 3, nrow = 3)
```

```
## Warning in matrix(1:10, ncol = 3, nrow = 3): data length [10] is not a
## sub-multiple or multiple of the number of rows [3]
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```r
matrix(1:3, nrow = 3, ncol = 4)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    1    1
## [2,]    2    2    2    2
## [3,]    3    3    3    3
```

```
matrix(1:3, nrow = 3)
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
```

```
mode(X)
```

```
## [1] "numeric"
```

```
typeof(X)
```

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

```
str(X) # shows indices of rows and columns
```

```
##  int [1:5, 1:2] 1 3 5 7 9 2 4 6 8 10
```

Above, we created a matrix by supplying a vector of values (from 1 to 10). We may also coerce a vector by using the `as.matrix()` function or by assigning the dimensions attribute:

```
(x <- 1:12)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12
```

```
is.vector(x)
```

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

```
is.matrix(x)
```

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

```
(y <- as.matrix(x))
```

```
##       [,1]
##  [1,]    1
##  [2,]    2
##  [3,]    3
##  [4,]    4
##  [5,]    5
##  [6,]    6
##  [7,]    7
##  [8,]    8
##  [9,]    9
## [10,]   10
## [11,]   11
## [12,]   12
```

```
is.vector(y)
```

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

```
is.matrix(y)
```

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

```r
str(y) # 12x1 matrix
```

```
##  int [1:12, 1] 1 2 3 4 5 6 7 8 9 10 ...
```

```r
dim(x) <- c(4, 3)
x # 4x3 matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```r
is.vector(x)
```

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

```r
is.matrix(x)
```

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

```r
str(x)
```

```
##  int [1:4, 1:3] 1 2 3 4 5 6 7 8 9 10 ...
```

```r
dim(x) <- c(3, 4)
x
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

`rbind()` and `cbind()` are used to combine matrices row- or column-wise:

```r
X
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
## [4,]    7    8
## [5,]    9   10
```

```r
(Z <- matrix(11:20, ncol = 2))
```

```
##      [,1] [,2]
## [1,]   11   16
## [2,]   12   17
## [3,]   13   18
## [4,]   14   19
## [5,]   15   20
```

```r
rbind(X, Z)
```

```
##       [,1] [,2]
##  [1,]    1    2
##  [2,]    3    4
##  [3,]    5    6
##  [4,]    7    8
##  [5,]    9   10
##  [6,]   11   16
##  [7,]   12   17
##  [8,]   13   18
##  [9,]   14   19
## [10,]   15   20
```

```r
cbind(X, Z)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2   11   16
## [2,]    3    4   12   17
## [3,]    5    6   13   18
## [4,]    7    8   14   19
## [5,]    9   10   15   20
```

```r
rbind(X, Y) # number of columns (and correspondingly, rows when using cbind())
```

```
## Error in rbind(X, Y): number of columns of matrices must match (see arg 2)
```

```r
# must match
```

Special types of matrices

```r
matrix(0, 3, 3) # Zero matrix
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    0    0
```

```r
diag(4) # Identity/Unit matrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    1    0    0
## [3,]    0    0    1    0
## [4,]    0    0    0    1
```

```r
diag(1:4) # Diagonal matrix
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    0    2    0    0
## [3,]    0    0    3    0
## [4,]    0    0    0    4
```

```r
Y
```

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

```r
diag(Y)
```

```
## [1] 1 5 1
```

```r
diag(diag(Y))
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    5    0
## [3,]    0    0    1
```

```
diag(Y) <- 44
Y
```

```
##      [,1] [,2] [,3]
## [1,]   44    4    7
## [2,]    2   44    8
## [3,]    3    6   44
```

```
diag(1:3, nrow = 4, ncol = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    2    0
## [3,]    0    0    3
## [4,]    0    0    0
```

```
diag(1:3, nrow = 4, ncol = 6)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    0    0    0    0    0
## [2,]    0    2    0    0    0    0
## [3,]    0    0    3    0    0    0
## [4,]    0    0    0    1    0    0
```

Extraction and replacement/Subsetting

```
X[1, 2] # select element (1, 2)
```

```
## [1] 2
```

```
X[1, ]  # select row 1
```

```
## [1] 1 2
```

```
is.vector(X[1, ]) # coerced to a vector!
```

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

```
is.matrix(X[1, ])
```

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

```
X[1, , drop = FALSE]
```

```
##      [,1] [,2]
## [1,]    1    2
```

```
is.vector(X[1, , drop = F]) # no coercion!
```

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

```
is.matrix(X[1, , drop = F])
```

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

```
X[, 2]  # select column 2
```

```
## [1]  2  4  6  8 10
```

```
X[2, 2] <- 44 # replace element (2, 2)
X
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3   44
## [3,]    5    6
## [4,]    7    8
## [5,]    9   10
```

```
X[, 2] <- X[, 2] * 10 # replace column 2
X
```

```
##      [,1] [,2]
## [1,]    1   20
## [2,]    3  440
## [3,]    5   60
## [4,]    7   80
## [5,]    9  100
```

```
X[, c(T, F)] # only first column
```

```
## [1] 1 3 5 7 9
```

```
X[c(T, F), ] # first, third, fifth row ( = T, F, T, F, T -> recycling rule)
```

```
##      [,1] [,2]
## [1,]    1   20
## [2,]    5   60
## [3,]    9  100
```

Misc. functions

Dimensionality

It is possible to get information about the dimensionality of the matrix using the following functions:

```
nrow(X) # number of rows
```

```
## [1] 5
```

```
ncol(X) # number of columns
```

```
## [1] 2
```

```
dim(X) # dimensionality
```

```
## [1] 5 2
```

```
length(X) # number of elements (= product of the dimensions)
```

```
## [1] 10
```

```
length(dim(X)) # 2 for matrices
```

```
## [1] 2
```

Naming

Each row and each column can be named:

```
X
```

```
##      [,1] [,2]
## [1,]    1   20
## [2,]    3  440
## [3,]    5   60
## [4,]    7   80
## [5,]    9  100
```

```
rownames(X)
```

```
## NULL
```

```
colnames(X)
```

```
## NULL
```

```
rownames(X) <- paste("Person", 1:5)
rownames(X)
```

```
## [1] "Person 1" "Person 2" "Person 3" "Person 4" "Person 5"
```

```
X
```

```
##          [,1] [,2]
## Person 1    1   20
## Person 2    3  440
## Person 3    5   60
## Person 4    7   80
## Person 5    9  100
```

```
colnames(X) <- paste("Variable", 1:2)
X
```

```
##          Variable 1 Variable 2
## Person 1          1         20
## Person 2          3        440
## Person 3          5         60
## Person 4          7         80
## Person 5          9        100
```

```
dimnames(X) # list (see below) of row and column names
```

```
## [[1]]
## [1] "Person 1" "Person 2" "Person 3" "Person 4" "Person 5"
##
## [[2]]
## [1] "Variable 1" "Variable 2"
```

```
X["Person 1", ]
```

```
## Variable 1 Variable 2
##          1         20
```

```
X[, "Variable 1"]
```

```
## Person 1 Person 2 Person 3 Person 4 Person 5
##        1        3        5        7        9
```

```
X["Person 1", "Variable 1"]
```

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

Matrix arithmetic

```
(X <- matrix(1:10, ncol = 2))
```

```
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
```

```
(Y <- t(X)) # transpose matrix
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
```

```
X * Y  # "*" is not the operator used for matrix multiplication!
```

```
## Error in X * Y: non-conformable arrays
```

```
X %*% Y # Matrix multiplication, gives matrix product (1 * 1 + 6 * 6 = 37 etc.)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   37   44   51   58   65
## [2,]   44   53   62   71   80
## [3,]   51   62   73   84   95
## [4,]   58   71   84   97  110
## [5,]   65   80   95  110  125
```

```
crossprod(X, X) # Matrix Crossproduct
```

```
##      [,1] [,2]
## [1,]   55  130
## [2,]  130  330
```

```
# (1 * 1 + 2 * 2 + 3 * 3 + 4 * 4 + 5 * 5 = 55 etc.)
X * X # Hadamard product (1 * 1 = 1, 2 * 2 = 4 etc.)
```

```
##      [,1] [,2]
## [1,]    1   36
## [2,]    4   49
## [3,]    9   64
## [4,]   16   81
## [5,]   25  100
```

Variance-covariance matrix

```
set.seed(1)
(X <- matrix(rnorm(15), nrow = 5))
```

```
##            [,1]       [,2]       [,3]
## [1,] -0.6264538 -0.8204684  1.5117812
## [2,]  0.1836433  0.4874291  0.3898432
## [3,] -0.8356286  0.7383247 -0.6212406
## [4,]  1.5952808  0.5757814 -2.2146999
## [5,]  0.3295078 -0.3053884  1.1249309
```

```r
rownames(X) <- paste("Per.", 1:5)
colnames(X) <- paste("Var.", 1:3)
X
```

```
##            Var. 1     Var. 2     Var. 3
## Per. 1 -0.6264538 -0.8204684  1.5117812
## Per. 2  0.1836433  0.4874291  0.3898432
## Per. 3 -0.8356286  0.7383247 -0.6212406
## Per. 4  1.5952808  0.5757814 -2.2146999
## Per. 5  0.3295078 -0.3053884  1.1249309
```

```r
mean(X[, 1]) # mean of the first variable
```

```
## [1] 0.1292699
```

```r
var(X[, 1]) # Variance of the first variable
```

```
## [1] 0.9235968
```

```r
cov(X[, 1], X[, 2]) # joint variability of the first two variables
```

```
## [1] 0.1792734
```

```r
VCM <- cov.wt(X)
VCM
```

```
## $cov
##            Var. 1     Var. 2     Var. 3
## Var. 1  0.9235968  0.1792734 -0.8858445
## Var. 2  0.1792734  0.4473392 -0.7883769
## Var. 3 -0.8858445 -0.7883769  2.2466246
##
## $center
##     Var. 1     Var. 2     Var. 3
## 0.12926990 0.13513567 0.03812297
##
## $n.obs
## [1] 5
```

Standardize variables (Standard/z score)

```r
(X <- matrix(1:15, ncol = 3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```r
mean(X[, 1])
```

```
## [1] 3
```

```r
var(X[, 1])
```

```
## [1] 2.5
```

```
(Y <- scale(X)) # variables now have mean 0 and variance 1
```

```
##            [,1]       [,2]       [,3]
## [1,] -1.2649111 -1.2649111 -1.2649111
## [2,] -0.6324555 -0.6324555 -0.6324555
## [3,]  0.0000000  0.0000000  0.0000000
## [4,]  0.6324555  0.6324555  0.6324555
## [5,]  1.2649111  1.2649111  1.2649111
## attr(,"scaled:center")
## [1]  3  8 13
## attr(,"scaled:scale")
## [1] 1.581139 1.581139 1.581139
```

```
mean(Y[, 1])
```

```
## [1] 0
```

```
var(Y[, 1])
```

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

```
(Z <- scale(X, scale = FALSE)) # only subtract mean
```

```
##      [,1] [,2] [,3]
## [1,]   -2   -2   -2
## [2,]   -1   -1   -1
## [3,]    0    0    0
## [4,]    1    1    1
## [5,]    2    2    2
## attr(,"scaled:center")
## [1]  3  8 13
```

```
mean(Z[, 1])
```

```
## [1] 0
```

```
var(Z[, 1])
```

```
## [1] 2.5
```

Arrays

Arrays are data structures that can have 1, 2 or more dimensions. As stated in the documentation, *“It is simply a vector which is stored with additional attributes giving the dimensions (attribute”dim”) and optionally names for those dimensions (attribute “dimnames”)“*.

Array creation and description

Arrays are created similar to matrices, using function `array()`:

```
(X <- array(1:24, dim = 2:4)) # 2x3x4 array: 2 rows, 3 columns, 4 "strata"
```

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

```
## , , 2
## 
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
## 
## , , 3
## 
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
## 
## , , 4
## 
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   24
```

```
(X <- array(1:12, dim = 2:4)) # 2 rows, 3 columns, 4 "strata"; recycling rule
```

```
## , , 1
## 
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
## 
## , , 2
## 
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
## 
## , , 3
## 
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
## 
## , , 4
## 
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

```
mode(X)
```

```
## [1] "numeric"
```

```
typeof(X)
```

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

```
str(X)
```

```
##  int [1:2, 1:3, 1:4] 1 2 3 4 5 6 7 8 9 10 ...
```

Subsetting

```r
X[1, ,]
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    7    1    7
## [2,]    3    9    3    9
## [3,]    5   11    5   11
```

```r
X[, 2, 2]
```

```
## [1]  9 10
```

```r
X[1, 2, 3]
```

```
## [1] 3
```

Array transposition

```r
aperm(X, 3:1) # 4 rows, 3 columns, 2 "strata"
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    7    9   11
## [3,]    1    3    5
## [4,]    7    9   11
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    2    4    6
## [2,]    8   10   12
## [3,]    2    4    6
## [4,]    8   10   12
```

```r
aperm(X, c(1, 3, 2)) # 2 rows, 4 columns, 3 "strata"
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    7    1    7
## [2,]    2    8    2    8
##
## , , 2
##
##      [,1] [,2] [,3] [,4]
## [1,]    3    9    3    9
## [2,]    4   10    4   10
##
## , , 3
##
##      [,1] [,2] [,3] [,4]
## [1,]    5   11    5   11
## [2,]    6   12    6   12
```

Naming

```r
dim(X)
```

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

```r
dimnames(X)
```

```
## NULL
```

```r
dimnames(X)[1] <- list(paste0("Z", 1:2))
dimnames(X)
```

```
## [[1]]
## [1] "Z1" "Z2"
##
## [[2]]
## NULL
##
## [[3]]
## NULL
```

```r
X
```

```
## , , 1
##
##    [,1] [,2] [,3]
## Z1    1    3    5
## Z2    2    4    6
##
## , , 2
##
##    [,1] [,2] [,3]
## Z1    7    9   11
## Z2    8   10   12
##
## , , 3
##
##    [,1] [,2] [,3]
## Z1    1    3    5
## Z2    2    4    6
##
## , , 4
##
##    [,1] [,2] [,3]
## Z1    7    9   11
## Z2    8   10   12
```

Data Frames

In contrast to other programming languages, data structures were designed to represent data in a natural way, useful for statistical estimation, testing, and modeling. The most important type of data structure in this regard are *data frames*.

Data frames are similar to matrices in that they are 2-dimensional arrays of elements, but they have some specific features: First of all, the rows always represent observational units (persons, companies, countries, . . .) and the columns represent variables (in the statistical sense, i.e., characteristics that are used to describe the observational units, e.g., height, gender, salary, . . .). It is not possible to represent observational units by columns and variables by rows, respectively! Variables in a data frame are vectors that are allowed to have different modes (numerical, logical, string vectors). Categorical variables enter data frames as factors.

Data frame creation

We could generate a data frame by coercion of a matrix:

```
(X <- matrix(1:15, ncol = 3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
Y <- data.frame(X)
Y
```

```
##   X1 X2 X3
## 1  1  6 11
## 2  2  7 12
## 3  3  8 13
## 4  4  9 14
## 5  5 10 15
```

The entries are (and look) the same, but variable names (`X1`, `X2`, `X3`) and names of the observational units (`1` to `5`) were created automatically.

```
attributes(Y)
```

```
## $names
## [1] "X1" "X2" "X3"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3 4 5
```

```
names(Y)
```

```
## [1] "X1" "X2" "X3"
```

```
colnames(Y)
```

```
## [1] "X1" "X2" "X3"
```

```
rownames(Y)
```

```
## [1] "1" "2" "3" "4" "5"
```

```
dimnames(Y)
```

```
## [[1]]
## [1] "1" "2" "3" "4" "5"
##
## [[2]]
## [1] "X1" "X2" "X3"
```

```
mode(Y)
```

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

```
typeof(Y)
```

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

```
str(Y)
```

```
## ’data.frame’:    5 obs. of  3 variables:
##  $ X1: int  1 2 3 4 5
##  $ X2: int  6 7 8 9 10
##  $ X3: int  11 12 13 14 15
```

```
is.matrix(Y) # a data frame is not a matrix by this test!
```

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

```
is.data.frame(Y)
```

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

```
is.list(Y)
```

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

A data frame is a list of variables of the same number of rows with unique row names.

```
df1 <- letters[1:10] # letters a...j
df2 <- 1:10
df3 <- 10:1
str(df1)
```

```
##  chr [1:10] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
str(df2)
```

```
##  int [1:10] 1 2 3 4 5 6 7 8 9 10
```

```
# concatenate as data frame; object names are used as variable names:
(df <- data.frame(df1, df2, df3))
```

```
##    df1 df2 df3
## 1    a   1  10
## 2    b   2   9
## 3    c   3   8
## 4    d   4   7
## 5    e   5   6
## 6    f   6   5
## 7    g   7   4
## 8    h   8   3
## 9    i   9   2
## 10   j  10   1
```

```
str(df)
```

```
## ’data.frame’:    10 obs. of  3 variables:
##  $ df1: chr  "a" "b" "c" "d" ...
##  $ df2: int  1 2 3 4 5 6 7 8 9 10
##  $ df3: int  10 9 8 7 6 5 4 3 2 1
```

```
# convert character vectors to factors (default changed to FALSE in R 4.0.0):
(df <- data.frame(df1, df2, df3, stringsAsFactors = TRUE))
```

```
##    df1 df2 df3
## 1    a   1  10
## 2    b   2   9
## 3    c   3   8
## 4    d   4   7
## 5    e   5   6
## 6    f   6   5
## 7    g   7   4
## 8    h   8   3
## 9    i   9   2
## 10   j  10   1
```

```
colnames(df) <- c("let", "numbers.incr", "numbers.decr")
str(df)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ let         : Factor w/ 10 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10
##  $ numbers.incr: int  1 2 3 4 5 6 7 8 9 10
##  $ numbers.decr: int  10 9 8 7 6 5 4 3 2 1
```

We can add variables using `cbind()`:

```
LET <- LETTERS[1:10] # uppercase letters A...J
(df <- cbind(df, LET))
```

```
##    let numbers.incr numbers.decr LET
## 1    a            1           10   A
## 2    b            2            9   B
## 3    c            3            8   C
## 4    d            4            7   D
## 5    e            5            6   E
## 6    f            6            5   F
## 7    g            7            4   G
## 8    h            8            3   H
## 9    i            9            2   I
## 10   j           10            1   J
```

And we can add rows using `rbind()`:

```
(df <- rbind(df, list("j", 2000, 0, "J")))
```

```
##    let numbers.incr numbers.decr LET
## 1    a            1           10   A
## 2    b            2            9   B
## 3    c            3            8   C
## 4    d            4            7   D
## 5    e            5            6   E
## 6    f            6            5   F
## 7    g            7            4   G
## 8    h            8            3   H
## 9    i            9            2   I
## 10   j           10            1   J
## 11   j         2000            0   J
```

```
str(df)
```

```
## 'data.frame':    11 obs. of  4 variables:
##  $ let         : Factor w/ 10 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ numbers.incr: num  1 2 3 4 5 6 7 8 9 10 ...
##  $ numbers.decr: num  10 9 8 7 6 5 4 3 2 1 ...
##  $ LET         : chr  "A" "B" "C" "D" ...
```

```
str(rbind(df, c("j", 2000, 0, "J"))) # probably undesirable!
```

```
## 'data.frame':    12 obs. of  4 variables:
##  $ let         : Factor w/ 10 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ numbers.incr: chr  "1" "2" "3" "4" ...
##  $ numbers.decr: chr  "10" "9" "8" "7" ...
##  $ LET         : chr  "A" "B" "C" "D" ...
```

Extraction and replacement/Subsetting

Subsetting works similar to matrices (using the `[` operator), but has some peculiarities:

```
df[1, 2] # select element (1, 2)
```

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

```
df[1, ]  # select row 1
```

```
##   let numbers.incr numbers.decr LET
## 1   a            1           10   A
```

```
is.vector(df[1, ]) # not coerced to a vector
```

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

```
is.data.frame(df[1, ]) # still a data frame...
```

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

```
is.list(df[1, ]) # ...and a list as well
```

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

```
df[1, , drop = TRUE]
```

```
## $let
## [1] a
## Levels: a b c d e f g h i j
## 
## $numbers.incr
## [1] 1
## 
## $numbers.decr
## [1] 10
## 
## $LET
## [1] "A"
```

```
is.vector(df[1, , drop = T]) # coerced to a vector (of mode "list", hence not atomic)
```

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

```
is.data.frame(df[1, , drop = T]) # no longer a data frame...
```

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

```
is.list(df[1, , drop = T]) # ...but still a list
```

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

```
df[, 2]  # select column 2
```

```
##  [1]    1    2    3    4    5    6    7    8    9   10 2000
```

```
df[2, 2] <- c(44) # replace element (2, 2)
df
```

```
##    let numbers.incr numbers.decr LET
## 1    a            1           10   A
## 2    b           44            9   B
## 3    c            3            8   C
## 4    d            4            7   D
## 5    e            5            6   E
## 6    f            6            5   F
## 7    g            7            4   G
## 8    h            8            3   H
## 9    i            9            2   I
## 10   j           10            1   J
## 11   j         2000            0   J
```

```
df[, 2] <- df[, 2] * 10 # replace column 2
df
```

```
##    let numbers.incr numbers.decr LET
## 1    a           10           10   A
## 2    b          440            9   B
## 3    c           30            8   C
## 4    d           40            7   D
## 5    e           50            6   E
## 6    f           60            5   F
## 7    g           70            4   G
## 8    h           80            3   H
## 9    i           90            2   I
## 10   j          100            1   J
## 11   j        20000            0   J
```

```
df[, c(T, F)] # first and third column ( = T, F, T, F -> recycling rule)
```

```
##    let numbers.decr
## 1    a           10
## 2    b            9
## 3    c            8
## 4    d            7
## 5    e            6
## 6    f            5
## 7    g            4
## 8    h            3
## 9    i            2
## 10   j            1
## 11   j            0
```

```
df[c(T, F), ] # first, third, fifth, ... row ( = T, F, T, F, T. ... -> recycling rule)
```

```
##    let numbers.incr numbers.decr LET
## 1    a           10           10   A
## 3    c           30            8   C
## 5    e           50            6   E
## 7    g           70            4   G
## 9    i           90            2   I
## 11   j        20000            0   J
```

Using an index vector, we can change the positions of the variables:

```
df <- df[, 4:1]
df # "LET" now is the first variable etc.
```

```
##    LET numbers.decr numbers.incr let
## 1    A           10           10   a
## 2    B            9          440   b
## 3    C            8           30   c
## 4    D            7           40   d
## 5    E            6           50   e
## 6    F            5           60   f
## 7    G            4           70   g
## 8    H            3           80   h
## 9    I            2           90   i
## 10   J            1          100   j
## 11   J            0        20000   j
```

We could also use the [in combination with the variable name to access the variables, but when working with data frames, it is usually more convenient to use the dollar operator $ instead:

```
names(df)
```

```
## [1] "LET"          "numbers.decr" "numbers.incr" "let"
```

```
df[, "let"]
```

```
##  [1] a b c d e f g h i j j
## Levels: a b c d e f g h i j
```

```
df$let # the same
```

```
##  [1] a b c d e f g h i j j
## Levels: a b c d e f g h i j
```

This operation can also be combined with the [operator, e.g., in order to replace elements:

```
df$let[1] # First element of variable "let"
```

```
## [1] a
## Levels: a b c d e f g h i j
```

```
df$let[1] <- "b"
df$let
```

```
##  [1] b b c d e f g h i j j
## Levels: a b c d e f g h i j
```

Another possibility: We add the object to the search path using `attach()`:

```
let # not found
```

```
## Error in eval(expr, envir, enclos): object 'let' not found
```

```
attach(df)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##     LET
```

```
let
```

```
##  [1] b b c d e f g h i j j
## Levels: a b c d e f g h i j
```

When working with different objects, one may run into problems when using `attach()` repeatedly (e.g., because of identical variable names in different objects, e.g., "age", "sex" etc.). To avoid this, `detach()` should be used to remove the object from the search path again:

```
detach(df)
let
```

```
## Error in eval(expr, envir, enclos): object 'let' not found
```

Enter data

```
# create empty data frame 'newdat', open it and enter the variable names + values:
newdat <- edit(data.frame())
newdat
newdat <- edit(newdat) # edit entries
newdat
edit(newdat)
newdat
```

Misc. functions

```
nrow(df)
```

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

```
ncol(df)
```

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

```
dim(df)
```

```
## [1] 11  4
```

```
length(df) # number of variables, not number of elements!
```

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

```
head(df) # first 6 observations
```

```
##   LET numbers.decr numbers.incr let
## 1   A           10           10   b
## 2   B            9          440   b
## 3   C            8           30   c
## 4   D            7           40   d
## 5   E            6           50   e
## 6   F            5           60   f
```

```
tail(df) # last 6 observations
```

```
##    LET numbers.decr numbers.incr let
## 6    F            5           60   f
## 7    G            4           70   g
## 8    H            3           80   h
## 9    I            2           90   i
## 10   J            1          100   j
## 11   J            0        20000   j
```

```
head(df, 2) # first 2 observations
```

```
##   LET numbers.decr numbers.incr let
## 1   A           10           10   b
## 2   B            9          440   b
```

```
summary(df)
```

```
##      LET             numbers.decr   numbers.incr        let
##  Length:11          Min.   : 0.0   Min.   :   10   b      :2
##  Class  :character   1st Qu.: 2.5   1st Qu.:   45   j      :2
##  Mode   :character   Median : 5.0   Median :   70   c      :1
##                      Mean   : 5.0   Mean   : 1906   d      :1
##                      3rd Qu.: 7.5   3rd Qu.:   95   e      :1
##                      Max.   :10.0   Max.   :20000   f      :1
##                                                     (Other):3
```

The `summary()` function gives us a variable overview: Frequency tables for factors, five number summary for numeric variables (plus mean) and some technical information for character variables. In addition, the number of missings is printed separately for each variable (if there are any).

Filtering/Subsetting

Using logical expressions, it is possible to filter (comparable to, e.g., the SQL `select` statement in databases). The `subset()` function allows us to filter rows (selection) or columns (projection).

```
subset(df, numbers.decr > 5)
```

```
##   LET numbers.decr numbers.incr let
## 1   A           10           10   b
## 2   B            9          440   b
## 3   C            8           30   c
## 4   D            7           40   d
## 5   E            6           50   e
```

```
subset(df, let %in% letters[1:5] & numbers.incr != 10)
```

```
##   LET numbers.decr numbers.incr let
## 2   B            9          440   b
## 3   C            8           30   c
## 4   D            7           40   d
## 5   E            6           50   e
```

```
subset(df, let %in% letters[1:5], numbers.decr)
```

```
##   numbers.decr
## 1           10
## 2            9
## 3            8
## 4            7
## 5            6
```

```
subset(df, , c(LET, let))
```

```
##    LET let
## 1    A   b
## 2    B   b
## 3    C   c
## 4    D   d
## 5    E   e
## 6    F   f
## 7    G   g
## 8    H   h
## 9    I   i
## 10   J   j
## 11   J   j
```

Note that the logical "And" is expressed by a single `&`.

To remove observations with missing values (after checking that only few observations will get lost doing so!) we may use the `complete.cases()` function:

```
smoker <- c(TRUE, FALSE, TRUE, FALSE)
name <- c("Tim", "Susi", "Horst", "Walter")
sex <- factor(c("M", "F", "M", "F"), levels = c("F", "M"),
              labels = c("Female", "Male"))
figure <- c("chubby", "lean", "normal", "skinny")
age <- c(20, 30, 40, 50)
height <- c(1.65, 1.75, 1.85, 1.95)
piercings <- c(1, NA, 0, 2)

friends <- data.frame(name, age, smoker, sex, figure, height, piercings,
                      stringsAsFactors = FALSE)
head(friends)
```

```
##     name age smoker    sex figure height piercings
## 1    Tim  20   TRUE   Male chubby   1.65         1
## 2   Susi  30  FALSE Female   lean   1.75        NA
## 3  Horst  40   TRUE   Male normal   1.85         0
## 4 Walter  50  FALSE Female skinny   1.95         2
```

```
ok = complete.cases(friends)
sum(ok) # number of complete observations
```

```
## [1] 3
```

```
sum(!ok) # number of incomplete observations
```

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

```
friends_complete = subset(friends, ok) # extract all complete observations
friends_complete
```

```
##     name age smoker    sex figure height piercings
## 1    Tim  20   TRUE   Male chubby   1.65         1
## 3  Horst  40   TRUE   Male normal   1.85         0
## 4 Walter  50  FALSE Female skinny   1.95         2
```

```r
subset(friends, !ok)
```

```
##   name age smoker    sex figure height piercings
## 2 Susi  30  FALSE Female   lean   1.75        NA
```

Aggregation

Similar to SQL, it is possible to group rows and to summarize metric variables using aggregation functions:

```r
## Most piercings by sex:
aggregate(piercings ~ sex, data = friends, FUN = max)
```

```
##      sex piercings
## 1 Female         2
## 2   Male         1
```

```r
## average height by figure:
aggregate(height ~ figure, friends, mean)
```

```
##   figure height
## 1 chubby   1.65
## 2   lean   1.75
## 3 normal   1.85
## 4 skinny   1.95
```

```r
## average height by figure and sex:
aggregate(height ~ figure + sex, friends, mean)
```

```
##   figure    sex height
## 1   lean Female   1.75
## 2 skinny Female   1.95
## 3 chubby   Male   1.65
## 4 normal   Male   1.85
```

```r
## average of height and age by sex:
aggregate(cbind(height, age) ~ sex, friends, mean)
```

```
##      sex height age
## 1 Female   1.85  40
## 2   Male   1.75  30
```

The first argument expects a so-called *model formula* - the tilde ("~") stands for "evaluate by". The variables that should be aggregated are placed left of the tilde (function `cbind()`, usually used to combine two variables into a table, here means: "jointly evaluate"). On the right-hand side we place the grouping variables (The plus-sign (`+`) doesn't stand for an addition, but simply means "and").

Lists

Lists are the most versatile type of data structure. Their elements may be different data types of different classes and lengths, and they are recursive, i.e., an element of the list may be a list itself ("sublist"). Usually, we create lists using the `list()` function.

```r
vec1 <- 1:10
list1 <- list(df1, df2, vec1, X) # list of letters, vectors, matrix
list1[1] # select first list component (letters), keeping the name -> result is
```

```
## [[1]]
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
# a list ("sublist")
list1[[1]] # select first list component, dropping the name -> result is a
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
# character vector
is.list(list1[1]) # sublist is a list itself
```

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

```
is.list(list1[[1]]) # not a list
```

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

```
list1[1:3] # select the first 3 elements of the list
```

```
## [[1]]
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
##
## [[2]]
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## [[3]]
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
list1[[1:3]] # only one list element can be selected if the name is dropped!
```

```
## Error in list1[[1:3]]: recursive indexing failed at level 2
```

```
list1[[1]][5] # pick out 5th letter
```

```
## [1] "e"
```

```
list1[[4]] # select the matrix
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
list1[[4]][4, 2] # pick out element (4, 2)
```

```
## [1] 9
```

```
names(list1) <- c("let", "seq", "vec", "mat") # name/label list
list1
```

```
## $let
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
##
## $seq
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $vec
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $mat
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
list1$mat # instead of list[[4]]
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8   13
## [4,]    4    9   14
## [5,]    5   10   15
```

```
list1$mat[, 2]
```

```
## [1]  6  7  8  9 10
```

As said above, data frames are special types of lists. Hence, we can coerce a data frame into a list:

```
friends.l <- as.list(friends)
friends.l
```

```
## $name
## [1] "Tim"    "Susi"   "Horst"  "Walter"
##
## $age
## [1] 20 30 40 50
##
## $smoker
## [1]  TRUE FALSE  TRUE FALSE
##
## $sex
## [1] Male   Female Male   Female
## Levels: Female Male
##
## $figure
## [1] "chubby" "lean"   "normal" "skinny"
##
## $height
## [1] 1.65 1.75 1.85 1.95
##
## $piercings
## [1]  1 NA  0  2
```

```
is.data.frame(friends.l)
```

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

```
attributes(friends)
```

```
## $names
## [1] "name"      "age"       "smoker"    "sex"       "figure"    "height"
## [7] "piercings"
##
## $class
## [1] "data.frame"
```

```
## 
## $row.names
## [1] 1 2 3 4
```

```
attributes(friends.l)
```

```
## $names
## [1] "name"      "age"       "smoker"    "sex"       "figure"    "height"   
## [7] "piercings"
```

We may flatten lists (= produce a vector) using `unlist()`:

```
(friends.v <- unlist(friends.l))
```

```
##      name1      name2      name3      name4       age1       age2       age3 
##      "Tim"     "Susi"    "Horst"   "Walter"       "20"       "30"       "40" 
##       age4    smoker1    smoker2    smoker3    smoker4       sex1       sex2 
##       "50"     "TRUE"    "FALSE"     "TRUE"    "FALSE"        "2"        "1" 
##       sex3       sex4    figure1    figure2    figure3    figure4    height1 
##        "2"        "1"   "chubby"     "lean"   "normal"   "skinny"     "1.65" 
##    height2    height3    height4 piercings1 piercings2 piercings3 piercings4 
##     "1.75"     "1.85"     "1.95"        "1"         NA        "0"        "2"
```

```
unlist(friends) # the same, i.e., also applicable to data frames
```

```
##      name1      name2      name3      name4       age1       age2       age3 
##      "Tim"     "Susi"    "Horst"   "Walter"       "20"       "30"       "40" 
##       age4    smoker1    smoker2    smoker3    smoker4       sex1       sex2 
##       "50"     "TRUE"    "FALSE"     "TRUE"    "FALSE"        "2"        "1" 
##       sex3       sex4    figure1    figure2    figure3    figure4    height1 
##        "2"        "1"   "chubby"     "lean"   "normal"   "skinny"     "1.65" 
##    height2    height3    height4 piercings1 piercings2 piercings3 piercings4 
##     "1.75"     "1.85"     "1.95"        "1"         NA        "0"        "2"
```

```
str(friends.v) # character vector
```

```
##  Named chr [1:28] "Tim" "Susi" "Horst" "Walter" "20" "30" "40" "50" "TRUE" ...
##  - attr(*, "names")= chr [1:28] "name1" "name2" "name3" "name4" ...
```

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
list(1:10)
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
is.recursive(1:10)
```

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

```
is.recursive(list(1:10))
```

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

```
is.atomic(1:10)
```

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

```
is.atomic(list(1:10))
```

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

Results of statistical tests or models are frequently saved in lists.