R Programming as a Part of Bigdata Course
Introduction to R language Elements

John Aa. Sørensen, lektor
Section for Information Technology, DTU Diplom

Center for Continuing Education, DTU

9 February 2017

R Programming Examples TOC (1)

```
# Filename: 2_R_Programming.R
# Objective:
# Introduction to R language elements and functions.
#
#  TABLE OF CONTENTS
#
# 2.0: Enter the main R Language documentation.
# 2.1: Scalar operations and examples on functions.
# 2.2: Vector definitions and operations.
# 2.3: Matrix definitions and vector, matrix operations.
# 2.4: Factors. (Ordinal variables, where only the ordering matters).
# 2.5: Lists - the most general ordered collection of objects, which
#         can be of mixed types.
```

R Programming Examples TOC (2)

R Programming Examples TOC (3)

R Programming Examples TOC (4)

Main R Manual and I/O Parameters

```
> ##################################################################
> # 2.0: Enter the main R Language documentation.
> #
> help.start()   # This is main entry to the R documentation.
If nothing happens, you should open
http://127.0.0.1:13955/doc/html/index.html yourself
> #
> .opar <- par(no.readonly=TRUE)  # Store original I/O parameters
>                                 # for later restoring.
```

Packages, Install and Include in Library

```
> #
> # Install packages needed in this script.
> #
> install.packages("combinat",lib="C:/R_packages")
trying URL
 'https://cran.rstudio.com/bin/windows/contrib/3.2/combinat_0.0-8.zip'
Content type 'application/zip' length 29294 bytes (28 KB)
downloaded 28 KB

package combinat successfully unpacked and MD5 sums checked

The downloaded binary packages are in
C:\Users\jaas\AppData\Local\Temp\Rtmp4gZ46V\downloaded_packages
> library("combinat",lib="C:/R_packages")
Det folgende objekt er maskeret fra package:utils:
    combn
>
```

Scalar operations, Functions, logicals (1)

```
> ##################################################################
> # 2.1: Examples on scalar operations, functions and logicals
> #
> 10*32          # Multiply.
[1] 320
> (10+2)*5       # Do not use [10+2]*5. It don't work.
[1] 60
> 2^3            # Exponent
[1] 8
> 2**4           # Exponent
[1] 16
> (2+3i)^2       # Square a complex number: 4 + 12i - 9 = -5+12i
[1] -5+12i
> 12/3           # Division
[1] 4
>
```

Scalar operations, Functions, logicals (2)

```
> #
> # Some constants: pi and e.
> #
> print(pi, digits=15)
[1] 3.14159265358979
> print(exp(1), digits=10)
[1] 2.718281828
>
```

Scalar operations, Functions, logicals (3)

```
> #
> # Some functions, e.g. [Kabacoff, 2015] page 91, Table 5.2
> #
> abs(-4)
[1] 4
> sqrt(2.71^2)
[1] 2.71
> ceiling(15.29)
[1] 16
> floor(15.29)
[1] 15
> trunc(-1.5)                     # Truncation towards 0
[1] -1
> round(3.14159265, digits=3)     # Round down to 3 decimal places.
[1] 3.142
> signif(3.141592, digits=2)      # Signicant digits.
[1] 3.1
>
```

Scalar operations, Functions, logicals (4)

```
> ac <- c(1,2,5,100)
> median_ac <- median(ac); median_ac
[1] 3.5
> mean_ac <- mean(ac);mean_ac
[1] 27
> #
> exp(1)        # Exponential function of 1, Eulers number e=2.71828
[1] 2.718282
> log(exp(1))   # Natural logarithm to Eulers number e, which is 1
[1] 1
> log10(10)     # Base 10 logarithm to 10 which is 1.
[1] 1
> log10(1)      # Base 10 logarithm to 1, which is 0.
[1] 0
> log(3^2, base=3) # Base 3 logarithm of 9, which is 2, bc 3^2=9.
[1] 2
```

Scalar operations, Functions, logicals (5)

```
> #
> # Constants
> #
> ?NA  # NA: Not available/missing value. NA is a reserved word in R.
> is.na(c(1,2,NA))  # Ask if there is an NA in the array.
[1] FALSE FALSE  TRUE
> #
> # INF, -INF Numbers out of numerical range.
> ?Inf       # Inf is a reserved word in the R.
> 2^1024     # This return an Inf.
[1] Inf
> -2^1024    # This returns an -Inf.
[1] -Inf
> #
> # NaN  Not a number.
> ?NaN
> 0/0        # Notice this is "Not a number"
[1] NaN
> 1/0        # Notice this is Inf
[1] Inf
>
```

Scalar operations, Functions, logicals (6)

```
> #
> # Logicals, relations and set operations.
> #
> z <- 1:5;z                   # z is a vector.
[1] 1 2 3 4 5
> test_z1 <- (z < 3); test_z1   # Test "less than" on each position.
[1]  TRUE  TRUE FALSE FALSE FALSE
> test_z2 <- (z == 2); test_z2  # Test "equal to" on each position.
[1] FALSE  TRUE FALSE FALSE FALSE
> test_z3 <- (z != 2); test_z3  # Test "not equal to" on each position.
[1]  TRUE FALSE  TRUE  TRUE  TRUE
> test_z4 <- (z > 1 & z < 3); test_z4 # Test "logical AND".
[1] FALSE  TRUE FALSE FALSE FALSE
> test_z5 <- (z > 4 | z < 1); test_z5 # Test "logical OR".
[1] FALSE FALSE FALSE FALSE  TRUE
# test membership on each position
> test_z6 <- (z %in% c(0,-1,1,5,6,7,8,9,10,100)); test_z6
[1]  TRUE FALSE FALSE FALSE  TRUE
>
```

Scalar operations, Functions, logicals (7)

```
> #
> # Set operations: union, intersect, setdif
> #
> z1 <- 1:3;z1
[1] 1 2 3
> z2 <- 2:5;z2
[1] 2 3 4 5
> z3 <- union(z1, z2); z3
[1] 1 2 3 4 5
> z4 <- intersect(z1, z2); z4
[1] 2 3
> z5 <- setdiff(z1, z2); z5  # Remove from z1 the elements in z2
[1] 1
> z6 <- setdiff(z2, z1); z6  # Remove from z2 the elements in z1
[1] 4 5
>
```

Vector Definitions and Operation (1)

```
> ##################################################################
> # 2.2: Examples on vector definitions and vector operations.
> #
> #   Examples on atomic vectors, c.f. [Kabacoff, 2015] page 464.
> #   Atomic vectors are arrays of a single type:
> #     Logical, real, complex, character
> passed <- c(TRUE, TRUE, FALSE, TRUE) # Vector of logicals.
> passed
[1]  TRUE  TRUE FALSE  TRUE
> ages <- c(15, 18, 25, 14, 19); ages      # Vector of numericals.
[1] 15 18 25 14 19
> cmplxNums <- c(1+2i, 0+1i, 39+3i, 12+2i) # Vector of numericals.
> cmplxNums
[1]  1+2i  0+1i 39+3i 12+2i
> cmplxNums1 <- c(1, 1i, 39+3i, 12+2i) # "It's ok to skip ..."
> cmplxNums1
[1]  1+0i  0+1i 39+3i 12+2i
>
```

Vector Definitions and Operation (2)

```
> #      Examples on vector operations, based on
> #      examples from R Manual display on console using  >?c
> #      and [Kabacoff, 2015] page 22-23.
> #
> a <- c(1:1,7:9); a                 # Create array with 4 elements.
[1] 1 7 8 9
> (a1 <- seq(from=3, to= 12, by=2))  #
[1]  3  5  7  9 11
> b <- c(1:5, 10.5, "next"); b       # Create array with 7 elements.
[1] "1"    "2"    "3"    "4"    "5"    "10.5" "next"
> c <- letters[3:5];c                # letters c,d,e
[1] "c" "d" "e"
> #
> # Used with a single argument to drop attributes
> x <- 1:4; x                    # Create and display x
[1] 1 2 3 4
> y <- c(1:4); y                 # Same as the immediate former line.
[1] 1 2 3 4
```

Vector Definitions and Operation (3)

```
> # Alternative methods for creating vectors
> a1 <- rep(1:4,2); a1            #
[1] 1 2 3 4 1 2 3 4
> a2 <- rep(1:4, each=2); a2      #
[1] 1 1 2 2 3 3 4 4
> a3 <- rep(1:4, c(2,2,2,2)); a3   # Same as the former result.
[1] 1 1 2 2 3 3 4 4
> a4 <- c(1,2,3,4, 'bigdata'); a4  # Vector with different types.
[1] "1"       "2"       "3"       "4"       "bigdata"
>                                 # All entries becomes type: string.
> a5 <- length(a4); a5            # Length of vector
[1] 5
> #
> x <- c(1,2,3,4,5,6,7,8)    # Create a vector
> x1 <- x[x > 4 & x < 7]; x1  # All elements with  4 < index < 7
[1] 5 6
>
```

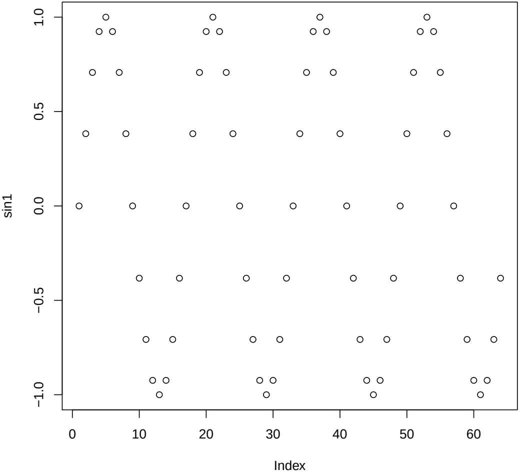
Vector Definitions and Operation (4)

```
> # Assign names to the components of x using the  names() function.
> # Check the function of names() using R Manual display on console
> #  using  >?names
> ?names()                          # Check manual
> names(x) <- letters[1:4]; x # Assigns the letters a b c d to the
   a    b    c    d <NA> <NA> <NA> <NA>
   1    2    3    4    5    6    7    8
>                                   # 4 components of x.
> c(x)                              # The array has names.
   a    b    c    d <NA> <NA> <NA> <NA>
   1    2    3    4    5    6    7    8
> as.vector(x)                      # The vector has no names.
[1] 1 2 3 4 5 6 7 8
>
```

Vector Definitions and Operation (5)

```
> # Example on a vector argument to a function
> arg1 <- c(0:63)*pi/8;        # Generate a vector with pi/8 multiples.
> head(arg1)             # List top of vector arg1
[1] 0.0000000 0.3926991 0.7853982 1.1780972 1.5707963 1.9634954
> sin1 <- sin(arg1);
> head(sin1)             # List top of vector sin1
[1] 0.0000000 0.3826834 0.7071068 0.9238795 1.0000000 0.9238795
> plot(sin1)
> #
> pdf("fig_2_sin1.pdf")  # Generate pdf of plot
> par(.opar)
> plot(sin1)
> dev.off()
RStudioGD
        2
> par(.opar)
>
```

sin()



Matrix Definitions and Vector, Matrix Operations (1)

```
> ###################################################################
> # 2.3: Examples on matrix definitions and vector, matrix operations.
> #
> # Define a matrix, c.f. [Kabacoff,2015] page 23; display the matrix
> #   and check its class and attributes.
> #   Notice that the matrix is generated columnwise from a vector 1:N.
> #   If generated by rows instead of by columns use the following
> #   attribute in the matrix argument: matrix( .... byrow=T)
>
```

Matrix Definitions and Vector, Matrix Operations (2)

```
> #   Now continue from here using by column  organisation.
> #   Notice there is no bycolumn (byrow=FALSE) attribute.
> #
> Nelem <- 32; Nrow <- 4; Ncol <- 8
>                     # Notice that Nelem = Nrow*Ncol (required).
> # Create a matrix.
> z <- matrix(1:Nelem, nrow=Nrow, ncol=Ncol, byrow=FALSE); z
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    5    9   13   17   21   25   29
[2,]    2    6   10   14   18   22   26   30
[3,]    3    7   11   15   19   23   27   31
[4,]    4    8   12   16   20   24   28   32
> class(z)
[1] "matrix"
> attributes(z)     # Display the attributes on the console.
$dim
[1] 4 8

> dim(z)            # Display the dimension on the console.
[1] 4 8
> #
```

Matrix Definitions and Vector, Matrix Operations (3)

```
> # Check matrix creation if too few elements.
> # Notice no warning but circular allocation.
> # Create an 4 x 8 matrix.
> z1 <- matrix(1:31, nrow=4, ncol=8, byrow=FALSE); z1
Warning message:
In matrix(1:31, nrow = 4, ncol = 8, byrow = FALSE) :
datalength [31] is not sub-multiple or multiple of number of rows [4]
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    5    9   13   17   21   25   29
[2,]    2    6   10   14   18   22   26   30
[3,]    3    7   11   15   19   23   27   31
[4,]    4    8   12   16   20   24   28    1
> #
```

Matrix Definitions and Vector, Matrix Operations (4)

```
> # Get single element in matrix z ("the lower right corner element")
> z[Nrow,Ncol]
[1] 32
> # Get column number 2 in matrix z.
> z[,2]
[1] 5 6 7 8
> # Get row number 3 in matrix z.
> z[3,]
[1]  3  7 11 15 19 23 27 31
> # Get last row
> z[Nrow,]
[1]  4  8 12 16 20 24 28 32
> # Get last column
> z[,Ncol]
[1] 29 30 31 32
```

Matrix Definitions and Vector, Matrix Operations (5)

```
> # Get all rows, except rows 2 and 3
> z[-(2:3),]
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    5    9   13   17   21   25   29
[2,]    4    8   12   16   20   24   28   32
> # Get all columns, except columns no. 1, 2, 3
> z[,-(1:3)]
     [,1] [,2] [,3] [,4] [,5]
[1,]   13   17   21   25   29
[2,]   14   18   22   26   30
[3,]   15   19   23   27   31
[4,]   16   20   24   28   32
> #
```

Matrix Definitions and Vector, Matrix Operations (6)

```
> #
> # Transpose matrix z ("Exchange rows and columns") and display
> z_t <- t(z); z_t
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
[5,]   17   18   19   20
[6,]   21   22   23   24
[7,]   25   26   27   28
[8,]   29   30   31   32
> #
```

Matrix Definitions and Vector, Matrix Operations (7)

```
> # The norm (length) of a vector:
> z2 <- matrix(c(3,4), nrow=2, ncol=1); z2
     [,1]
[1,]    3
[2,]    4
> length_z2 <- sqrt(t(z2)%*%z2); length_z2
     [,1]
[1,]    5
> #
> # Multiply a matrix with a scalar, a vector and a matrix
> z2=0.5*z; z2
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]  0.5  2.5  4.5  6.5  8.5 10.5 12.5 14.5
[2,]  1.0  3.0  5.0  7.0  9.0 11.0 13.0 15.0
[3,]  1.5  3.5  5.5  7.5  9.5 11.5 13.5 15.5
[4,]  2.0  4.0  6.0  8.0 10.0 12.0 14.0 16.0
>
```

Matrix Definitions and Vector, Matrix Operations (8)

```
> # Create a vector of apropriate size and multiply with matrix z
> v <- matrix(c(0,0,0,1,0,0,0,0), nrow=8, ncol=1)
> v
     [,1]
[1,]    0
[2,]    0
[3,]    0
[4,]    1
[5,]    0
[6,]    0
[7,]    0
[8,]    0
> dim(v)
[1] 8 1
```

Matrix Definitions and Vector, Matrix Operations (9)

```
> # Multiply matrix z and vector v and display the product.
> zv <- z%*%v; zv    # Multiply and display.
     [,1]
[1,]   13
[2,]   14
[3,]   15
[4,]   16
> z             # Display matrix and verify column and zv.
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    5    9   13   17   21   25   29
[2,]    2    6   10   14   18   22   26   30
[3,]    3    7   11   15   19   23   27   31
[4,]    4    8   12   16   20   24   28   32
>
```

Matrix Definitions and Vector, Matrix Operations (10)

```
> # Scalar product (inner product, "dot" product) between
> #  two vectors w1 and w2.
> w1 <- matrix(c(1,2,3), nrow=3, ncol=1); w1
     [,1]
[1,]    1
[2,]    2
[3,]    3
> w2 <- matrix(c(3,2,1), nrow=3, ncol=1); w2
     [,1]
[1,]    3
[2,]    2
[3,]    1
> a1 <- sum(w1*w2); a1  # Elementwise product and sum.
[1] 10
> a2 <- t(w2)%*%w1; a2  # Matrix product of a row (use transpose) and
     [,1]
[1,]   10
>                       # a column vector.
```

Matrix Definitions and Vector, Matrix Operations (11)

```
> # Outer product of two vectors w1 and w2 into matrix A1 and display
> A1 <- w1 %*% t(w2); A1
     [,1] [,2] [,3]
[1,]    3    2    1
[2,]    6    4    2
[3,]    9    6    3
> #
```

Matrix Definitions and Vector, Matrix Operations (12)

```
> # Element wise product of two matrices
> A2 <- matrix(c(1,2,3,4,5,6), nrow=2, ncol=3, byrow=TRUE);A2
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> A3 <- matrix(c(6,5,4,3,2,1), nrow=2, ncol=3, byrow=TRUE);A3
     [,1] [,2] [,3]
[1,]    6    5    4
[2,]    3    2    1
> A2A3 <- A2*A3;A2A3
     [,1] [,2] [,3]
[1,]    6   10   12
[2,]   12   10    6
>
```

Matrix Definitions and Vector, Matrix Operations (13)

```
> # Arrays: matrices with more than 2 dimensions.
> AR1 <- array(data=1:24,dim=c(3,4,2)) # 3 dim with dimensions 3x4x2
> AR1  # Display AR1
, , 1

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

, , 2

     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
[3,]   15   18   21   24
```

Matrix Definitions and Vector, Matrix Operations (14)

```
> AR2 <- array(1:24,c(3,3,2))  # identical array
> AR2  # Display AR2
, , 1

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

, , 2

     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18
```

Matrix Definitions and Vector, Matrix Operations (15)

```
> # Examples on special vectors and matrices
> # Generate and display a column zero vector of size N=3.
> N <- 3
> zero_c <- matrix(rep(0,N), nrow=N, ncol=1); zero_c
     [,1]
[1,]    0
[2,]    0
[3,]    0
> #
> # Generate and display a row vector with 1 of size N=3.
> N <- 3
> one_r <- matrix(rep(1,N), nrow=1, ncol=N); one_r
     [,1] [,2] [,3]
[1,]    1    1    1
>
```

Matrix Definitions and Vector, Matrix Operations (16)

```
> # Generate and display a matrix with zero entries
> M0 <- matrix(0, nrow=3, ncol=2); M0
     [,1] [,2]
[1,]    0    0
[2,]    0    0
[3,]    0    0
> #
> # Generate and display a diagonal matrix
> ?diag()              # Look into the manual of diag()
> Md <- diag(c(1,2,3,4,5,6)); Md
     [,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    4    0    0
[5,]    0    0    0    0    5    0
[6,]    0    0    0    0    0    6
> #
```

Matrix Definitions and Vector, Matrix Operations (17)

```
> #
> # Generate and display an N x N unit matrix
> Nd=3; Munit=diag(1,Nd); Munit
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> #
```

Matrix Definitions and Vector, Matrix Operations (18)

```
> # Define and exemplify multiplication of matrices MM1 %*% MM2
> # Notice that the number of columns in the leftmost matrix MM1
> #   must be equal to the number of rows of rightmost matrix MM2.
> n1row <- 2;     n1col <- 3; # Rows and columns of leftmost matrix.
> n2row <- n1col; n2col <- 4; # Rows and columsn of rightmost matrix.
> MM1 <- matrix(1:(n1row*n1col),nrow=n1row, ncol=n1col); MM1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> MM2 <- matrix(1:(n2row*n2col),nrow=n2row, ncol=n2col); MM2
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> MM12 <- MM1 %*% MM2; MM12  # F.inst chec manually elements of MM12
     [,1] [,2] [,3] [,4]
[1,]   22   49   76  103
[2,]   28   64  100  136
>
```

Matrix Definitions and Vector, Matrix Operations (19)

```
> # Determine the inverse of a quadratic matrix.
> # Create an example quadratic matrix A1
> A1 <- matrix(c(1,2,3,4), nrow=2, ncol=2, byrow=FALSE); A1
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> #
> A1_inv <- solve(A1); A1_inv   # Determine and display inverse A1.
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
> #
> # Check that A1 and A1_inv are inverse by multiplying.
> A1A1_inv <- A1 %*% A1_inv; A1A1_inv
     [,1] [,2]
[1,]    1    0
[2,]    0    1
> # Check also the alternative ordering
> A1_invA1 <- A1_inv %*% A1; A1_invA1
     [,1] [,2]
[1,]    1    0
[2,]    0    1
```

Matrix Definitions and Vector, Matrix Operations (20)

```
> # Solve a quadratic equation system A1 %*% x1 = a1
> a1=matrix(c(5,6), nrow=2, ncol=1, byrow=FALSE); a1
     [,1]
[1,]    5
[2,]    6
> x1 <- A1_inv %*% a1; x1
     [,1]
[1,]   -1
[2,]    2
> #
```

Factors (1)

```
> ##################################################################
> # 2.4: Factors
> #   (factors: Ordinal variables, where only the ordering matters.
> #             The numerical values and differences does not matter.)
> #     Ref. [Kabacoff, 2015] page 28.
> #
> ?factor()         # Check manual on factor.
> #
>
```

Factors (2)

```
> # Example:
> # Likert scale for representing responses on statements on product.
> # Scale value 5: The customer strongly agree (SA).
> # Scale value 4: The customer agree (A).
> # Scale value 3: The customer is neutral (N).
> # Scale value 2: The customer disagree (D).
> # Scale value 1: The customer strongly disagree (SD).
> # Scale value NA: The customer abstains (AB) from assessing.
> #
> # factor levels: Strongly Agree (SA), Agree (A),
> #                Neutral (N),
> #                Disagree(D), Strongly Disagree (SD).
> #
```

Factors (3)

```
> survey.vector <- c('A','SA','N','SD','A','A','D')
> survey.vector
[1] "A"  "SA" "N"  "SD" "A"  "A"  "D"
> #
> # Assign numerical value to category by ordered list
> #
> survey.factor <- factor(survey.vector, order=TRUE,
+                        levels=c('SD','D','N','A','SA'))
> #                      values     1   2   3   4  5
> survey.factor
[1] A  SA N  SD A  A  D
Levels: SD < D < N < A < SA
# Check the numerical values of categories.
> (as.numeric(survey.factor))
[1] 4 5 3 1 4 4 2
>
```

Lists (1)

```
> ##################################################################
> # 2.5: lists - the most general ordered collection of objects,
> #              which can be of mixed types.
> #
> ?list()   # Check manual for the list() function.
> obj1 <- 1:3; obj2 <- c('text1', 'text2', 'text3');
> Nelem <- 6; Nrow <- 3; Ncol <- 2 # Notice Nelem = Nrow*Ncol (req.)
> obj3 <- matrix(1:Nelem, nrow=Nrow, ncol=Ncol, byrow=FALSE)
> # create a list with 3 objects:
> m1 <- list(name1=obj1, name2=obj2, name3=obj3); m1
$name1
[1] 1 2 3
$name2
[1] "text1" "text2" "text3"
$name3
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
>
```

Lists (2)

```
> # Accessing list objects individually through indexing
> m1[[1]]          # The same as m1$name1
[1] 1 2 3
> m1[['name1']]    # The same as m1$name1
[1] 1 2 3
> m1[[2]]          # The same as m1$name2
[1] "text1" "text2" "text3"
> m1[[2]][2]       # 2'nd list object element no. [2]
[1] "text2"
> m1[[3]]          # The same as m1$name3
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> m1[[3]][3,2]     # 3'nd list object element no. [3,2]
[1] 6
>
```

Lists (3)

```
#
> m1$name1[2]   # The same as obj1[2]
[1] 2
> m1$name2[3]   # The same as obj2[3]
[1] "text3"
> m1$name3[2,2] # The same as obj3[2,2]
[1] 5
```

Lists (4)

```
> # Further examples on defining and applying lists.
> #      Ref. [Kabacoff, 2015] page 30.
> #
> # Using the example [Kabacoff, 2015] page 31, demonstrates ex. of
> # a list with: string, numeric vector, matrix and character vector.
> g <- "first list"
> h <- c(25,26,18,39)
> j <- matrix(1:10, nrow=5)   # By-column as byrow=F (default)
> k <- c("one","two","three")
> newlist <- list(title=g,ages=h,j,k)
```

Lists (5)

```
> newlist <- list(title=g,ages=h,j,k)
> newlist
$title
[1] "first list"

$ages
[1] 25 26 18 39

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

[[4]]
[1] "one"   "two"   "three"

>
```

Lists (6)

```
> # Refer to the second object in the list, by index [2]
> newlist[[2]]
[1] 25 26 18 39
> # Refer to the second object in the list, by name ["ages"]
> newlist[["ages"]]
[1] 25 26 18 39
> # Refer to the second named component
> newlist$ages
[1] 25 26 18 39
> # Refer to the first named component.
> newlist$title
[1] "first list"
```

Dataframes (1)

```
> ################################################################
> # 2.6: Examples on definition and operations on data frames.
> #
> ?data.frame() # Initially look into the manual for data.frame().
> #
> # QUOTE FROM MANUAL: "This function creates data frames,
> # tightly coupled collections of variables which share many
> # of the properties of matrices and of lists, used as the
> # fundamental data structure by most of R's modeling software".
>
```

Dataframes (2)

```
> # In the following is an example on a data frame for the represen-
> # tation of heterogeneous data from an example on user assessments
> # of products. The example contains the following operations:
> #  - create a data frame.
> #  - Insert a new variable in a data frame.
> #  - Merge two sets of observations for the same set of variables.
> #  - Identifying missing values in a data frame.
> #  - Excluding missing values in preparation for analysis.
> #  - Grouping observations, example into age groups:
> #    Teen (13-19), Young (20-39), MidAged (40-69),
> #    MidAgeP (70-79), Old (80-).
> #  - Sorting observations according to one variable.
> #  - Plot selected parts of data frame.
>
```

Dataframes (3)

```
> # 2.6.1: Create a data frame -----------------------------------
> #
> # Create a data frame which contains a mixture of numerical and
> # character values, exemplified by customer assessment of two
> # products, using a Likert scale for the product assessment.
> #
> # Create a data frame which contains a mixture of numerical and
> # character values, exemplified by customer assessment of two
> # products, using a Likert scale for the product assessment.
> #
> # The Likert scale used for assessing a statement about the product:
> # Scale value 5: The customer strongly agree (SA).
> # Scale value 4: The customer agree (A).
> # Scale value 3: The customer is neutral (N).
> # Scale value 2: The customer disagree (D).
> # Scale value 1: The customer strongly disagree (SD).
> # Scale value NA: The customer abstains (AB) from assessing.
> #
```

Dataframes (4)

```
> # There are two products, denoted Product 1 and Product 2.
> # Each customer assessment is labeled with the customer gender.
> # Each customer register the customer age.
> # Each customer assesses the correctness of each of 3 statements
> #    about the product:
> #   S1: The product is useful.
> #   S2: The product price is acceptable.
> #   S3: The customer will, without hesitation, recommend the product
> #       to a person known well by the customer.
> #
> # In the present example there are 8 customers.
> #
> # Now build a data frame for representing the above assessments on
> #  the two products: Product 1 and Product 2.
>
```

Dataframes (5)

```
> # The products 1 or 2 assessed by the customer 1 to 8.
> Prod_no <- c(1,1,2,1,2,1,2,2 )
> # F: Female and M: Male.
> Gender <- c("F","F","M","F","M","M","F","M")
> Age <- c(37,81,57,79,17,18,67,45)  # The customers ages
> S1 <- c(4,3,2,5,NA,3,5,5)          # Assessments of Statement S1.
> S2 <- c(3,4,1,4,3,NA,1,1)          # Assessments of Statement S2.
> S3 <- c(4,3,2,5,3,3,4,5)           # Assessments of Statement S3.
> # Create dataframe.
> Assessments <- data.frame(Prod_no, Gender, Age, S1, S2, S3)
```

```
> Assessments         # Display the values in the complete data frame.
  Prod_no Gender Age S1 S2 S3
1       1      F  37  4  3  4
2       1      F  81  3  4  3
3       2      M  57  2  1  2
4       1      F  79  5  4  5
5       2      M  17 NA  3  3
6       1      M  18  3 NA  3
7       2      F  67  5  1  4
8       2      M  45  5  1  5
> Assessments$Prod_no  # Display the product numbers, assessed.
[1] 1 1 2 1 2 1 2 2
> Assessments$Gender   # Display the customer gender.
[1] F F M F M M F M
Levels: F M
> Assessments$Age      # Display the customers age.
[1] 37 81 57 79 17 18 67 45
> Assessments$S1       # Display customer replies for Statement 1.
[1]  4  3  2  5 NA  3  5  5
> Assessments$S2       # Display customer replies for Statement 2.
[1]  3  4  1  4  3 NA  1  1
> Assessments$S3       # Display customer replies for Statement 3.
[1] 4 3 2 5 3 3 4 5
>
```

Dataframes (7)

```
> # 2.6.2: Insert a new variable in a data frame ------------------
> #
> # Example: Extend the Assessments dataframe with one more question
> #          S4 by inserting a new column S4 in the dataframe.
> #
> # S4: The product design is attractive.
> #
> S4 <- c(5 ,3 ,1 ,4 ,3 ,4 ,5 ,NA )
> ?transform()          # Look into the manual
> Assessments_1 <- transform(Assessments, S4=S4)
> Assessments_1
  Prod_no Gender Age S1 S2 S3 S4
1       1      F  37  4  3  4  5
2       1      F  81  3  4  3  3
3       2      M  57  2  1  2  1
4       1      F  79  5  4  5  4
5       2      M  17 NA  3  3  3
6       1      M  18  3 NA  3  4
7       2      F  67  5  1  4  5
8       2      M  45  5  1  5 NA
```

Dataframes (8)

```
> # ... alternative solution using cbind()
> #    (Bind a new column to the dataframe).
> Assessments_2 <- cbind(Assessments, S4)
> Assessments_2
  Prod_no Gender Age S1 S2 S3 S4
1       1      F  37  4  3  4  5
2       1      F  81  3  4  3  3
3       2      M  57  2  1  2  1
4       1      F  79  5  4  5  4
5       2      M  17 NA  3  3  3
6       1      M  18  3 NA  3  4
7       2      F  67  5  1  4  5
8       2      M  45  5  1  5 NA
> #
```

Dataframes (9)

```
> # 2.6.3: ----------------------------------------------
> # Merge two sets of observations for the same variables.
> # Insert additional rows with observations in a data frame
> # and generate a new extended dataframe.
> # Notice: In this example the observations from one loca-
> #         tion are just repeated at the other location.
> #
> # Create assessments for one location denoted A.
> Assessments_location_A <- Assessments
> # Create assessments for one other location B.
> Assessments_location_B <- Assessments
> # One dataframe with all observations.
> Assessments_all <- rbind(Assessments_location_A, +
                                  Assessments_location_B
> #
```

Dataframes (10)

```
> Assessments_all
   Prod_no Gender Age S1 S2 S3
1        1      F  37  4  3  4
2        1      F  81  3  4  3
3        2      M  57  2  1  2
4        1      F  79  5  4  5
5        2      M  17 NA  3  3
6        1      M  18  3 NA  3
7        2      F  67  5  1  4
8        2      M  45  5  1  5
9        1      F  37  4  3  4
10       1      F  81  3  4  3
11       2      M  57  2  1  2
12       1      F  79  5  4  5
13       2      M  17 NA  3  3
14       1      M  18  3 NA  3
15       2      F  67  5  1  4
16       2      M  45  5  1  5
>
```

Dataframes (11)

```
> # 2.6.4: Identifying missing values in a data frame -----
> #
> # Example: Identify the observations (rows) in the data
> #          frame with missing values.
> #
> ?is.na()                              # Manual for is.na()
> Missing <- is.na(Assessments_1[,4:7]) # Check 4 columns
> Missing
        S1    S2    S3    S4
[1,] FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE
[3,] FALSE FALSE FALSE FALSE
[4,] FALSE FALSE FALSE FALSE
[5,]  TRUE FALSE FALSE FALSE
[6,] FALSE  TRUE FALSE FALSE
[7,] FALSE FALSE FALSE FALSE
[8,] FALSE FALSE FALSE  TRUE
```

Dataframes (12)

```
> # 2.6.5:
> # Remove missing values from dataframe before analysis.
> #
> ?na.omit()
> # Exclude those data frame rows (observations) with NA.
> Assessments_2 <- na.omit(Assessments_1)
> Assessments_2
  Prod_no Gender Age S1 S2 S3 S4
1       1      F  37  4  3  4  5
2       1      F  81  3  4  3  3
3       2      M  57  2  1  2  1
4       1      F  79  5  4  5  4
7       2      F  67  5  1  4  5
>
```

Dataframes (13)

```
> # 2.6.6: Grouping observations into age groups:
> #        Teen, Young, MidAged, MidAgeP, Old
> # Teen (13-19), Young (20-39), MidAged (40-69),
> #     MidAgeP (70-79), Old (80-)
> # Include the age categories in a new variable.
> #
> Assessments$AgeCat[Assessments$Age >= 13 &
>      Assessments$Age <= 19] <- "Teen"
> Assessments$AgeCat[Assessments$Age >= 20 &
>      Assessments$Age <= 39] <- "Young"
> Assessments$AgeCat[Assessments$Age >= 40 &
>      Assessments$Age <= 69] <- "MidAge"
> Assessments$AgeCat[Assessments$Age >= 70 &
>      Assessments$Age <= 79] <- "MidAgeP"
> Assessments$AgeCat[Assessments$Age >= 80]  <- "Old"
```

Dataframes (14)

```
> Assessments   # Display Assessments with age categories.
  Prod_no Gender Age S1 S2 S3  AgeCat
1       1      F  37  4  3  4   Young
2       1      F  81  3  4  3     Old
3       2      M  57  2  1  2  MidAge
4       1      F  79  5  4  5 MidAgeP
5       2      M  17 NA  3  3    Teen
6       1      M  18  3 NA  3    Teen
7       2      F  67  5  1  4  MidAge
8       2      M  45  5  1  5  MidAge
```

Dataframes (15)

```
> # 2.6.7:
> # Sorting observations in dataframe accd. to one variable
> #
> ?order()   # Display the manual for the function order().
> # Reuse the data frame with product assessments and sort
> #    according to the customer age.
> # The products assessed by the customers.
> Prod_no <- c(1,1,2,1,2,1,2,2 )
> # F: Female and M: male.
> Gender <- c("F","F","M","F","M","M","F","M")
> Age <- c(37,81,57,79,17,18,67,45) # The customers ages
> S1 <- c(4,3,2,5,NA,3,5,5 )    # Assessments of Statement S1.
> S2 <- c(3,4,1,4,3,NA,1,1 )    # Assessments of Statement S2.
> S3 <- c(4,3,2,5,3,3,4,5 )     # Assessments of Statement S3.
> # Create the data frame.
> Assessments <- data.frame(Prod_no, Gender, Age, S1, S2, S3)
```

Dataframes (16)

```
> Assessments # Display the values in the total data frame.
  Prod_no Gender Age S1 S2 S3
1       1      F  37  4  3  4
2       1      F  81  3  4  3
3       2      M  57  2  1  2
4       1      F  79  5  4  5
5       2      M  17 NA  3  3
6       1      M  18  3 NA  3
7       2      F  67  5  1  4
8       2      M  45  5  1  5
> Assessments_sorted <-
        Assessments[order(Assessments$Age),]
> Assessments_sorted
  Prod_no Gender Age S1 S2 S3
5       2      M  17 NA  3  3
6       1      M  18  3 NA  3
1       1      F  37  4  3  4
8       2      M  45  5  1  5
3       2      M  57  2  1  2
7       2      F  67  5  1  4
4       1      F  79  5  4  5
2       1      F  81  3  4  3
```

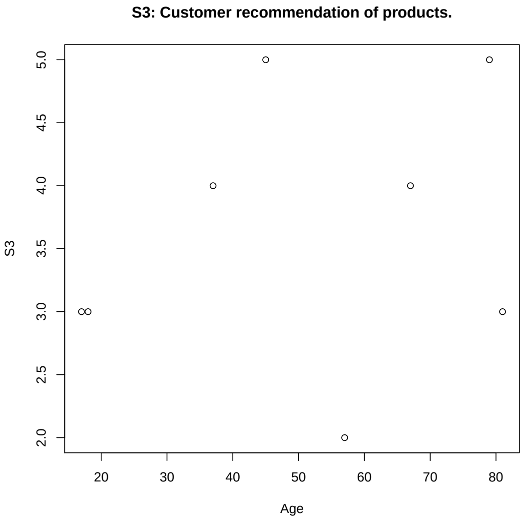
Dataframes (17)

```
> # 2.6.8: Plot selected parts of data frame exemplified by
> #        the Assessments data frame.
> #
> ?plot()              # Display manual for plot()
> ?with                # Display manual for with()
> with(Assessments, {
+   Assessments
+   plot(Age,S3);title(main = "S3: Customer recomm. ")
+   plot(Age,Prod_no); title(main = "Product number assessed at
+   given age.")
+ })
```

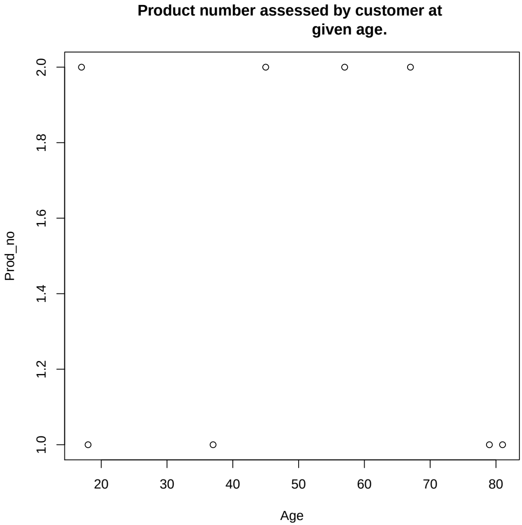
Dataframes (18)

```
> par(.opar)
> with(Assessments, {
+   Assessments
+   pdf(file = "fig_2_Assessments.pdf")
+   plot(Age,S3);title(main = "S3: Customer rec. of prod.")
+   dev.off()
+   par(.opar)
+   pdf(file = "fig_2_Assessments_2.pdf")
+   plot(Age,Prod_no); title(main = "Product number assessed
+                         by customer at given age.")
+   dev.off()
+   par(.opar)
+ })
>
```

Dataframes (19)



Dataframes (20)



Dataframes (21)

```
> # 2.6.9: Delete (remove) rows or columns in dataframes.
> #
> x1 <- c("(r1, c1)", "(r2, c1)", "(r3, c1)", "(r4, c1)")
> x2 <- c("(r1, c2)", "(r2, c2)", "(r3, c2)", "(r4, c2)")
> x3 <- c("(r1, c3)", "(r2, c3)", "(r3, c3)", "(r4, c3)")
> xx <- data.frame(x1,x2,x3) # Form data frame.
> xx
        x1       x2       x3
1 (r1, c1) (r1, c2) (r1, c3)
2 (r2, c1) (r2, c2) (r2, c3)
3 (r3, c1) (r3, c2) (r3, c3)
4 (r4, c1) (r4, c2) (r4, c3)
>
```

Dataframes (22)

```
> # Remove row 2 from xx, generate xx1 and verify.
> xx1 <- xx[-2,] # Remove row 2.
> head(xx1)      # Check that row 2 is removed.
        x1       x2       x3
1 (r1, c1) (r1, c2) (r1, c3)
3 (r3, c1) (r3, c2) (r3, c3)
4 (r4, c1) (r4, c2) (r4, c3)
> head(xx)       # ... for comparison.
        x1       x2       x3
1 (r1, c1) (r1, c2) (r1, c3)
2 (r2, c1) (r2, c2) (r2, c3)
3 (r3, c1) (r3, c2) (r3, c3)
4 (r4, c1) (r4, c2) (r4, c3)
>
```

Dataframes (23)

```
> # Remove col 2 from xx, generate xx2 and verify.
> xx2 <- xx[,-2] # Remove col. 2.
> head(xx2)      # Check that col 2 is removed.
        x1       x3
1 (r1, c1) (r1, c3)
2 (r2, c1) (r2, c3)
3 (r3, c1) (r3, c3)
4 (r4, c1) (r4, c3)
> head(xx)       # ... for comparison.
        x1       x2       x3
1 (r1, c1) (r1, c2) (r1, c3)
2 (r2, c1) (r2, c2) (r2, c3)
3 (r3, c1) (r3, c2) (r3, c3)
4 (r4, c1) (r4, c2) (r4, c3)
>
```

Dataframes (24)

```
> # Remove consequtive rows.
> xx3 <- xx[-2:-3,] # remove row 2 and 3.
> head(xx3)       # Check that row 2 and 3 are deleted.
        x1       x2       x3
1 (r1, c1) (r1, c2) (r1, c3)
4 (r4, c1) (r4, c2) (r4, c3)
> head(xx)        # ... for comparison.
        x1       x2       x3
1 (r1, c1) (r1, c2) (r1, c3)
2 (r2, c1) (r2, c2) (r2, c3)
3 (r3, c1) (r3, c2) (r3, c3)
4 (r4, c1) (r4, c2) (r4, c3)
>
```

Dataframes (25)

```
> # Remove column 2 from xx.
> xx[,2] <- NULL
> xx                   # Check that column 2 is removed.
        x1       x3
1 (r1, c1) (r1, c3)
2 (r2, c1) (r2, c3)
3 (r3, c1) (r3, c3)
4 (r4, c1) (r4, c3)
> #
```

Control Structures

```
> #######################################################
> # 2.7:  Examples on program control structures
> #
> # 2.7.1: The if-else condition
> # if (condition) expr1 else expr2
> x <- 4; y <- 20
> if (x==0) y <- 0 else y <- y/x
> y
[1] 5
>
```

Control Structures

```
> #--------------------------------------------------
> # 2.7.2: The for-loop
> for (i in 2012:2016){
+   print(paste("The year is", i))
+ }
[1] "The year is 2012"
[1] "The year is 2013"
[1] "The year is 2014"
[1] "The year is 2015"
[1] "The year is 2016"
>
```

Control Structures

```
> #--------------------------------------------------
> # 2.7.3: The while-loop
> # while (condition) expr
> while (x > 0) {print(x); x <- x-1;}
[1] 4
[1] 3
[1] 2
[1] 1
```

Control Structures

```
> #--------------------------------------------------
> # 2.7.4: The repeat function
> # repeat expr, use break to exit the loop
> x <- 0; xbreak <- 5
> repeat {print(x); x <- x+1; if (x>xbreak) break}
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
> #
```

R function

```
> #----------------------------------------------------
> # 2.7.5: Define an R function returning cubes
> #         of a sequence of integers, here
> #         cubes of 2, 3 and 4.
> #
> arg1 <- 2; arg2 <- 4;
> newfunction <- function(arg1, arg2){  # Define function.
+  cubes <- (arg1:arg2)^3
+   return(cubes)
+ }
> result <- newfunction(arg1,arg2)      # Apply function.
> result                                # Display result.
[1]  8 27 64
>
```

Probability Distributions, Sets, Combinations and Permutations (1)

```
> ##########################################################
> # 2.8: Example Probability Distributions, and
> #      Sets, Combinations and Permutations.
> #
> # 2.8.1: Bionomial Distribution, mean and variance.
> #
> ?dbinom()        # Check manual the Binomial Distribution
> #
> # Plot the binomial distribution function for n=1, p=03
> par(.opar)
> x1 <- dbinom(0:1,size=1,prob=0.3)
> barplot(x1,names.arg = c(0,1))
> title(main="Binominal distribution with n=1, p=0,3")
> par(.opar)
>
```

Probability Distributions, Sets, Combinations and Permutations (2)

```
> #
> x2 <- dbinom(0:2,size=2,prob=0.3)
> barplot(x2,names.arg = c(0:2))
> title(main="Binominal distribution with n=2, p=0,3")
> par(.opar)
> #
> x5 <- dbinom(0:5,size=5,prob=0.3)
> barplot(x5,names.arg = c(0:5))
> title(main="Binominal distribution with n=5, p=0,3")
> par(.opar)
> #
> x10 <- dbinom(0:10,size=10,prob=0.3)
> barplot(x10,names.arg = c(0:10))
> title(main="Binominal distribution with n=10, p=0,3")
> par(.opar)
>
```

Probability Distributions, Sets, Combinations and Permutations (3)

```
> x25 <- dbinom(0:25,size=25,prob=0.3)
> barplot(x25,names.arg = c(0:25))
> title(main="Binominal distribution with n=25, p=0,3")
> par(.opar)
> #
> x100 <- dbinom(0:100,size=100,prob=0.3)
> barplot(x100,names.arg = c(0:100))
> title(main="Binominal distribution with n=100, p=0,3")
> par(.opar)        # Restore default settings before continuing.
>
```

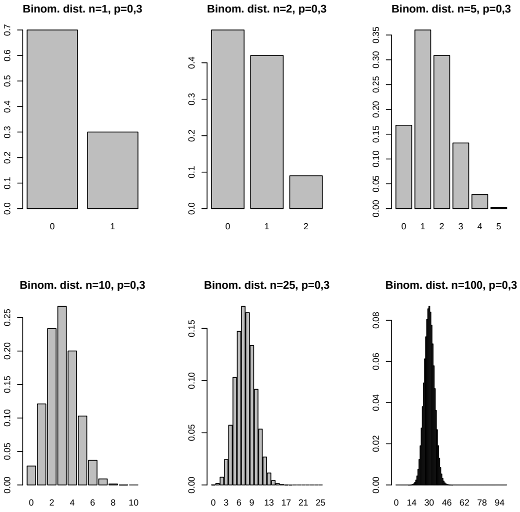
Probability Distributions, Sets, Combinations and Permutations (4)

```
> # Plot binomial distributions in a single  plot with
> # 2 rows and 3 columns.
> par(mfrow=c(2,3)) # 2 rows and 3 columns.
> barplot(x1,names.arg = c(0,1))
> title(main="Binom. dist. n=1, p=0,3")
> barplot(x2,names.arg = c(0:2))
> title(main="Binom. dist. n=2, p=0,3")
> barplot(x5,names.arg = c(0:5))
> title(main="Binom. dist. n=5, p=0,3")
> barplot(x10,names.arg = c(0:10))
> title(main="Binom. dist. n=10, p=0,3")
> barplot(x25,names.arg = c(0:25))
> title(main="Binom. dist. n=25, p=0,3")
> barplot(x100,names.arg = c(0:100))
> title(main="Binom. dist. n=100, p=0,3")
> par(.opar)
>
```

Probability Distributions, Sets, Combinations and Permutations (5)

```
> # save single plot into pdf file in the working directory.
> # Format 2 rows, 3 columns.
> pdf(file = "fig_2_barplot_binom.pdf")
> par(mfrow=c(2,3))          # 2 rows and 3 columns.
> barplot(x1,names.arg = c(0,1))
> title(main="Binom. dist. n=1, p=0,3")
> barplot(x2,names.arg = c(0:2))
> title(main="Binom. dist. n=2, p=0,3")
> barplot(x5,names.arg = c(0:5))
> title(main="Binom. dist. n=5, p=0,3")
> barplot(x10,names.arg = c(0:10))
> title(main="Binom. dist. n=10, p=0,3")
> barplot(x25,names.arg = c(0:25))
> title(main="Binom. dist. n=25, p=0,3")
> barplot(x100,names.arg = c(0:100))
> title(main="Binom. dist. n=100, p=0,3")
> par(.opar)
> dev.off()
RStudioGD
        2
>
```

Probability Distributions, Sets, Combinations and Permutations (6)



Probability Distributions, Sets, Combinations and Permutations (7)

```
> # 2.8.2 Examples of mean and median value of set of numbers.
> #
> x <- c(1:10,50);x
 [1]  1  2  3  4  5  6  7  8  9 10 50
> ?mean()                             # Check manual.
> x_mean <- mean(x); x_mean
[1] 9.545455
> ?median()                           # Check manual.
> x_median <- median(x); x_median    # Why is the xmedian= 6?
[1] 6
> #
> # Examples of variance and standard deviation.
> #
> ?var()                            # Check manual.
> x_var <- var(x); x_var
[1] 188.2727
> ?sd()                             # Check manual.
> x_sd  <- sd(x); x_sd
[1] 13.72125
>
```

Probability Distributions, Sets, Combinations and Permutations (8)

```
> # 2.8.3: Recap set operations,
> #        sampling, union, intersect, difference,
> #
> ?union()        # Check manual for set operations.
> ?intersect()
> ?setdiff()
> ?setequal()
> ?sort()         # Check manual.
> ?sample()       # Check the random sampling function.
> #
```

Probability Distributions, Sets, Combinations and Permutations (9)

```
> # sample: Select randomly 9 values from the set 1:20.
> x <- c(sort(sample(1:20, 9)), NA); x
 [1]  2  4  6  8 14 16 17 19 20 NA
> # sample: Select randomly 7 values from the set 3:23.
> y <- c(sort(sample(3:23, 7)), NA); y
[1]  3 10 11 12 14 15 16 NA
> xy_union <- union(x, y); xy_union
 [1]  2  4  6  8 14 16 17 19 20 NA  3 10 11 12 15
> xy_intersect <- intersect(x, y); xy_intersect
[1] 14 16 NA
> xy_setdiff <- setdiff(x, y); xy_setdiff
[1]  2  4  6  8 17 19 20
> yx_setdiff <- setdiff(y, x); yx_setdiff
[1]  3 10 11 12 15
> xy_setequal <- setequal(x, y); xy_setequal
[1] FALSE
>
```

Probability Distributions, Sets, Combinations and Permutations (10)

```
> # 2.8.4: Combinations and Permutations.
> #
> # Generate all combinations of two letters from a,b,c,d.
> # Notice that when generating combinations: the order does not
> # matter. This means that the combinations a,b and b,a does only
> # count for one.
> ?combn()                # Check manual
```

Probability Distributions, Sets, Combinations and Permutations (11)

```
> a1 <- combn(letters[1:4], 2); a1
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] "a"  "a"  "a"  "b"  "b"  "c"
[2,] "b"  "c"  "d"  "c"  "d"  "d"
> # ... now try two  digit combinations from the digits 0:9
> a2 <- combn(0:9, 2);a2
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] ...
[1,]    0    0    0    0    0    0    0    0    0     1 ...
[2,]    1    2    3    4    5    6    7    8    9     2 ...
     [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] ...
[1,]     1     1     2     2     2     2     2     2 ...
[2,]     8     9     3     4     5     6     7     8 ...
     [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] ...
[1,]     3     4     4     4     4     4     5     5 ...
[2,]     9     5     6     7     8     9     6     7 ...
     [,44] [,45]
[1,]     7     8
[2,]     9     9
```

Probability Distributions, Sets, Combinations and Permutations (12)

```
> a2_size <- dim(a2); a2_size
[1]  2 45
> a2_comb_count <- a2_size[2]; a2_comb_count
[1] 45
> #
> # Check the number of combinations of r out of n.
> #   comb_r_n = n!/(r!*(n-r)!) notice that 0!=1
> comb_2_10 <- factorial(10)/(factorial(2)*factorial(10-2))
> comb_2_10
[1] 45
> #
> ?factorial()              # Check n! the factorial function.
>
```

Probability Distributions, Sets, Combinations and Permutations (13)

```
> # 2.8.5: Generate all permutations of the numbers 0:3.
> # Notice fort permulations: the the order does
> # matter. This means 0,1,2,3 and 1,0,2,3 are different
> # and each contribute with one to the total count of
> # the number of permutations.
> a3 <- permn(c(0:3));a3  # Generate a list with permutations.
[[1]]
[1] 0 1 2 3

[[2]]
[1] 0 1 3 2

[[3]]
[1] 0 3 1 2

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

Probability Distributions, Sets, Combinations and Permutations (14)

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

[[6]]
[1] 0 3 2 1

[[7]]
[1] 0 2 3 1

[[8]]
[1] 0 2 1 3

[[9]]
[1] 2 0 1 3

[[10]]
[1] 2 0 3 1

[[11]]
[1] 2 3 0 1
```

Probability Distributions, Sets, Combinations and Permutations (15)

```
[[12]]
[1] 3 2 0 1

[[13]]
[1] 3 2 1 0

[[14]]
[1] 2 3 1 0

[[15]]
[1] 2 1 3 0

[[16]]
[1] 2 1 0 3

[[17]]
[1] 1 2 0 3

[[18]]
[1] 1 2 3 0
```

Probability Distributions, Sets, Combinations and Permutations (16)

```
[[19]]
[1] 1 3 2 0

[[20]]
[1] 3 1 2 0

[[21]]
[1] 3 1 0 2

[[22]]
[1] 1 3 0 2

[[23]]
[1] 1 0 3 2

[[24]]
[1] 1 0 2 3
```

Probability Distributions, Sets, Combinations and Permutations (17)

```
> a3_class <- class(a3); a3_class
[1] "list"
> a3_length <- length(a3); a3_length
[1] 24
> #
> # Check number of permutations against the formula
> # for number of permulations of a set of distinct elements
> #
> perm_4 <- factorial(4);perm_4
[1] 24
> # Notice, as expected perm_4=4!=4*3*2*1=24
>
```

Probability Distributions, Sets, Combinations and Permutations (18)

```
> # 2.8.6: Central Limit Theorem (CLT) eg. with
> #           runif() uniform density seed.
> #
> # The Central Limit Theorem:
> #  The mean of a set of independent, identically distributed
> #  (iid) random variables where mean and variance exist, will
> #  approximate a normal distribution for increasinbg set size.
> #
> ?runif # Check manual for generator for random uniform distrib.
> #
> Nr <- 5000  # Repeat all experiments Nr times.
>
> x1_1 <- replicate(Nr, {
+   mm <- runif(1)
+   sum(mm)
+ })
>
```

Probability Distributions, Sets, Combinations and Permutations (19)

```
>
> x1_2 <- replicate(Nr, {
+   mm <- runif(2)
+   sum(mm)
+ })
>
> x1_4 <- replicate(Nr, {
+   mm <- runif(4)
+   sum(mm)
+ })
>
> x1_6 <- replicate(Nr, {
+   mm <- runif(6)
+   sum(mm)
+ })
>
> x1_10 <- replicate(Nr, {
+   mm <- runif(10)
+   sum(mm)
+ })
```

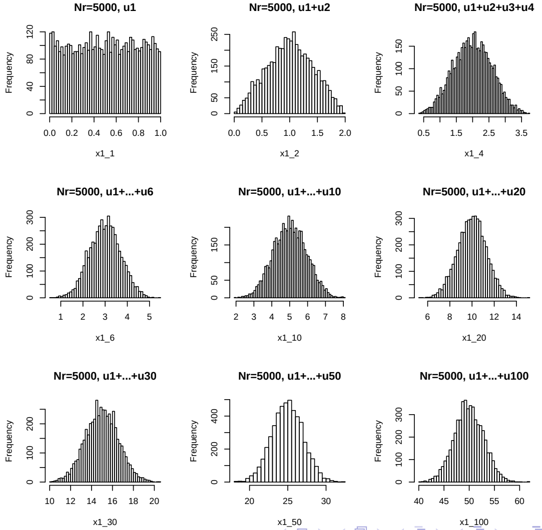
Probability Distributions, Sets, Combinations and Permutations (20)

```
>
> x1_20 <- replicate(Nr, {
+   mm <- runif(20)
+   sum(mm)
+ })
>
> x1_30 <- replicate(Nr, {
+   mm <- runif(30)
+   sum(mm)
+ })
>
> x1_50 <- replicate(Nr, {
+   mm <- runif(50)
+   sum(mm)
+ })
>
> x1_100 <- replicate(Nr, {
+   mm <- runif(100)
+   sum(mm)
+ })
>
```

Probability Distributions, Sets, Combinations and Permutations (21)

```
par(.opar)          # Restore original parameters.
par(mfrow=c(3,3))   # Figure with 3 rows and 3 columns.
hist(x1_1, breaks=50, main="Nr=5000, u1")
hist(x1_2, breaks=50, main="Nr=5000, u1+u2")
hist(x1_4, breaks=50, main="Nr=5000, u1+u2+u3+u4")
hist(x1_6, breaks=50, main="Nr=5000, u1+...+u6")
hist(x1_10, breaks=50, main="Nr=5000, u1+...+u10")
hist(x1_20, breaks=50, main="Nr=5000, u1+...+u20")
hist(x1_30, breaks=50, main="Nr=5000, u1+...+u30")
hist(x1_50, breaks=50, main="Nr=5000, u1+...+u50")
hist(x1_100, breaks=50, main="Nr=5000, u1+...+u100")
par(.opar)
dev.off()
```

Probability Distributions, Sets, Combinations and Permutations (22)



Probability Distributions, Sets, Combinations and Permutations (23)

```
> # save plot into file in the working directory.
> pdf(file = "fig_2_CLT_unif.pdf")
> par(.opar)                    # Restore original parameters.
> par(mfrow=c(3,3))             # 3 rows and 3 columns.
> hist(x1_1, breaks=50, main="Nr=5000, u1")
> hist(x1_2, breaks=50, main="Nr=5000, u1+u2")
> hist(x1_4, breaks=50, main="Nr=5000, u1+u2+u3+u4")
> hist(x1_6, breaks=50, main="Nr=5000, u1+...+u6")
> hist(x1_10, breaks=50, main="Nr=5000, u1+...+u10")
> hist(x1_20, breaks=50, main="Nr=5000, u1+...+u20")
> hist(x1_30, breaks=50, main="Nr=5000, u1+...+u30")
> hist(x1_50, breaks=50, main="Nr=5000, u1+...+u50")
> hist(x1_100, breaks=50, main="Nr=5000, u1+...+u100")
> par(.opar)
> dev.off()
null device
1 >
```

Probability Distributions, Sets, Combinations and Permutations (24)

```
> # 2.8.7: Central Limit Theorem (CLT) example with rbinom()
> #        binomial density seed.
> #
> # The Central Limit Theorem:
> #  The mean of a set of independent, identically distributed
> #  (iid) random variables where mean and variance exist, will
> #  approximate a normal distribution for increasing set size.
> #
> #
> ?rbinom()   # Check manual for generator for random binomial
> #             distributions.
> #
> Nr <- 5000   # Repeat all experiments Nr times.
> p=0.5        # Probability of sucess.
> #
> x1_1 <- replicate(Nr, {
+   mm <- rbinom(1, size=1, prob=p)
+   sum(mm)
+ })
```

Probability Distributions, Sets, Combinations and Permutations (25)

```
>
> x1_2 <- replicate(Nr, {
+   mm <- rbinom(1, size=2, prob=p)
+   sum(mm)
+ })
>
> x1_4 <- replicate(Nr, {
+   mm <- rbinom(1, size=4, prob=p)
+   sum(mm)
+ })
>
> x1_6 <- replicate(Nr, {
+   mm <- rbinom(1, size=6, prob=p)
+   sum(mm)
+ })
>
> #
> x1_10 <- replicate(Nr, {
+   mm <- rbinom(1, size=10, prob=p)
+   sum(mm)
+ })
```

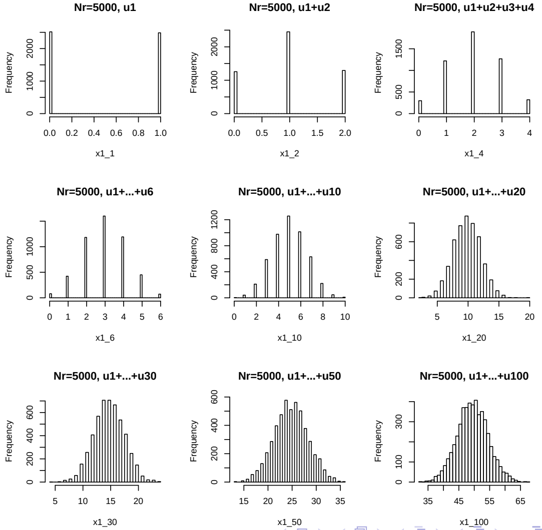
Probability Distributions, Sets, Combinations and Permutations (26)

```
> x1_20 <- replicate(Nr, {
+   mm <- rbinom(1, size=20, prob=p)
+   sum(mm)
+ })
>
> x1_30 <- replicate(Nr, {
+   mm <- rbinom(1, size=30, prob=p)
+   sum(mm)
+ })
>
> x1_50 <- replicate(Nr, {
+   mm <- rbinom(1, size=50, prob=p)
+   sum(mm)
+ })
>
> x1_100 <- replicate(Nr, {
+   mm <- rbinom(1, size=100, prob=p)
+   sum(mm)
+ })
>
```

Probability Distributions, Sets, Combinations and Permutations (27)

```
> # save plot into file in the working directory.
> pdf(file = "fig_2_CLT_binom.pdf")
> par(opar)          # Restore original parameters.
> par(mfrow=c(3,3))  # Figure with 3 rows and 3 columns.
> hist(x1_1, breaks=50, main="Nr=5000, u1")
> hist(x1_2, breaks=50, main="Nr=5000, u1+u2")
> hist(x1_4, breaks=50, main="Nr=5000, u1+u2+u3+u4")
> hist(x1_6, breaks=50, main="Nr=5000, u1+...+u6")
> hist(x1_10, breaks=50, main="Nr=5000, u1+...+u10")
> hist(x1_20, breaks=50, main="Nr=5000, u1+...+u20")
> hist(x1_30, breaks=50, main="Nr=5000, u1+...+u30")
> hist(x1_50, breaks=50, main="Nr=5000, u1+...+u50")
> hist(x1_100, breaks=50, main="Nr=5000, u1+...+u100")
> par(opar)
> dev.off()
null device          1
>
```

Probability Distributions, Sets, Combinations and Permutations (28)



Probability Distributions, Sets, Combinations and Permutations (29)

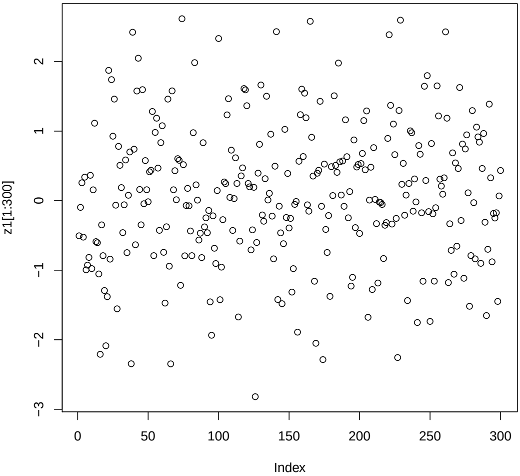
```
> # 2.8.8: The standard normal probability density function,
> #        mean=0, spread=1, variance=1.
> #
> ?rnorm()      # Check manual for normal distribution.
> Ns=100000     # Number of samples.
> mean <- 0; sd <- 1
> # generate vector of std., normal distributed samples.
> z1 <- rnorm(Ns, mean, sd)
> (class(z1))  # Check which class attribute.
[1] "numeric"
> par(.opar)
> plot(z1[1:300])      # Plot the first 300 samples.
> par(.opar)
> pdf(file = "fig_2_Std_norm_z1().pdf")
> plot(z1[1:300])      # Plot the first samples to pdf file.
> dev.off()
RStudioGD
        2
> par(.opar)
```

Probability Distributions, Sets, Combinations and Permutations (30)

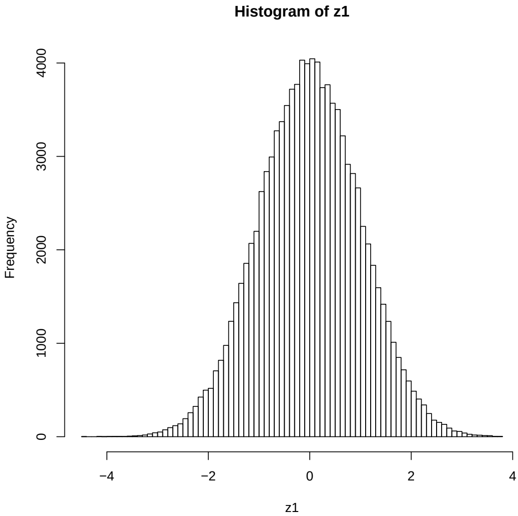
```
> #
> hist(z1, breaks=100) # Plot histogram with 50 bins=breaks.
> par(.opar)
> pdf(file = "fig_2_Std_Hist(z1).pdf")
> hist(z1, breaks=100)  # Plot histogram to pdf file.
> dev.off()
RStudioGD
        2
> par(.opar)
>
```

Probability Distributions, Sets, Combinations and Permutations (31)

+



Probability Distributions, Sets, Combinations and Permutations (32)



Probability Distributions, Sets, Combinations and Permutations (33)

```
# Arguments for density/distribution function below.
> x <- seq(-3,3,0.1)
> # density
> d1 <- dnorm(x, mean = 0, sd = 1) # Generate density function.
> str(d1)        # Check structure of d1.
 num [1:61] 0.00443 0.00595 0.00792 0.01042 0.01358 ...
> # distribution
> p1 <- pnorm(x, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
> #
> plot(x,d1, main="Std. normal density function, dnorm(), m=0, sd=1")
>dev.off()
RStudioGD
        2
> plot(x,p1, main="Std. normal dist. function, pnorm(), m=0, sd=1")
>dev.off()
RStudioGD
        2
> #
```

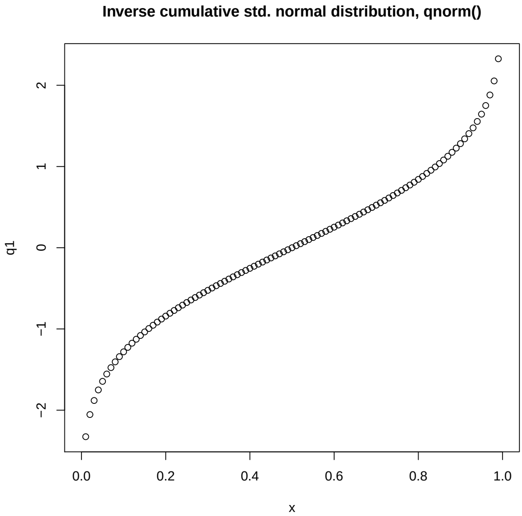
Probability Distributions, Sets, Combinations and Permutations (33)

```
> # Plot to pdf files
> pdf(file = "fig_2_5_Std_Norm_Density.pdf")
> plot(x,d1, main="Std. normal density function, dnorm(), m=0, sd=1")
> dev.off()
RStudioGD
        2
> par(.opar)
> pdf(file = "fig_2_6_Std_Norm_Distribution.pdf")
> plot(x,p1, main="Std. normal dist. function, pnorm(), m=0, sd=1")
> dev.off()
RStudioGD
        2
> par(.opar)
>
```

Probability Distributions, Sets, Combinations and Permutations (34)

```
?qnorm()   # Quantile function.
x<-seq(0,1,0.01)
q1 <- qnorm(x, mean=0, sd=1)
plot(x,q1, main="Inverse cumulative std. normal dist., qnorm()")
par(.opar)
#
# Plot to pdf files
pdf(file = "fig_2_Inv_Std_Norm_Cumulative.pdf")
plot(x,q1, main="Inverse cumulative std. normal distrib., qnorm()")
dev.off()
par(.opar)
```

Probability Distributions, Sets, Combinations and Permutations (35)



Probability Distributions, Sets, Combinations and Permutations (36)

```
> # Values for q such that  P(q>limit) for 0.5%, 1%, 2.5%, 5%
> #                                          10%, 25%, 50%.
> # NOTICE: lower.tail = FALSE => the upper tail is used (one sided).
> #
> (q_0.005 <- qnorm(0.005, mean=0, sd=1, lower.tail=FALSE))
[1] 2.575829
> (q_0.01 <- qnorm(0.01, mean=0, sd=1,lower.tail=FALSE))
[1] 2.326348
> (q_0.025 <- qnorm(0.025, mean=0, sd=1, lower.tail=FALSE))
[1] 1.959964
> (q_0.05 <- qnorm(0.05, mean=0, sd=1,lower.tail=FALSE))
[1] 1.644854
> (q_0.1 <- qnorm(0.1, mean=0, sd=1, lower.tail=FALSE))
[1] 1.281552
> (q_0.25 <- qnorm(0.25, mean=0, sd=1,lower.tail=FALSE))
[1] 0.6744898
> (q_0.50 <- qnorm(0.5, mean=0, sd=1, lower.tail=FALSE))
[1] 0
```

Probability Distributions, Sets, Combinations and Permutations (37)

```
> #
> # The -1.96, 1.96 limits for 95% in two sided std. normal
> # distribution.
> # Cut off the left 2.5% and the right 2.5% of the
> # density function, thus giving the interval [-1.96, 1.96].
> # Use the qnorm()
> a <- 0.05
> (thres_005= qnorm(1-a/2))
[1] 1.959964
>
```

References I

Joseph Adler (2012)
R in a Nutshell
OReilly

Robert I. Kabacoff (2015)
R in Action
Manning Publications 2'Ed.

R Core Team and contributors worldwide (2015)
The R Language Manual System
CRAN e.g. via RStudio

Tom Short, (2004)
Short Reference Card
CRAN cran.r-project.org/doc/contrib/Short-refcard.pdf

Paul Teetor
R Cookbook
O'Reilley

References II

Paul Torfs, Caludia Brauer
A (very) Short Introduction to R.
CRAN cran.r-project.org/doc/contrib/Torfs+Brauer-Short-R-Intro.pdf

Yanchang Zhao
R and Data Mining.
Elsevier 2013.

Yanchang Zhao
R Reference Card for Data Mining.
www.rdatamining.com
www.rdatamining.com/docs/r-reference-card-for-data-mining.pdf