



masem training
research institute

Tailor made statistical education

Statistics with R

v4

masem research institute GmbH
Bereich masem training

Unter den Eichen 5, Gebäude G,
D-65195 Wiesbaden
Registergericht Wiesbaden HRB 25700
Geschäftsführer: Dr. Guido Möser

Wissenschaftlicher Beirat:
Prof. Dr. Peter Schmidt

www.masem-training.com

Datasets and File Locations: Kompaktkurs R

Mainly datasets provided by R and additional packages. These packages will be provided together with packages.

How to access datasets: Load package using `library(packagename)` command and use `data(datasetname)` to load dataset

Larger collection of datasets in package **datasets**. To load this package just enter `library(datasets)`. For a complete list use `library(help = "datasets")`

- **iris**: Edgar Anderson's Iris Data – The famous (Fisher's and Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are *Iris setosa*, *versicolor*, and *virginica*.
- **mtcars**: Motor Trend Car Road Tests - the data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).
- **swiss**: Swiss Fertility and Socioeconomic Indicators (1888) Data: Standardized fertility measure and socio-economic indicators for each of 47 French-speaking provinces of Switzerland at about 1888.
- ...

Datasets and File Locations: Kompaktkurs R

- **cars**: Speed and Stopping Distances of Cars – The data give the speed of cars and the distances taken to stop. Note that the data were recorded in the 1920s. Can be used to generate scatterplots, calculate correlation coefficients, run a simple linear regression etc.
- **airquality**: New York Air Quality Measurements - Daily air quality measurements in New York, May to September 1973. Contains NA values.
- ...

Vorstellungsrunde

- Beschreiben Sie kurz Ihren wissenschaftlichen / beruflichen Hintergrund
- Was erwarten Sie von diesem Seminar!



Introduction and Help

masem research institute GmbH
Bereich masem training

Unter den Eichen 5, Gebäude G,
D-65195 Wiesbaden
Registergericht Wiesbaden HRB 25700
Geschäftsführer: Dr. Guido Möser

Wissenschaftlicher Beirat:
Prof. Dr. Peter Schmidt

www.masem-training.com

Two+ Main Ressources

The Comprehensive R Archive Network

Web: <https://cran.r-project.org/>

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R.

Bioconductor: Open Source Software for Bioinformatics

Web: <https://www.bioconductor.org/>

Bioconductor provides tools for the analysis and comprehension of high-throughput genomic data. Bioconductor uses the R statistical programming language, and is open source and open development.

R: The R Project for Statistical Computing

Web: <https://www.r-project.org/>

R Project Website

Contains: Download (link to CRAN); R project; R Foundation; Help with R; Documentation; Links



R: Manuals

← → ↻ <https://cran.r-project.org/manuals.html>   

The R Manuals

edited by the R Development Core Team.

The following manuals for R were created on Debian Linux and may differ from the manuals for Mac or Windows on platform-specific pages, but most parts will be identical for all platforms. The correct version of the manuals for each platform are part of the respective R installations. The manuals change with R, hence we provide versions for the most recent released R version (R-release) a very current version for the patched release version (R-patched) and finally a version for the forthcoming R version that is still in development (R-devel).

Here they can be downloaded as PDF files, EPUB files, or directly browsed as HTML:

Manual	R-release	R-patched	R-devel
An Introduction to R is based on the former "Notes on R", gives an introduction to the language and how to use R for doing statistical analysis and graphics.	HTML PDF EPUB	HTML PDF EPUB	HTML PDF EPUB
R Data Import/Export describes the import and export facilities available either in R itself or via packages which are available from CRAN.	HTML PDF EPUB	HTML PDF EPUB	HTML PDF EPUB
R Installation and Administration	HTML PDF EPUB	HTML PDF EPUB	HTML PDF EPUB
Writing R Extensions covers how to create your own packages, write R help files, and the foreign language (C, C++, Fortran, ...) interfaces.	HTML PDF EPUB	HTML PDF EPUB	HTML PDF EPUB
A draft of The R language definition documents the language <i>per se</i> . That is, the objects that it works on, and the details of the expression evaluation process, which are useful to know when programming R functions.	HTML PDF EPUB	HTML PDF EPUB	HTML PDF EPUB
R Internals : a guide to the internal structures of R and coding standards for the core team working on R itself.	HTML PDF EPUB	HTML PDF EPUB	HTML PDF EPUB
The R Reference Index : contains all help files of the R standard and recommended packages in printable form. (9MB, approx. 3500 pages)	PDF	PDF	PDF

Translations of manuals into other languages than English are available from the [contributed documentation](#) section (only a few translations are available).


The LaTeX or Texinfo sources of the latest version of these documents are contained in every R source distribution (in the subdirectory `doc/manual` of the extracted archive). Older versions of the manual can be found in the respective [archives of the R sources](#). The HTML versions of the manuals are also part of most R installations (accessible using function `help.start()`).

Please check the manuals for R-devel before reporting any issues with the released versions.

R: `help.start()` or in R Menu (`help > manuals > ...`)
Online: Official manuals: <https://cran.r-project.org/manuals.html>



Task Views on CRAN

← → ↻  <https://cran.r-project.org/web/views/>

CRAN Task Views

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
SpatioTemporal	Handling and Analyzing Spatio-Temporal Data
Survival	Survival Analysis
TimeSeries	Time Series Analysis
WebTechnologies	Web Technologies and Services
gR	gRaphical Models in R

To automatically install these views, the `ctv` package needs to be installed, e.g., via

```
install.packages("ctv")
library("ctv")
```

and then the views can be installed via `install.views` or `update.views` (which first assesses which of the packages are already installed and up-to-date), e.g.,

```
install.views("Econometrics")
or
update.views("Econometrics")
```

Task Views: <https://cran.r-project.org/web/views/>



R

R: Script Editor

```
RGui (64-bit)
Datei Bearbeiten Pakete Windows Hilfe

R Console
Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, od$
'help.start()' für eine HTML Browserschnittstelle zur Hilfe.
Tippen Sie 'q()', um R zu verlassen.

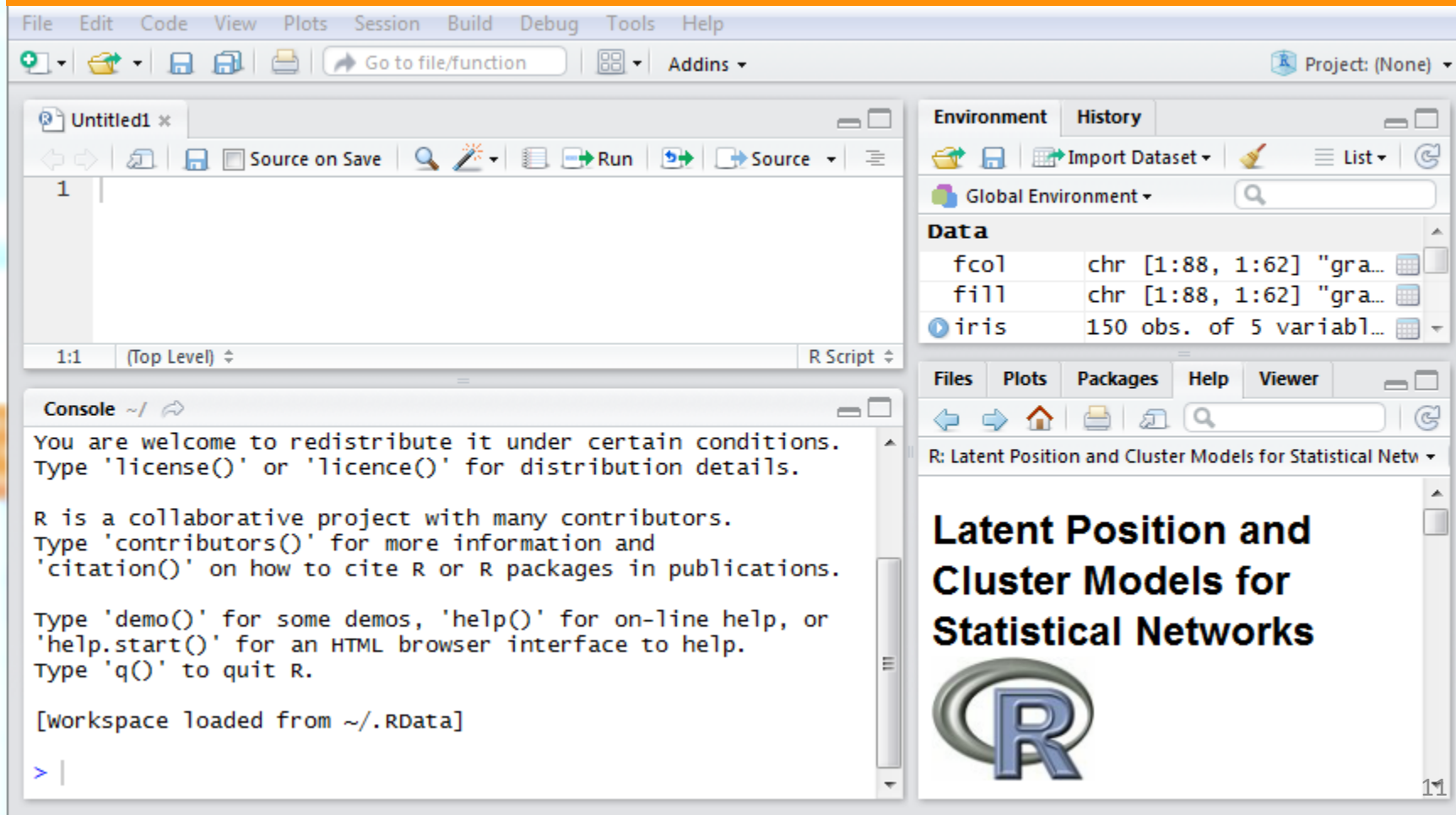
[Vorher gesicherter Workspace wiederhergestellt]

> ## for
> data(iris)
> str(iris)
'data.frame':  150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 $
> par(mfrow = c(2,2))
> ##
> for (i in 1:4) {
+ boxplot(iris[,i])
+ }
> par(mfrow = c(2,2))
> ##
> for (i in 1:4) {
+ boxplot(iris[,i],
+ col = i+1)
+ }
> |

Namenlos - R Editor
## for loop
data(iris)
str(iris)
par(mfrow = c(2,2))
##
for (i in 1:4) {
  boxplot(iris[,i],
    col = i+1)
}
```

RStudio

The most popular editor is RStudio, introduced in 2011, and growing rapidly in functionality.

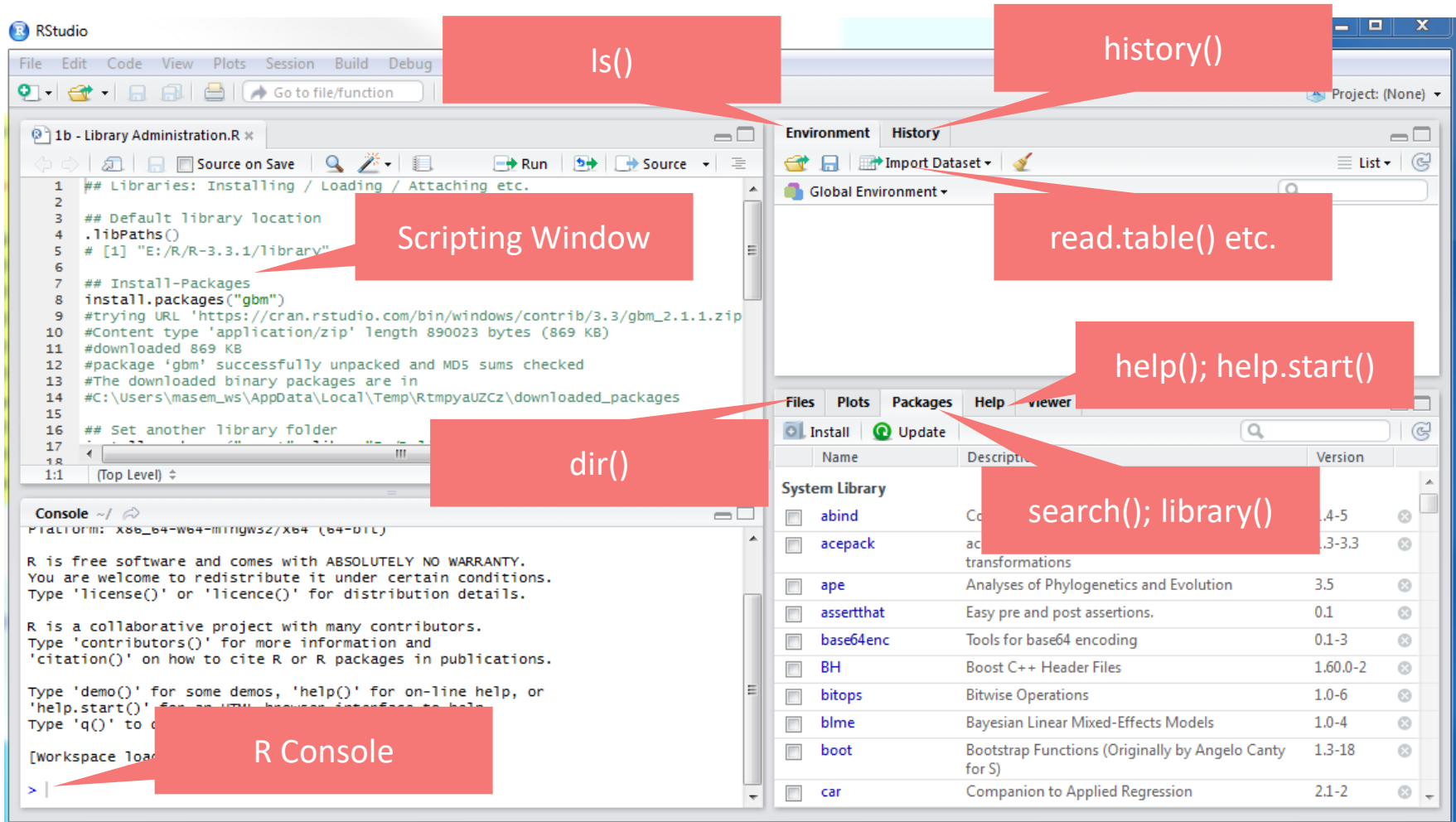


The screenshot displays the RStudio integrated development environment (IDE) interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Tools, and Help. Below the menu is a toolbar with icons for file operations and a search bar labeled 'Go to file/function'. The main workspace is divided into four panes:

- Source Editor:** Shows a file named 'Untitled1' with a single line of code '1'. The status bar at the bottom indicates '1:1 (Top Level)' and 'R Script'.
- Console:** Displays the R startup message: 'You are welcome to redistribute it under certain conditions. Type 'license()' or 'licence()' for distribution details. R is a collaborative project with many contributors. Type 'contributors()' for more information and 'citation()' on how to cite R or R packages in publications. Type 'demo()' for some demos, 'help()' for on-line help, or 'help.start()' for an HTML browser interface to help. Type 'q()' to quit R. [workspace loaded from ~/.RData]'. The prompt '>' is visible at the bottom.
- Environment:** Shows the 'Global Environment' with a search bar. Under the 'Data' tab, it lists variables: 'fcol' (chr [1:88, 1:62] "gra..."), 'fill' (chr [1:88, 1:62] "gra..."), and 'iris' (150 obs. of 5 variabl...). Buttons for 'Import Dataset' and 'List' are present.
- Help:** Displays the documentation for the 'R: Latent Position and Cluster Models for Statistical Networks' package, featuring the title and the R logo.

The bottom right corner of the window shows the page number '11'.

Rstudio and related R commands



The screenshot shows the RStudio interface with several callouts pointing to specific features and R commands:

- Scripting Window:** Points to the main editor window containing R code for installing and managing packages.
- ls():** Points to the `list.files()` function in the code.
- history():** Points to the History pane on the right.
- read.table() etc.:** Points to the Environment pane on the right.
- help(); help.start():** Points to the Help pane on the right.
- search(); library():** Points to the Packages pane on the right.
- dir():** Points to the `list.files()` function in the code.
- R Console:** Points to the console window at the bottom.

The code in the Scripting Window is as follows:

```
1 ## Libraries: Installing / Loading / Attaching etc.
2
3 ## Default library location
4 .libPaths()
5 # [1] "E:/R/R-3.3.1/library"
6
7 ## Install-Packages
8 install.packages("gbm")
9 #trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/gbm_2.1.1.zip'
10 #Content type 'application/zip' length 890023 bytes (869 KB)
11 #downloaded 869 KB
12 #package 'gbm' successfully unpacked and MD5 sums checked
13 #The downloaded binary packages are in
14 #C:\Users\masem_ws\AppData\Local\Temp\RtmpyAUZCz\downloaded_packages
15
16 ## Set another library folder
17
18
19 1:1 (Top Level) ↓
```

The console output shows the R startup message:

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded]
> |
```

Install Packages

Three Main Sources: R CRAN; Bioconductor; Git

R CRAN

Standard:

- `> install.packages("ggplot2")`

Without root access:

- `> install.packages("ggplot2", lib="/data/Rpackages/")`
- `> library(ggplot2, lib.loc="/data/Rpackages/")`

Bioconductor

Using Bioconductor:

- `> source("https://bioconductor.org/biocLite.R")`
- `> biocLite()`

Install Bioconductor Packages

- `> biocLite(c("GenomicFeatures", "AnnotationDbi"))`

Git

devtools::install_github()

- `> library(devtools)`
- `> install_github("hadley/dplyr")`

githubinstall::githubinstall()

- `> githubinstall("AnomalyDetection")`

Documentation

```
help(topic, package = NULL, lib.loc = NULL,  
      verbose = getOption("verbose"),  
      try.all.packages = getOption("help.try.all.packages"),  
      help_type = getOption("help_type"))
```

help()

- **help** is the primary interface to the help systems - Alternatively, use **?topic**
- **topic**: usually, a name or character string specifying the topic for which help is sought. A character string (enclosed in explicit single or double quotes) is always taken as naming a topic.
- **package**: a name or character vector giving the packages to look into for documentation, or NULL. By default, all packages whose namespaces are loaded are used. To avoid a name being deparsed use e.g. (pkg_ref) (see the examples).
- ...
- **help_type**: character string: the type of help required. Possible values are "text", "html" and "pdf". Case is ignored, and partial matching is allowed.

```
help()  
help(help)      # the same  
help(lapply)  
help("for")     # or ?"for", but quotes/backticks are needed  
help(package = "splines") # get help even when package is not loaded  
help("bs", try.all.packages = TRUE) # reports can be found in package 'splines'
```

Loading/Attaching and Listing of Packages

```
library(package, help, pos = 2, lib.loc = NULL,  
        character.only = FALSE, logical.return = FALSE,  
        warn.conflicts = TRUE, quietly = FALSE,  
        verbose = getOption("verbose"))
```



- **library** and **require** load and attach add-on packages.
- **package, help**: the name of a package, given as a name or literal character string, or a character string, depending on whether `character.only` is `FALSE` (default) or `TRUE`.
- **lib.loc**: a character vector describing the location of R library trees to search through, or `NULL`. The default value of `NULL` corresponds to all libraries currently known to `.libPaths()`. Non-existent library trees are silently ignored.
- **library(help = somename)**: computes basic information about the package *somename*, and returns this in an object of class "packageInfo".

```
library()           # list all available packages  
library(lib.loc = .Library) # list all packages in the default library  
library(help = splines)  # documentation on package 'splines'  
library(splines)        # attach package 'splines'  
require(splines)        # the same  
search()               # "splines", too  
detach("package:splines") # unload package  
remove.packages("gbm", lib = .libPaths()) # Remove package permanently
```

Loading/Attaching and Listing of Packages – Complete example

```
## Libraries: Installing / Loading / Attaching etc.
```

```
## Default library location
```

```
.libPaths()
```

Check default library
location

```
# [1] "E:/R/R-3.3.1/library"
```

```
## Install-Packages
```

```
install.packages("gbm")
```

Install Package –
check default values!

```
#trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.3/gbm_2.1.1.zip'
```

```
#Content type 'application/zip' length 890023 bytes (869 KB)
```

```
#downloaded 869 KB
```

```
#package 'gbm' successfully unpacked and MD5 sums checked
```

```
#The downloaded binary packages are in
```

```
#C:\Users\masem_ws\AppData\Local\Temp\RtmpyaUZCz\downloaded_packages
```


Loading/Attaching and Listing of Packages – Complete example

```
## Set another library folder  
install.packages("caret", lib = "E:/R.lib")
```

Install package in a
specific location

```
## Permanently change library location  
.libPaths <- "E:/R/R-3.3.1/library"
```

Change .libPath
permanently

```
## Load package  
# If not installed in default folder, lib is necessary
```

```
library(caret, lib = "E:/R.lib")
```

Load package - Enter lib if not installed
in default folder!

```
## Package specific help  
help(package = "caret")
```

Show package-specific help

```
## Unload package  
detach(package:caret)
```

Unload package

```
## Uninstall package  
remove.packages("caret", lib = "E:/R.lib")
```

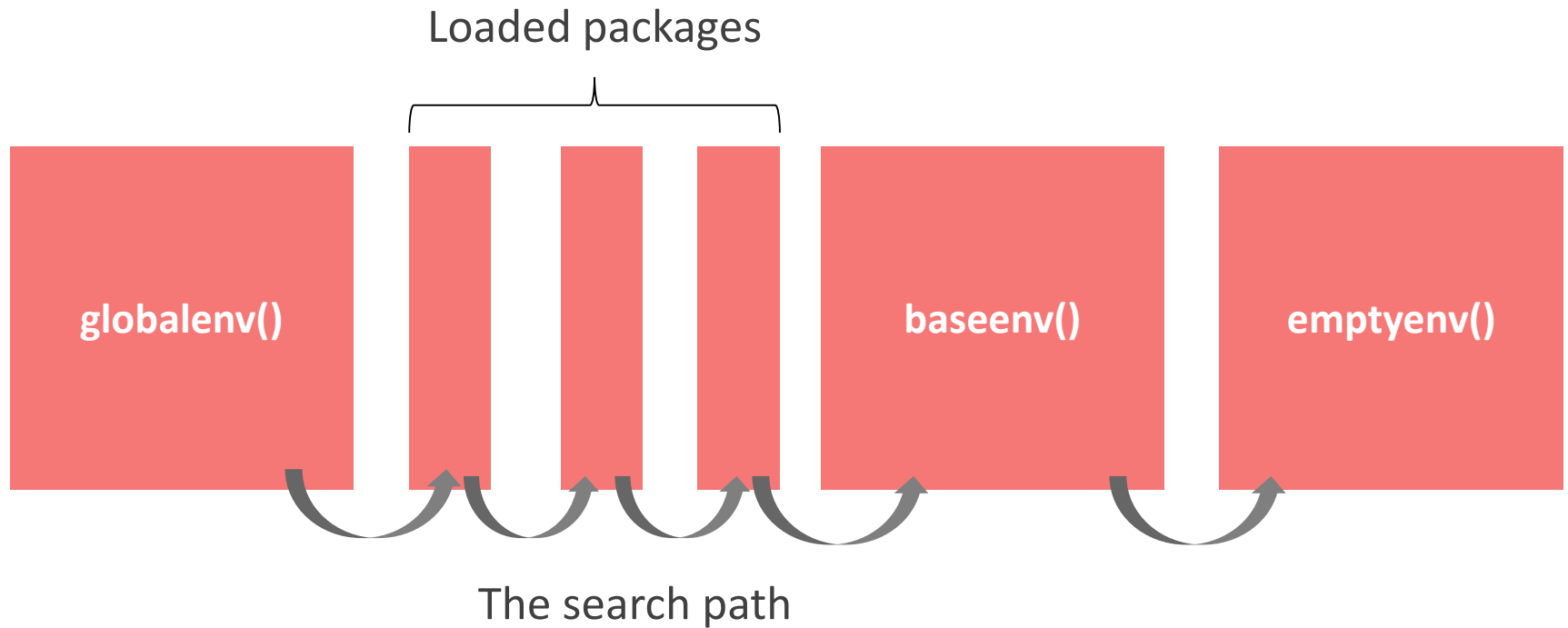
Uninstall package

Four special Environments

- | | | |
|---|----------------------------|---|
| 1 | <code>globalenv()</code> | The global environment is the <u>interactive</u> workspace. In this environment the user works. The parent of the global environment is the last package attached with <code>library()</code> or <code>require()</code> |
| 2 | <code>baseenv()</code> | The base environment is the environment of the base package. Parent is the empty environment. |
| 3 | <code>emptyenv()</code> | The empty environment is the ultimate ancestor of all environments, and the only environment without a parent. |
| 4 | <code>environment()</code> | The <code>environment()</code> is the current environment. |

`search()` lists all parents of the global environment.
`as.environment()` allows to access any environment on the search list:
`as.environment("package:stats")`

Connection between Environments and Packages



Scripting vs. Programming

Scripting

Scripting is done by users
interactively (in the console)

The **interactive** console provides help in case of warnings and errors to which the user can immediately respond

Scripts are normally stored in **.R** files and are pushed into the console

vs.

Programming

Programming is when users are writing functions or packages (collections of functions) which should work more or less **autonomously**.

Functions should check for inputs and report problems to users;

Three main pitfalls (R will be improved in the future, still an ongoing evolutionary process):

1. Type unstable functions
2. Non-standard evaluation
3. Hidden arguments

Programming: Three Main Problems

- 1 Type unstable functions** Return different types of things: with one type of input they will produce a vector, with another one, a data.frame
- 2 Non-standard evaluation** Different packages, but this causes ambiguity
- 3 Hidden arguments** Global options, like StringsAsFactors

Object Class

Class of an object is determined by its class attribute, a character vector of class names

```
## generate a vector
x <- c(1,2,5,NA)
class(x) # vector
## set class to foo
attr(x, "class") <- "foo"
x
## use structure instead
# Or in one line
x <- structure(c(1,2,5,NA), class = "foo")
x
```

Methods

A generic function has methods for certain classes

```
## Methods  
# function mean()  
methods("mean")  
# function transpose - t()  
methods("t")
```

S3 vs. S4 classes in R

R has three class systems: **S3**, **S4** and more recently **Reference class** systems.

S3 Class

- S3 class is somewhat primitive in nature.
- It lacks a formal definition and object of this class can be created simply by adding a class attribute to it. (This simplicity accounts for the fact that it is widely used in R programming language.)
- In fact most of the R built-in classes are of this type.

S4 Class

- S4 class are an improvement over S3 class.
- They have a formally defined structure which helps in making object of the same class look more or less similar.
- Class components are properly defined using the `setClass()` function and objects are created using the `new()` function.

Reference Class

- Reference class were introduced later, compared to the other two.
- It is more similar to the object oriented programming we are used to seeing in other major programming languages.
- Reference classes are basically S4 classed with an environment added to it.

S3 vs S4 vs Reference Class in R

S3 Class	S4 Class	Reference Class
Lacks formal definition	Class defined using <code>setClass()</code>	Class defined using <code>setRefClass()</code>
Objects are created by setting the class attribute	Objects are created using <code>new()</code>	Objects are created using generator functions
Attributes are accessed using <code>\$</code>	Attributes are accessed using <code>@</code>	Attributes are accessed using <code>\$</code>
Methods belong to generic function	Methods belong to generic function	Methods belong to the class
Follows copy-on-modify semantics	Follows copy-on-modify semantics	Does not follow copy-on-modify semantics

Data Import and Export

masem research institute GmbH
Bereich masem training

Unter den Eichen 5, Gebäude G,
D-65195 Wiesbaden
Registergericht Wiesbaden HRB 25700
Geschäftsführer: Dr. Guido Möser

Wissenschaftlicher Beirat:
Prof. Dr. Peter Schmidt

www.masem-training.com

Main Ressource: R Manual: R
Data Import/Export

R Data Import: Spreadsheet-like Data

```
read.table(file, header = FALSE, sep = "", quote = "\"\"",  
  dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),  
  row.names, col.names, as.is = !stringsAsFactors,  
  na.strings = "NA", colClasses = NA, nrows = -1,  
  skip = 0, check.names = TRUE, fill = !blank.lines.skip,  
  strip.white = FALSE, blank.lines.skip = TRUE,  
  comment.char = "#",  
  allowEscapes = FALSE, flush = FALSE,  
  stringsAsFactors = default.stringsAsFactors(),  
  fileEncoding = "", encoding = "unknown", text, skipNul =  
FALSE)
```

read.table()

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

R Data Import: `read.table()` – Core Settings

1	Input File Name	<code>read.table(file = ..., ...)</code>
2	Encoding	<code>read.table(encoding = "unknown", ...)</code>
3	Heading	<code>read.table(header = FALSE, ...)</code> # Default: FALSE!!
4	Row Names	<code>read.table(row.names, ...)</code>
5	Separator	<code>read.table(sep = "", ...)</code>
6	Decimal	<code>read.table(dec = ".", ...)</code>
7	Quote	<code>read.table(quote = "\"\"", ...)</code>
8	na.strings	<code>read.table(na.strings = "NA", ...)</code>
9	Strings as factor	<code>read.table(stringsAsFactors = default.stringsAsFactors(), ...)</code>
...	...	

Empty cells (numerical ones) will be automatically transformed into NA

R Data Import: Spreadsheet-like Data

```
## dataset iris
data(iris)
## write out dataset
write.table(iris, file = "iris.csv", sep = ";", dec = ",", row.names = FALSE)
dir()
## Import dataset
iris.2 <- read.table(file = "iris.dat", sep = ";", dec = ",", header = TRUE)
iris.2
## Show first six rows
head(iris.2)
## summary and structure
summary(iris.2)
```

Wrapper: read.csv() (and read.delim())

```
read.csv(file, header = TRUE, sep = ",", quote = "\"",  
         dec = ".", fill = TRUE, comment.char = "", ...)
```

```
read.csv2(file, header = TRUE, sep = ";", quote = "\"",  
          dec = ",", fill = TRUE, comment.char = "", ...)
```

read.csv()

- read.csv and read.csv2 are identical to read.table except for the defaults. They are intended for reading 'comma separated value' files ('.csv') or (read.csv2) the variant used in countries that use a comma as decimal point and a semicolon as field separator. Similarly, read.delim and read.delim2 are for reading delimited files, defaulting to the TAB character for the delimiter. Notice that header = TRUE and fill = TRUE in these variants, and that the comment character is disabled.

```
## Comma Separated Values  
read.csv2(file = "iris.csv")  
## Show first six rows  
head(iris.2)  
## summary and structure  
summary(iris.2)
```

- Import ASCII-style datasets, like *.csv;
*.dat or *.txt;
- Import MS Excel-Files;
- Import Statistical Files (SPSS, SAS, Stata);



Use RStudio Import Dataset Button to Import Datasets

Import Text Data

File/Url:
G:/OneDrive/TRAINING/Folien/Kompaktkurs R/R.datafiles/diamonds_cor_dec.points.csv Browse...

Data Preview:

carat (double) ▾	cut (character) ▾	color (character) ▾	clarity (character) ▾	depth (double) ▾	table (integer) ▾	price (integer) ▾	x (double) ▾	y (double) ▾	z (double) ▾
0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	
0.21	Premium	F	SI1	59.8	61	326	3.89	3.84	
0.23	Good	G	SI2	56.9	65	327	4.05	4.07	
0.29	Premium	I	VS2	62.4	58	334	4.20	4.23	
0.31	Good	J	SI2	63.3	58	335	4.34	4.35	
0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48
0.24	Very Good	I	VVS1	62.3	57	336	3.95	3.98	2.47
0.26	Very Good	H	SI1	61.9	55	337	4.07	4.11	2.53
0.23	Very Good	F	VS2	65.1	61	337	3.87	3.78	2.49

Strings as factors cannot be set:
Rationale (Recommendation!): Import characters, transform in factor in R later on!

Set Heading to Yes (default is No!)

Import Options:

Name: diamonds_cor_dec_points ☒ First Row as Names
 Skip: 0 ☒ Trim Spaces
☒ Open Data Viewer
 Delimiter: Semicolon Escape: None
 Quotes: Default Comment: Default
 Locale: Configure... NA: Default

Code Preview:

```
library(readr)
diamonds_cor_dec_points <- read_delim("G:/OneDrive/TRAINING/Folien/Kompaktkurs R/R.datafiles/diamonds_cor_dec.points.csv",
  ";", escape_double = FALSE, trim_ws = TRUE)
view(diamonds_cor_dec_points)
```

Import Cancel

Empty cells will be automatically transformed into NA (Not Available / Missing Values)!

ODBC: Databases

1

RODBC

All purpose ODBC Driver

Available DB Specific Drivers (surely incomplete...)

2

RMySQL

6

RMongo; mongolite

3

ROracle

4

RPostgreSQL

5

RSQLite

1

RJDBC

Uses Java and can connect to any DBMS that has a JDBC driver

RODBC – How it works!

1	Load Driver Package	<code>install.packages("R2MySQL", dependencies = TRUE)</code> <code>library(R2MySQL)</code>
2	Establish Connection to DB	Database Information needed: DNS, UID, PW etc.
3	Check Connection	Check if connection works
...	Perform Typical Tasks	Perform Typical Tasks
LS	Close Connection	Last step is to close the connection to the database!

RODBC – Common Functions

Function	Decription
<code>odbcDriverConnect()</code>	Open a connection to an ODBC database
<code>sqlQuery()</code>	Submit a query to an ODBC database and return the results
<code>sqlTables()</code>	List Tables on an ODBC Connection
<code>sqlFetch()</code>	Read a table from an ODBC database into a data frame
<code>sqlColumns()</code>	Query Column Structure in ODBC Tables
<code>close(connection)</code>	Close the connection

R2MySQL

dbConnect(drv, ...)

dbConnect()

drv

An object that inherits from DBIDriver, or an existing DBIConnection object (in order to clone an existing connection).

...

Authentication arguments needed by the DBMS instance; these typically include

- user,
- password,
- host,
- port,
- dbname,
- etc.

```
drv = dbDriver("MySQL")
con <- dbConnect(drv,
  user = 'root',
  password = 'masemsql',
  host = '192.168.2.111',
  dbname = 'diamonds',
  port = 3306)
```

Import SPSS datasets

1

foreign::read.spss()

read.spss reads a file stored by the SPSS save or export commands.

Note: This was originally written in 2000 and has limited support for changes in SPSS formats since (which have not been many).

2

Hmisc::spss.get()

spss.get: Enhanced Importing of SPSS Files

- `spss.get` invokes the `read.spss` function in the `foreign` package to read an SPSS file, with a **default output format** of "data.frame".
- The `label` function is used to attach labels to individual variables instead of to the data frame as done by `read.spss`.
- By default, integer-valued variables are converted to a storage mode of integer unless `force.single=FALSE`.
- Date variables are converted to R Date variables.
- By default, underscores in names are converted to periods.

3

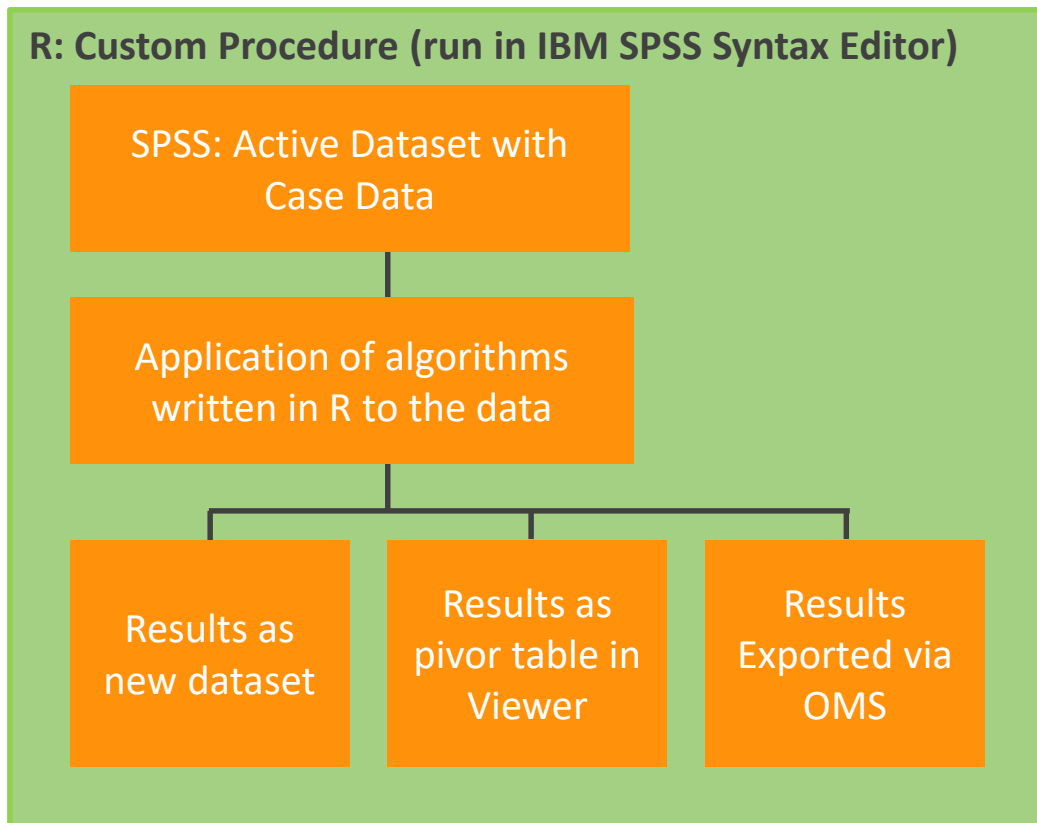
haven

3

read.table()

Export dataset in csv-format and export in R using `read.table()`-function (or wrapper)!

Official IBM SPSS Statistics® and R Integration



- R programs can access SPSS Statistics variable dictionary information, case data, procedure output, create new datasets, output in the form of pivot tables and R graphics
- Analyze data with an own written R functions or use functions from the extensive set of statistical routines available with R
- All is possible from within SPSS Statistics

Importing from other Statistical Systems: Package foreign

```
read.spss(file, use.value.labels = TRUE, to.data.frame = FALSE,  
  max.value.labels = Inf, trim.factor.names = FALSE,  
  trim_values = TRUE, reencode = NA, use.missings =  
    to.data.frame)
```

read.spss()

- Reads an SPSS Data File
- This was originally written in 2000 and has limited support for changes in SPSS formats since (which have not been many).

```
## Importing from other statistical systems  
library(foreign)  
setwd("D:\\OneDrive\\TRAINING\\Folien\\Kompaktkurs R\\R.datafiles")  
## Check if file exists  
dir()  
## Read SPSS demo dataset  
demo.spss <- read.spss("demo.sav", to.data.frame = TRUE)  
## convert variables with value labels into R factors with those levels  
demo2.spss <- read.spss("demo.sav", to.data.frame = TRUE, use.value.labels = FALSE)  
head(demo2.spss)
```


Data Import and Export

SPECIAL SECTION: MS EXCEL

1st Option: Export into csv-File: Recommendation by R Manual

R Manual: The most common R data import/export question seems to be ‘how do I read an Excel spreadsheet’.

This chapter collects together advice and options given earlier. Note that most of the advice is for pre-Excel 2007 spreadsheets and not the later .xlsx format.

The first piece of advice is to avoid doing so if possible! If you have access to Excel, export the data you want from Excel in tab-delimited or comma-separated form, and use `read.delim` or `read.csv` to import it into R. (You may need to use `read.delim2` or `read.csv2` in a locale that uses comma as the decimal point.) Exporting a DIF file and reading it using `read.DIF` is another possibility.

Overview

Package	Notes
RODBC::odbcConnectExcel()	MS Excel 97-2003, 32-bit R only!
RODBC::odbcConnectExcel2007()	MS Excel 2007 formats (xlsx); maybe 64-bit Windows is supported;
gdata::read.xls()	Perl installation necessary; supports only xls-format;
Packages dataframes2xls and WriteXLS	Write data frames, Python/Perl required
xlsx	Reads xlsx-files; Recommended! Requires JAVA-installation; JAVA path must be set in R;
XLConnect	Can read, write, and manipulate both Excel 97-2003 and Excel 2007/10 spreadsheets; JAVA required; Recommended!
readxl	Can read both Excel 97-2003 and Excel 2007/10 spreadsheet, using an included C library; Recommended!

read_excel() – Hadley Wickham

```
read_excel(path, sheet = 1, col_names = TRUE, col_types = NULL,  
na = "", skip = 0)
```

read.xl()

- **path:** Path to the xls/xlsx file
- **sheet:** Sheet to read. Either a string (the name of a sheet), or an integer (the position of the sheet). Defaults to the first sheet.
- **col_names:** Either TRUE to use the first row as column names, FALSE to number columns sequentially from X1 to Xn, or a character vector giving a name for each column.
- **col_types:** Either NULL to guess from the spreadsheet or a character vector containing "blank", "numeric", "date" or "text".
- **na:** Missing value. By default readxl converts blank cells to missing data. Set this value if you have used a sentinel value for missing values.
- **skip:** Number of rows to skip before reading any data.

Rstudio: readxl() – Hadley Wickham

Import Excel Data

File/Url:

Data Preview:

Zip (character) ▾	State (character) ▾	Country (character) ▾
A0A	Newfoundland and Labrador	Canada
A0B	Newfoundland and Labrador	Canada
A0C	Newfoundland and Labrador	Canada
A0E	Newfoundland and Labrador	Canada
A0G	Newfoundland and Labrador	Canada
A0H	Newfoundland and Labrador	Canada
A0J	Newfoundland and Labrador	Canada
A0K	Newfoundland and Labrador	Canada
A0L	Newfoundland and Labrador	Canada
A0M	Newfoundland and Labrador	Canada
A0N	Newfoundland and Labrador	Canada
A0P	Newfoundland and Labrador	Canada
A0R	Newfoundland and Labrador	Canada
A1A	Newfoundland and Labrador	Canada
A1B	Newfoundland and Labrador	Canada
A1C	Newfoundland and Labrador	Canada

Preview: data frame

Select Sheet to import:

Import Options

Name: ☒ First Row as Names

Sheet: NA:

Skip: ☒ Open Data Viewer

Code Preview:

```
library(readxl)
lab1 <- read_excel("G:/OneDrive/TRAINING/Folien/Kompaktkurs R/R.datafiles/lab1.xlsx",
  sheet = "Locations")
view(lab1)
```

MS Excel with RODBC

```
odbcConnectExcel(xls.file, readOnly = TRUE, ...)  
odbcConnectExcel2007(xls.file, readOnly = TRUE, ...)  
odbcConnectAccess(access.file, uid = "", pwd = "", ...)  
odbcConnectAccess2007(access.file, uid = "", pwd = "", ...)
```

odbcConnectExcel()

- Outdated: For many years it was the easiest solutions based on R code for Windows XP, 7 etc. users. Nowadays it still support only 32 bit versions of R and this limit discourage the use of this package.
- **Besides Microsoft Windows and 32-bit R, it requires the two Excel ODBC driver installed: One for xls and one for xlsx!**

```
## Using RODBC to import MS Excel files  
library(RODBC)  
conn = odbcConnectExcel("myfile.xlsx") # open a connection to the Excel file  
sqlTables(conn)$TABLE_NAME # show all sheets  
df = sqlFetch(conn, "Sheet1") # read a sheet  
df = sqlQuery(conn, "select * from [Sheet1 $]") # read a sheet (alternative SQL syntax)  
close(conn) # close the connection to the file
```


Objects and Data Types in R

masem research institute GmbH
Bereich masem training

Unter den Eichen 5, Gebäude G,
D-65195 Wiesbaden
Registergericht Wiesbaden HRB 25700
Geschäftsführer: Dr. Guido Möser

Wissenschaftlicher Beirat:
Prof. Dr. Peter Schmidt

www.masem-training.com

Objects

Empty vectors can be created with the `vector()` function (see next slide)!

R has five basic or “atomic” classes of objects:

1

character

```
x <- c("a", "b", "c") ## character
```

2

numeric (real numbers)

```
x <- c(0.5, 0.6) ## numeric
```

3

integer

```
x <- 9:29 ## integer  
x <- c(1L, 3L) ## integer
```

4

complex

```
x <- c(1+0i, 2+4i) ## complex
```

5

logical (TRUE/FALSE)

```
x <- c(TRUE, FALSE) ## logical  
x <- c(T, F) ## logical
```

The most basic object is a vector: A vector can only contain objects of the **same class**!

BUT: The one exception is a list, which is represented as a vector but can contain objects of different classes.

R Objects and Attributes

R Objects can have attributes

- 1 names, dimnames
- 2 dimensions (e.g. matrices, arrays)
- 3 class
- 4 length
- 5 Other user-defined attributes /metadata

Attributes of an object can be accessed using the `attributes()` function

Numbers

Numbers in R are generally treated as numeric objects (i.e. double precision real numbers)

- If you explicitly want an integer, you need to specify the L suffix
 - Entering 1 gives you a numeric object;
 - Entering 1L explicitly gives you an integer.

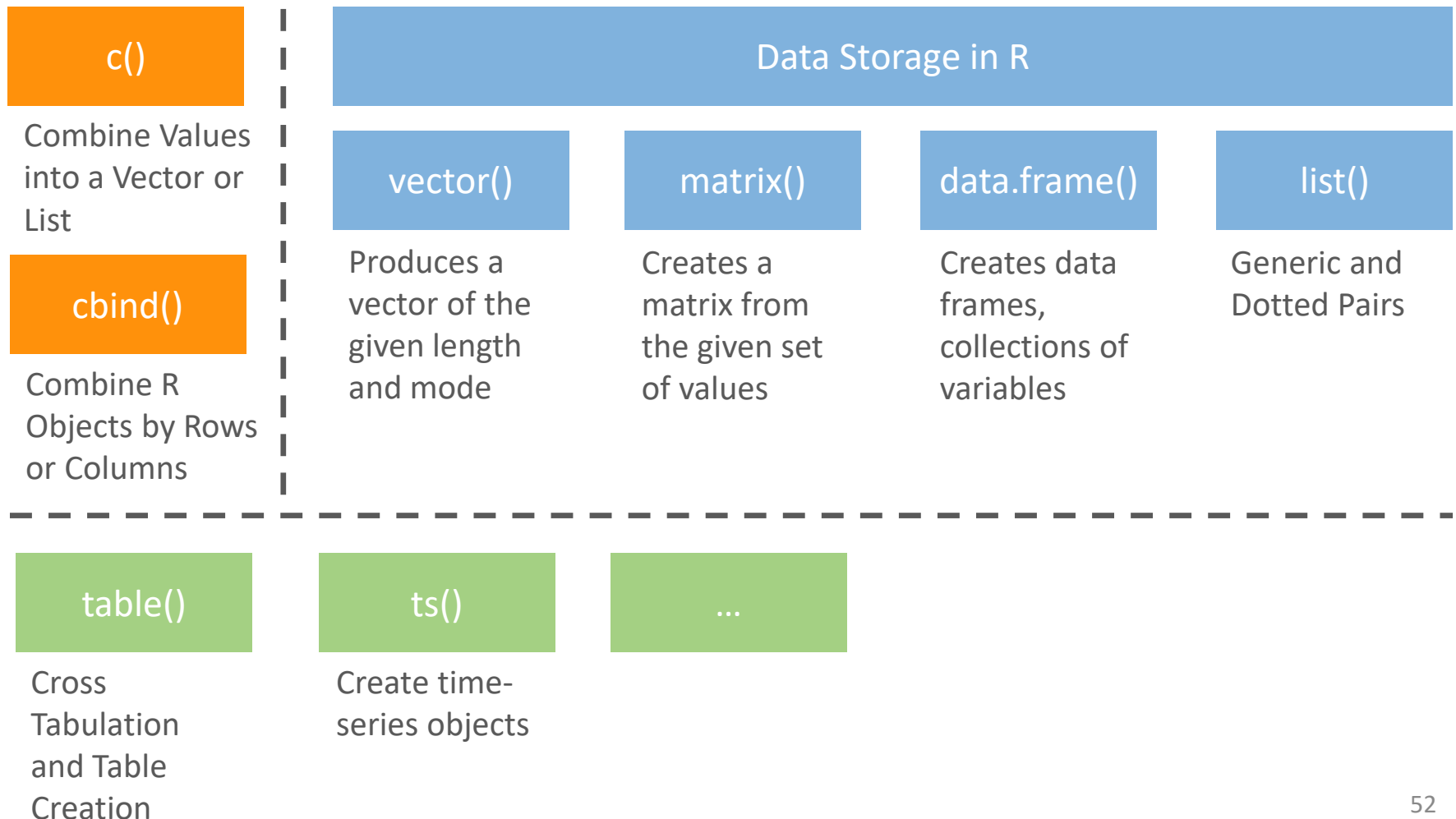
There is also a special number Inf which represents **Infinity**; e.g. $1 / 0$;

- Inf can be used in ordinary calculations; e.g. $1 / \text{Inf}$ is 0

The value NaN represents an **undefined value** ("not a number"); e.g. $0 / 0$;

- NaN can also be thought of as a missing value (more on that later)

Data Types, Data Storage, Data Handling etc. in R: An incomplete overview



Data Types etc. in R

WORKING WITH DATA

Combine values into a vector or list

```
c(..., recursive = FALSE)
```

```
c()
```

- Combine values into a Vector or List
- The default method combines list arguments to form a **vector**.
- The output type is determined from the highest type of the components in the hierarchy **NULL < raw < logical < integer < double < complex < character < list < expression**

```
c(1,7:9)
```

```
c(1:5, 10.5, "next")
```

```
## uses with a single argument to drop attributes
```

```
x <- 1:4
```

```
names(x) <- letters[1:4]
```

```
x
```

Combine values into a vector or list

```
cbind(..., deparse.level = 1)  
rbind(..., deparse.level = 1)
```

cbind()

- Take a sequence of vector, matrix or data-frame arguments and combine by columns or rows, respectively.
- `deparse.level`: integer controlling the construction of labels in the case of non-matrix-like arguments (for the default method):
 - `deparse.level = 0` constructs no labels;
 - `deparse.level = 1` or `2` constructs labels from the argument names

```
## cbind  
id <- 1:10  
v1 <- rnorm(10)  
v2 <- trunc(runif(10)*100)  
df1 <- cbind(id, v1, v2)  
class(df1)
```

Exercises cbind()

Exercise: Create a data.frame – enter also a running number (ID)!

age	glucose	income.tsd	cost.tsd
43	6,3	31,48	2,63
44	7,6	32,39	7,71
27	7,2	42,75	1,43
29	6,2	45,23	6,40
20	8,7	28,69	9,11
51	6,1	36,80	3,56
25	6,4	28,95	3,17
46	8,4	32,06	20,12
27	5,9	35,82	5,62
38	8,2	33,55	16,18

Produce a Data Frame

```
data.frame(..., row.names = NULL, check.rows = FALSE,  
           check.names = TRUE, fix.empty.names = TRUE,  
           stringsAsFactors = default.stringsAsFactors())
```

data.frame()

- **row.names:** NULL or a single integer or character string specifying a column to be used as row names
- **stringsAsFactors:** logical: should character vectors be converted to factors?
- ...

```
# data.frame  
L3 <- LETTERS[1:3]  
fac <- sample(L3, 10, replace = TRUE)  
(d <- data.frame(x = 1, y = 1:10, fac = fac))
```

Data Frames

- They are represented as a special type of list where every element of the list has to have the same length
- Each element of the list can be thought of as a column and the length of each element of the list is the number of rows
- Unlike matrices, data frames can store different classes of objects in each column (just like lists); matrices must have every element be the same class
- Data frames also have a special attribute called `row.names`
- Data frames are usually created by calling `read.table()` or `read.csv()`
- Can be converted to a matrix by calling `data.matrix()`

Produce a Vector

```
vector(mode = "logical", length = 0)
as.vector(x, mode = "any")
is.vector(x, mode = "any")
```

vector()

- **vector** produces a vector of the given length and mode.
- **modes** are "logical", "integer", "numeric" (synonym "double"), "complex", "character" and "raw"
- **length** is a positive integer specifying the length of the vector

```
## vector()
# Creates an empty vector with mode integer and length of 10
vector(mode = "integer", length = 10)

df <- data.frame(x = 1:3, y = 5:7)
## Error:
try(as.vector(data.frame(x = 1:3, y = 5:7), mode = "numeric"))
```

Produce a Matrix

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,  
       dimnames = NULL)
```

matrix()

- **Matrix** creates a matrix from the given set of values.
- **Data**: optional data vector.
- **nrow**: the desired number of rows
- **ncol**: the desired number of columns
- **byrow**: matrix is filled by columns or rows
- **dimnames**: row or column names

Produce a List

```
list(...)
```

```
list()
```

- Functions to construct, coerce and check for both kinds of R lists.
- ...

```
# list  
# create a plotting structure  
pts <- list(x = cars[,1], y = cars[,2])  
plot(pts)
```

Factors

Factors are used to represent categorical data.

Factors can be unordered or ordered.

(A factor is like an integer vector where each integer has a label.)

- Factors are treated specially by modelling functions like `lm()` and `glm()`

Using factors with labels is better than using integers because **factors are self-describing**;

- A variable that has values "East" and "West" is better than a variable that has values 1 and 2.

Note: One nasty behavior of R is option `StringsAsFactors`: Character Variables will be automatically transformed into factor-variables.

Factors- Example

```
> x <- factor(c("yes", "yes", "no", "yes", "no"))
> x
[1] yes yes no yes no
Levels: no yes
> table(x)
x
no yes
2 3
> unclass(x)
[1] 2 2 1 2 1
attr(,"levels")
[1] "no" "yes"
18
```

Factors: Order

The order of the levels can be set using the levels argument to factor().

Contrast Coding: In linear modelling often the first level is used as the baseline level.

```
> x <- factor(c("yes", "yes", "no", "yes", "no"),  
levels = c("yes", "no"))  
> x  
[1] yes yes no yes no  
Levels: yes no
```


Data Types etc. in R

WORKING WITH DATA

Entering Input: Type in expressions at the R prompt

The <- symbol is the assignment operator (see slide)

```
> x <- 5L
> print(x)
[1] 1
> x
[1] 1
> text <- "Hello World!"
```

The grammar of the language determines whether an expression is complete or not.

```
> x <- ## Incomplete expression
```

The # character indicates a comment. Anything to the right of the # (including the # itself) is ignored.

R Console: Interactive Working: Evaluation

When a complete expression is entered at the prompt, it is evaluated and the result of the evaluated expression is returned. The result may be auto-printed.

```
> x <- 5 ## nothing printed
> x ## auto-printing occurs
[1] 5
> print(x) ## explicit printing
[1] 5
```

The [1] indicates that x is a vector and 5 is the first element.

Assignment-Operator

```
x <- value  
x <<- value  
value -> x  
value ->> x  
x = value
```



- **There are three different assignment operators:** two of them have leftwards and rightwards forms.
- The operators `<-` and `=` assign into the environment in which they are evaluated. The operator `<-` can be used anywhere, whereas the operator `=` is only allowed at the top level
- The operators `<<-` and `->>` are normally only used in functions, and cause a search to be made through parent environments for an existing definition of the variable being assigned.

```
## <-  
x <- c(1:10)  
# copy an object  
iris -> y  
z <- 0
```

Mising Objects

When different objects are mixed in a vector, coercion occurs so that every element in the vector is of the same class.

```
> y <- c(1.7, "a") ## character  
> y <- c(TRUE, 2) ## numeric  
> y <- c("a", TRUE) ## character
```

Data Types etc. in R

DATA INFORMATION

Data Information

1

`ls()`

List objects in the global environment (working environment)

Object Specific

2

`names(object)`

List the variables in object

3

`str(object)`

List the structure of object

4

`levels(object[factor])`

List levels of a factor

5

`dim(object)`

Dimensions of an object

6

`class(object)`

Class of an object (matrix, data frame, list etc.)

7

`head(object, n = 6L)`

Print first six rows of an object

8

`tail(object, n = 6L)`

Print last six rows of an object

9

`is.family`

Functions to test inheritance relationships between an object and a class

Is an object from a Class?

```
is(object, class1)
```

```
is()
```

Functions to test inheritance relationships between an object and a class (is) or between two classes (extends), and to establish such relationships

- **object:** any R object.
- **class1:** the names of the classes between which is relations are to be examined defined, or (more efficiently) the class definition objects for the classes.

There are several functions like `is.na()`, `is.numeric()`, `is.list()` etc. which can be used instead (class1 has not to be defined in functions)

```
is.na(c(1, NA))    #> FALSE TRUE
is.na(paste(c(1, NA))) #> FALSE FALSE

## ... lists, and hence data frames, too:
dN <- dd <- USJudgeRatings; dN[3,6] <- NA
anyNA(dd) # FALSE
anyNA(dN) # TRUE
```


Data Types etc. in R

DATA TRANSFORMATION / EXPLICIT COERCION

Explicit Coercion

Objects can be explicitly coerced from one class to another using the **as.* functions**, if available.

```
> x <- 0:6
> class(x)
[1] "integer"
> as.numeric(x)
[1] 0 1 2 3 4 5 6
> as.logical(x)
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
> as.character(x)
[1] "0" "1" "2" "3" "4" "5" "6"
> as.complex(x)
[1] 0+0i 1+0i 2+0i 3+0i 4+0i 5+0i 6+0i
```

Explicit Coercion: Nonsensical Coercion results in NAs

Nonsensical Coercion results in **NAs**

```
> x <- c("a", "b", "c")
> as.numeric(x)
[1] NA NA NA
Warning message:
NAs introduced by coercion
> as.logical(x)
[1] NA NA NA
```

Data Types etc. in R

DATE AND TIME OBJECTS IN R

Overview

Besides built-in functions, many helpful functions in external packages available

Built-in Functions

Classes:

- Class **Date** is dates without times;
- **POSIXct** and **POSIXlt** are classes with dates and times;

Functions:

- **strptime()**: Functions to convert between character representations and objects of classes „POSIXlt“ and „POSIXct“ representing calendar dates and times.

External Packages

Package lubridate provides helpful functions to manipulate dates and times and provides the following object classes:

- Interval class: time between two specific instants;
- Duration class: time spans with exact length;
- Period class: time spans that may not have a consistent length in seconds;

Time Series Objects

Time Series Object Package		Description
fts	fts	An R interfact to tslib (a time series library in C++)
its	its	An S4 class for handling irregular time series
irts	tseries	irts objects are irregular time-series objects. These are scalar or vector valued time series indexed by a time-stamp of class POSIXct.
timeSeries	timeSeries	Rmetrics package of time series tools and utilities. Similar to the Tibco S-PLUS timeSeries class
ti	tis	Functions and S3 classes for time indexes and time indexed series, which are compatible with FAME frequencies
ts, mts	stats	Regularly spaced time series objects
zoo	zoo	S3 class of indexed totally ordered observations which includes irregular time series.
xts	xts	Extension of the zoo class

POSIXct and POSIXlt

POSIXlt

The POSIXct data type is the number of seconds since the start of January 1, 1970.

Negative numbers represent the number of seconds before this time, and positive numbers represent the number of seconds afterwards.

POSIXct

The POSIXlt data type is a vector, and the entries in the vector have the following meanings:

1. seconds
2. minutes
3. hours
4. day of month (1-31)
5. month of the year (0-11)
6. years since 1900
7. day of the week (0-6 where 0 represents Sunday)
8. day of the year (0-365)
9. Daylight savings indicator (positive if it is daylight savings)

POSIXct & POSIXlt

```
as.POSIXct(x, tz = "", ...)  
as.POSIXlt(x, tz = "", ...)
```

```
as.POSIXct()
```

```
as.POSIXlt()
```

Functions to manipulate objects of classes "POSIXlt" and "POSIXct" representing calendar dates and times.

- **x**: An object to be converted.
- **tz**: A time zone specification to be used for the conversion, if one is required. System-specific (see time zones), but "" is the current time zone, and "GMT" is UTC (Universal Time, Coordinated). Invalid values are most commonly treated as UTC, on some platforms with a warning.
- **...**: further arguments to be passed to or from other methods.
- **format**: character string giving a date-time format as used by `strptime`.

```
(now <- as.POSIXlt(Sys.time())) # the current datetime, as class "POSIXlt"  
unlist(unclass(now))           # a list shown as a named vector  
now$year + 1900                 # see ?DateTimeClasses  
months(now); weekdays(now)     # see ?months
```


Convert character variables containing date and time in POSIXct() objects

Use `strptime()` function to convert

```
as.POSIXct(strptime("2011-03-27 01:30:00", "%Y-%m-%d %H:%M:%S"))  
as.POSIXct(strptime("2010-10-31 01:30:00", "%Y-%m-%d %H:%M:%S"))
```

?strptime for more information
about abbreviations used

Indexing in R

masem research institute GmbH
Bereich masem training

Unter den Eichen 5, Gebäude G,
D-65195 Wiesbaden
Registergericht Wiesbaden HRB 25700
Geschäftsführer: Dr. Guido Möser

Wissenschaftlicher Beirat:
Prof. Dr. Peter Schmidt

www.masem-training.com

Subsetting

There are a number of operators that can be used to extract subsets of R objects.

- `[` always returns an object of the same class as the original;
 - can be used to select more than one element
 - there is one exception: One Vector data.frames return a vector
- `[[` is used to extract elements of a list or a data frame;
 - it can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame
- `$` is used to extract elements of a list or data frame by name;
 - semantics are similar to hat of `[[`.

List subset (binary): \$

```
data.frame$col.name
```



- **Subset variables in a data.frame or matrix if col.names are given**

```
## subset variables  
data(iris)  
iris$Species
```

Indexing Vectors: Four (Five) distinct types

- | | | |
|---|---|--|
| 1 | A logical vector | In this case the index vector is recycled to the same length as the vector from which elements are to be selected. |
| 2 | A vector of positive integral quantities | In this case the values in the index vector must lie in the set $\{1, 2, \dots, \text{length}(x)\}$. |
| 3 | A vector of negative integral quantities | A vector of negative integral quantities. Such an index vector specifies the values to be excluded rather than included. |
| 4 | A vector of character strings | This possibility only applies where an object has a names attribute to identify its components. |
-
- | | | |
|---|--------------------------|---|
| 5 | Function subset() | Return subsets of vectors, matrices or data frames which meet conditions. |
|---|--------------------------|---|

1 – A Logical Vector

In this case the index vector is recycled to the same length as the vector from which elements are to be selected: Values corresponding to TRUE in the index vector are selected and those corresponding to FALSE are omitted.

```
## create a numerical vector
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
## Select cases number 3, 4 and 7
# Create filter vevtor (logical vector)
filter <- c(FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE)
x[filter]
```

2 – A vector of positive integral quantities

In this case the values in the index vector must lie in the set $\{1, 2, \dots, \text{length}(x)\}$. The corresponding elements of the vector are selected and concatenated, in that order, in the result. The index vector can be of any length and the result is of the same length as the index vector.

```
## create a numerical vector  
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
## Select 5th value  
x[5]  
## Select cases number 3, 4 and 7  
x[c(3,4,7)]
```


3 – A vector of negative integral quantities

Such an index vector specifies the values to be excluded rather than included.

```
## create a numerical vector  
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
## All but the first five elements  
x[-c(1:5)]  
## Select cases number 3, 4 and 7  
x[-c(1, 2, 5, 6, 8, 9, 10)]
```

4 – A vector of character strings

This possibility only applies where an object has a names attribute to identify its components. In this case a sub-vector of the names vector may be used in the same way as the positive integral labels in item 2 further above.

```
## Create vector with names  
fruit <- c(5, 10, 1, 20)  
names(fruit) <- c("orange", "banana", "apple", "peach")  
## Select only apple and orange  
lunch <- fruit[c("apple", "orange")]
```

5 - Subsetting Vectors, Matrices and Data Frames

```
subset(x, subset, select, drop = FALSE, ...)
```

```
subset()
```

- **Return subsets of vectors, matrices or data frames which meet conditions**
- **x**: object to be subsetting.
- **subset**: logical expression indicating elements or rows to keep: missing values are taken as false.
- **select**: expression, indicating columns to select from a data frame.
- **drop**: passed on to [indexing operator
- ...

```
## subset  
subset(airquality, Temp > 80, select = c(Ozone, Temp))  
subset(airquality, Day == 1, select = -Temp)  
subset(airquality, select = Ozone:Wind)
```

Sorting or Ordering Vectors

```
sort(x, decreasing = FALSE, na.last = NA, ...)
```

```
sort()
```

- **Sort (or order) a vector or factor (partially) into ascending or descending order.**
- For ordering along more than one variable, e.g., for sorting data frames, see `order`.

```
## sort vector  
x <- swiss$Education[1:25]  
x;  
sort(x);
```

Sorting complete data frames: Ordering Permutation

```
order(..., na.last = TRUE, decreasing = FALSE,  
method = c("shell", "radix"))
```

order()

- **order** returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments.
- **order can be used to sort complete data frames.**
- **sort.list** is the same, using only one argument.

```
## sort data frames  
data(iris)  
# sort by Sepal.Length  
iris[ order(iris$Sepal.Length), ]  
# Sort by Species and Sepal.Length  
iris[ order(iris$Species, iris$Sepal.Length), ]
```

Subsetting Lists

Subsetting Lists

```
> x <- list(foo = 1:4, bar = 0.6)
> x[1]
$foo
[1] 1 2 3 4
> x[[1]]
[1] 1 2 3 4
> x$bar
[1] 0.6
> x[["bar"]]
[1] 0.6
> x["bar"]
$bar
[1] 0.6
```

Subsetting Lists

Extracting multiple elements of a list

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")  
> x[c(1, 3)]  
$foo  
[1] 1 2 3 4  
$baz  
[1] "hello"
```

Subsetting Lists

The `[[` operator can be used with computed indices; `$` can only be used with literal names.

```
> x <- list(foo = 1:4, bar = 0.6, baz = "hello")
> name <- "foo"
> x[[name]] ## computed index for `foo`
[1] 1 2 3 4
> x$name ## element `name` doesn't exist!
NULL
> x$foo
[1] 1 2 3 4 ## element `foo` does exist
```


Subsetting Nested Elements of a List

The `[[` can take an integer sequence.

```
> x <- list(a = list(10, 12, 14), b = c(3.14, 2.81))  
> x[[c(1, 3)]]  
[1] 14  
> x[[1]][[3]]  
[1] 14  
> x[[c(2, 1)]]  
[1] 3.14
```

Working with Data

JOIN-OPERATIONEN

JOIN-Operations with R - Packages

- | | | |
|---|---------------------------------|---|
| 1 | base::merge() | <code>merge(df1, df2, by = "CustomerId")</code> |
| 2 | sqldf:sqldf() | <pre>## inner join df3 <- sqldf("SELECT CustomerId, Product, State FROM df1 JOIN df2 USING(CustomerID)")</pre> |
| 3 | dplyr: inner_join() etc. | <code>inner_join(df1, df2, by = "x")</code>
<code>dplyr = inner_join(df1, df2, by = "x")</code> |
| 4 | data.table | <code>dt1[dt2, nomatch = 0L, on = "x"]</code> |

data.table + dplyr code → library
dtplyr

Merge Two Data Frames: merge()

```
merge(x, y, by = intersect(names(x), names(y)),  
      by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all,  
      sort = TRUE, suffixes = c(".x", ".y"),  
      incomparables = NULL, ...)
```

merge()

- Merge two data frames by common columns or row names, or do other versions of database *join* operations.
- by, by.x, by.y specifications of the columns used for merging – **in quotation marks!**
- ...

Merge Two Data Frames

```
## use character columns of names to get sensible sort order
authors <- data.frame(
  surname = I(c("Tukey", "Venables", "Tierney", "Ripley", "McNeil")),
  nationality = c("US", "Australia", "US", "UK", "Australia"),
  deceased = c("yes", rep("no", 4)))
books <- data.frame(
  name = I(c("Tukey", "Venables", "Tierney",
    "Ripley", "Ripley", "McNeil", "R Core")),
  title = c("Exploratory Data Analysis",
    "Modern Applied Statistics ...",
    "LISP-STAT",
    "Spatial Statistics", "Stochastic Simulation",
    "Interactive Data Analysis",
    "An Introduction to R"),
  other.author = c(NA, "Ripley", NA, NA, NA, NA,
    "Venables & Smith"))

(m1 <- merge(authors, books, by.x = "surname", by.y = "name"))
(m2 <- merge(books, authors, by.x = "name", by.y = "surname"))
## "R core" is missing from authors and appears only here :
merge(authors, books, by.x = "surname", by.y = "name", all = TRUE)
```

Merge Two Data Frames: merge(): JOINS

- | | | |
|---|------------------|---|
| 1 | Inner join | <code>merge(x = df1, y = df2, by = "CustomerId")</code> |
| 2 | Outer join | <code>merge(x = df1, y = df2, by = "CustomerId", all = TRUE)</code> |
| 3 | Left outer join | <code>merge(x = df1, y = df2, by = "CustomerId", all.x = TRUE)</code> |
| 4 | Right outer join | <code>merge(x = df1, y = df2, by = "CustomerId", all.y = TRUE)</code> |
| 5 | Cross Join | <code>merge(x = df1, y = df2, by = NULL)</code> |

Merge using more than one column

```
merge(x=df1,y=df2, by.x=c("x_col1","x_col2"),  
      by.y=c("y_col1","y_col2"))
```

Gabor Grothendieck's sqldf package

```
sqldf::sqldf(x, stringsAsFactors = FALSE,  
row.names = FALSE, ...)
```

sqldf()

- x: Character string representing an SQL select statement or character vector whose components each represent a successive SQL statement to be executed. The select statement syntax must conform to the particular database being used. If x is missing then it establishes a connection which subsequent sqldf statements access. In that case the database is not destroyed until the next sqldf statement with no x;
- ...

1

Inner join

```
## inner join  
df3 <- sqldf("SELECT CustomerId, Product, State  
FROM df1  
JOIN df2 USING(CustomerID)")
```

2

Left / Right join

```
## left join (substitute 'right' for right join)  
df4 <- sqldf("SELECT CustomerId, Product, State  
FROM df1  
LEFT JOIN df2 USING(CustomerID)")
```




Data Preparation

masem research institute GmbH
Bereich masem training

Unter den Eichen 5, Gebäude G,
D-65195 Wiesbaden
Registergericht Wiesbaden HRB 25700
Geschäftsführer: Dr. Guido Möser

Wissenschaftlicher Beirat:
Prof. Dr. Peter Schmidt

www.masem-training.com

Data Preparation

Built-in Functions

- Built-in functions are fast and always available!
- Use **indexing** to (1) change existing variables or (2) create new ones
- **Avoid** using **for()**-loops in Data Preparation
- Vector and matrix-style operations are faster, especially **apply()**-family in combination with functions (built-in or user-defined ones)

External Packages

- Use packages, e.g. Hmisc, e1071 or car to do some recoding
- How to find out which packages to use? **Stack Overflow** and **R Mailing Lists** are first-choice-sources! Also, use Google
- **CRAN Task Views** are another very helpful source; Task Views bundle topics and are maintained by experts; Important packages are referenced there: <https://cran.r-project.org/web/views/>

Missing Values

Missing values are denoted by NA or NaN for undefined mathematical operations.

- `is.na()` is used to test objects if they are NA
- `is.nan()` is used to test for NaN
- NA values have a class also, so there are integer NA, character NA, etc.
- A NaN value is also NA but the converse is not true

Missing Values

```
> x <- c(1, 2, NA, 10, 3)
> is.na(x)
[1] FALSE FALSE TRUE FALSE FALSE
> is.nan(x)
[1] FALSE FALSE FALSE FALSE FALSE
> x <- c(1, 2, NaN, NA, 4)
> is.na(x)
[1] FALSE FALSE TRUE TRUE FALSE
> is.nan(x)
[1] FALSE FALSE TRUE FALSE FALSE
```

Replace Missing Values (NA)

Missing Values (NA) can be replaced by indexing methods

```
# Generate a data.frame with missing values
m <- matrix(sample(c(NA, 1:10), 100, replace = TRUE), 10)
(d <- as.data.frame(m))
d[is.na(d)] <- 0 # Replace NA by 0
d
```

Be careful, this will ultimately change the existing variable! Instead, copy into a new variable!

```
# Generate a vector with missing values
v1 <- c(3, 2, 5, 4, NA, 7, NA, 8, 4, 3, 2)
# Copy v1 into v2
v2 <- v1
# Replace missing values by mean (mean substitution)
v2[is.na(v2)] <- mean(v2, na.rm = TRUE)
v2
```

Replace Missing Values (NA)

Missing Values (NA) can be replaced using `replace()`-function

```
# Generate a vector with missing values
v1 <- c(3, 2, 5, 4, NA, 7, NA, 8, 4, 3, 2)
# No need to copy first!
v2 <- replace(v1, which(is.na(v1)), mean(v1, na.rm = TRUE))
v2
```

Use norm-package (Schafer & Graham, 2002): <https://cran.r-project.org/web/packages/norm/norm.pdf>

```
# norm package
#install.packages("norm")
library(norm)
data(mdata)
s <- prelim.norm(mdata) #do preliminary manipulations
thetahat <- em.norm(s) #compute mle
getparam.norm(s,thetahat,corr=TRUE)$r #look at estimated correlations
```

Recoding Variables

Use built-in functions (indexing) to recode variables

```
## Recode into a new variable
# Generate an age vector
age <- round(runif(n = 100, min = 15, max = 100))
age
# Create 3 age categories
age.cat <- vector(mode = "numeric", length = 100)
age.cat[age > 75] <- "Elder"
age.cat[age > 45 & age <= 75] <- "Middle Aged"
age.cat[age <= 45] <- "Young"
age.cat
# Convert into unordered factor
age.cat <- as.factor(age.cat)
age.cat
# Frequency table
table(age.cat)
```

Recoding Variables: ifelse

Use ifelse (R's control structures) to create 2 age categories

```
## Recode into a new variable using ifelse()
age.cat2 <- vector(mode = "numeric", length = 100)
age.cat2 <- ifelse(age > 70, c("older"), c("younger"))
age.cat2
```


Recoding Variables: package car function: recode()

Use recode() function in package car

```
## 3rd option: recode()-function in package car
library(car) # should be available with main installation
# Generate a new vector
attitude <- c("agree", "neutral", "disagree", "agree", "disagree")
attitude.ger <- recode(attitude, "'agree' = 'stimme zu'; 'neutral'='teils teils'; 'disagree' = 'stimme
nicht zu'")
attitude.ger
```

Cut continuous variables into categorical variables

Use `recode()` function in package `car`

```
## Cut continuous variables into categorical variables using car-package  
# Generate an age vector  
age <- round(runif(n = 100, min = 15, max = 100))  
age  
recode(age, "0:40 = 1; 41:75 = 2; else = 3")
```

Turn ordered value ranges into factor levels using cut()

Use cut() function in package car

```
## Turn ordered value ranges into factor levels using cut()
# Generate an age vector
age <- round(runif(n = 100, min = 15, max = 100))
age
# use cut
cut(age, breaks=c(0,41,75, Inf), labels = c("young","middle aged", "old"))
```

Creating new variables

Use indexing in a data.frame

```
# 1st option: use indexing
# create a new data.frame
age <- round(runif(n = 100, min = 15, max = 100))
age.df <- as.data.frame(matrix(age, ncol = 2), dimnames = c(v1, v2))
age.df
# mean age and sum of age
age.df$mean.age <- (age.df$V1 + age.df$V2)/2
age.df$sum.age <- age.df$V1 + age.df$V2
age.df
```

Creating new variables

Use transform() in a data.frame

```
# 2nd option: use transform()-function
# create a new data.frame
age <- round(runif(n = 100, min = 15, max = 100))
age.df <- as.data.frame(matrix(age, ncol = 2), dimnames = c(v1, v2))
age.df
# mean age and sum of age
age.df <- transform(age.df, sum = V1 + V2, mean = (V1 + V2)/2)
age.df
```

Renaming variables

Use names()-function to rename variables

```
## Renameing variables
# create a new data.frame
age <- round(runif(n = 100, min = 15, max = 100))
age.df <- as.data.frame(matrix(age, ncol = 2), dimnames = c(v1, v2))
# print names
names(age.df)
# Rename V1 into age of main building and V2 into vehicle hall
names(age.df) <- c("main building", "vehicle hall")
head(age.df)
age.df$'main bildung' # use single quotes due to spaces in names
```

Renaming variables

Interactively using fix()

```
# 2nd option: interactively using fix()-function
# create a new data.frame
age <- round(runif(n = 100, min = 15, max = 100))
age.df <- as.data.frame(matrix(age, ncol = 2), dimnames = c(v1, v2))
# use fix()
fix(age.df) # be careful using RStudio!
age.df
```

Do not use fix() in scripts or functions, which should automatically run!

Renaming variables

Use reshape library

```
# 3rd option: use reshape-library
#install.packages("reshape")
library(reshape)
# create a new data.frame
age <- round(runif(n = 100, min = 15, max = 100))
age.df <- as.data.frame(matrix(age, ncol = 2), dimnames = c(v1, v2))
age.df <- rename(age.df, c(V1 = "main building", V2 = "vehicle hall"))
head(age.df )
```


Basic Statistical Analysis in R

masem research institute GmbH
Bereich masem training

Unter den Eichen 5, Gebäude G,
D-65195 Wiesbaden
Registergericht Wiesbaden HRB 25700
Geschäftsführer: Dr. Guido Möser

Wissenschaftlicher Beirat:
Prof. Dr. Peter Schmidt

www.masem-training.com

Object Summaries: summary()

```
summary(object, ...)
```

```
summary()
```

- **summary** is a generic function used to produce result summaries of the results of various model fitting functions.

```
## subset  
data(cars)  
## Linear regression  
lm1 <- lm(dist ~ speed, data = cars)  
## object lm1 summaries  
summary(lm1)
```

Compactly Display the Structure of an Arbitrary R Object: str()

```
str(object, ...)
```

```
str()
```

- Compactly display the internal structure of an R object, a diagnostic function;
- ...

```
## subset  
data(cars)  
## structure of dataset  
str(cars)  
## Linear regression  
lm1 <- lm(dist ~ speed, data = cars)  
## structure of object lm1  
summary(lm1)
```

Arithmetic Mean

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

```
mean()
```

- Generic function for the (trimmed) arithmetic mean. ;
- **x**: An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for trim = 0, only.
- **trim**: the fraction (0 to 0.5) of observations to be trimmed from each end of x before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
- **na.rm**: a logical value indicating whether NA values should be stripped before the computation proceeds

```
## subset  
data(cars)  
## mean of first variable  
mean(cars$dist)  
## Trimmed mean  
mean(cars$dist, trim = 0.05)
```

Standard Deviation

```
sd(x, na.rm = FALSE)
```

```
sd()
```

- This function computes the standard deviation of the values in `x`.
- **x**: An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
- **na.rm**: a logical value indicating whether NA values should be stripped before the computation proceeds

```
## Calculate Standard Deviation  
sd(1:2) ^ 2  
sd(iris$Sepal.Length)  
with(iris, sd(Sepal.Length))
```

Correlation, Variance and Covariance (Matrices)

```
var(x, y = NULL, na.rm = FALSE, use)
cov(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))
cor(x, y = NULL, use = "everything",
    method = c("pearson", "kendall", "spearman"))
cov2cor(V)
```

**var(),
cov(), cor()**

- **var**, **cov** and **cor** compute the variance of x and the covariance or correlation of x and y if these are vectors.
- If x and y are matrices then the covariances (or correlations) between the columns of x and the columns of y are computed.
- **cov2cor** scales a covariance matrix into the corresponding correlation matrix efficiently.

```
## Calculate variance, covariance, correlation
var(1:10) # 9.166667
var(1:5, 1:5) # 2.5
## Two simple vectors
cor(1:10, 2:11) # == 1
## Correlation Matrix of Multivariate sample:
(CI <- cor(longley))
```

Mean Absolute Deviation

```
mad(x, center = median(x), constant = 1.4826, na.rm = FALSE,  
    low = FALSE, high = FALSE)
```

mad()

- Compute the median absolute deviation, i.e., the (lo-/hi-) median of the absolute deviations from the median, and (by default) adjust by a factor for asymptotically normal consistency.
- **center** Optionally, the centre: defaults to the median.
- The default constant (1.4826 , $1 / \text{qnorm}(3/4)$) ensures consistency, i.e. $E[\text{mad}(x)] = \sigma$

```
## mean absolute deviation  
mad(c(1:9))  
# robust for outliers  
print(mad(c(1:9), constant = 1)) == mad(c(1:8, 100), constant = 1) # = 2 ; TRUE
```


Maxima and Minima

```
max(..., na.rm = FALSE)  
min(..., na.rm = FALSE)
```

```
max()  
min()
```

- Returns the (parallel) maxima and minima of the input values.
- ...: Numeric or character arguments
- **na.rm**: a logical value indicating whether NA values should be stripped before the computation proceeds

```
## min and max  
min(5:1, pi) #-> one number  
max(5:1, pi) #-> one number
```

Median Value

```
median(x, na.rm = FALSE)
```

```
median()
```

- Compute the sample median.
- **x**: an object for which a method has been defined, or a numeric vector containing the values whose median is to be computed.
- **na.rm**: a logical value indicating whether NA values should be stripped before the computation proceeds

```
## median  
median(1:4)          # = 2.5 [even number]  
median(c(1:3, 100, 1000)) # = 3 [odd, robust]
```

Mode Value

Mode(x)

Mode()

- Important: There is no built-in generic function in R to calculate sample mode
- Below a solution, which works for both numeric & character/factor data

```
## Calculate mode
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}

## Works also for multiple modes
Mode <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
```

Basic descriptive statistics (*not only*) useful for psychometrics {psych}

```
library(psych)
describe(x, na.rm = TRUE, interp=FALSE,skew = TRUE, ranges = TRUE,trim=.1,
type=3,check=TRUE,fast=NULL,quant=NULL,IQR=FALSE)
describeData(x,head=4,tail=4)
```

describe()

- Compactly display the internal structure of an R object, a diagnostic function;
- There are many summary statistics available in R; this function provides the ones most useful for scale construction and item analysis in classic psychometrics.
- **Range** is most useful for the first pass in a data set, to check for coding errors.
- **x**: A data frame or matrix
- **describeData** allows a quick pass after reading the dataset
- ...

```
data(sat.act)
describe(sat.act)
describe(sat.act,skew=FALSE)
describe(sat.act,IQR=TRUE) #show the interquartile Range
describe(sat.act,quant=c(.1,.25,.5,.75,.90) ) #find the 10th, 25th, 50th, 75th and 90th percentiles
describeData(sat.act) #the fast version
```

Basic summary statistics by group {psych}

```
library(psych)
describeBy(x, group=NULL,mat=FALSE,type=3,digits=15,...)
describe.by(x, group=NULL,mat=FALSE,type=3,...) # deprecated
```

describeBy()

- Report basic summary statistics by a grouping variable.
- Useful if the grouping variable is some experimental variable and data are to be aggregated for plotting.
- **x**: a data.frame or matrix. See note for statsBy.
- **group**: a grouping variable or a list of grouping variables
- **digits**: When giving matrix output, how many digits should be reported?
- ...

```
data(sat.act)
describeBy(sat.act,sat.act$gender) #just one grouping variable
#describeBy(sat.act,list(sat.act$gender,sat.act$education)) #two grouping variables
```

Concise Statistical Description of a Vector, Matrix, Data Frame, or Formula {Hmisc}

```
library(Hmisc)  
describe(x, descript, exclude.missing=TRUE,  
         digits=4, ...)
```

describe()

- **describe** is especially useful for describing data frames created by `*.get`, as labels, formats, value labels, and (in the case of `sas.get`) frequencies of special missing values are printed.
- **x**: a data frame, ...
- **digits**: number of significant digits to print
- ...

```
dfr <- data.frame(x=rnorm(400),y=sample(c('male','female'),400,TRUE))  
describe(dfr)
```

Apply Functions Over Array Margins

```
apply(X, MARGIN, FUN, ...)
```

```
apply()
```

apply returns a vector or array or list of values obtained by applying a function to margins of an array or matrix

- **X**: an array, including matrix and data.frame.
- **MARGIN**: a vector giving the subscripts which the function will be applied over.
 - E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns.
 - Where X has named dimnames, it can be a character vector selecting dimension names.
- **FUN**: the function to be applied
 - In the case of functions like +, %*%, etc., the function name must be backquoted or quoted.
- **...**: optional arguments to FUN.

```
## Compute row and column sums for a matrix:
```

```
x <- cbind(x1 = 3, x2 = c(4:1, 2:5))
```

```
dimnames(x)[[1]] <- letters[1:8]
```

```
apply(x, 2, mean, trim = .2)
```

```
col.sums <- apply(x, 2, sum)
```

```
row.sums <- apply(x, 1, sum)
```

```
rbind(cbind(x, Rtot = row.sums), Ctot = c(col.sums, sum(col.sums)))
```

```
## Sort the columns of a matrix
```

```
apply(x, 2, sort)
```

Basic Statistical Analysis in R

FREQUENCY TABLES

Tabulating Data

Function	Description
<code>table(var1, var2, ..., varN)</code>	Creates an N-way contingency table of counts from categorical variables
<code>xtabs(formula, data)</code>	Creates an N-way contingency table based on a formula and data
<code>prop.table(table, margins)</code>	Expresses entries in the table as fractions of the marginal table defined by margins
<code>margin.table(table, margins)</code>	Computes the sum of table entries for a marginal table defined by margins
<code>addmargins(table, margins)</code>	Adds sums to the margins of a table
<code>ftable(table)</code>	Creates a flat contingency table
<code>gmodels:CrossTable</code>	SAS [®] and SPSS [®] style cross-tables in R

Cross Tabulation and Table Creation: table()-function

```
table(...,  
  exclude = if (useNA == "no") c(NA, NaN),  
  useNA = c("no", "ifany", "always"),  
  dnn = list.names(...), deparse.level = 1)
```

table()

table uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

- **...**: one or more objects which can be interpreted as factors (including character strings), or a list (or data frame) whose components can be so interpreted. (For `as.table`, arguments passed to specific methods; for `as.data.frame`, unused.)
- **exclude**: levels to remove for all factors in If set to `NULL`, it implies `useNA = "always"`. See 'Details' for its interpretation for non-factor arguments.
- **useNA**: whether to include NA values in the table. See 'Details'. Can be abbreviated.

```
## Working Horse is table()-function  
# Use subset-function to extract gear-variable only  
head(subset(mtcars, select = 'gear'))  
# Possible values of gear  
factor(mtcars$gear)  
# Frequency table: How many cars are there for each number of forward gears?  
gear.table <- table(mtcars$gear)  
gear.table
```

plyr-Package: count()-function

```
count(df, vars = NULL, wt_var = NULL)
```

```
count()
```

Count the number of occurrences: Equivalent to `as.data.frame(table(x))`, but does not include combinations with zero counts.

- **Main advantage: result is a data.frame (which can easily be exported)**
- **df:** data frame to be processed
- **vars:** variables to count unique values of
- **wt_var:** optional variable to weight by - if this is non-NULL, count will sum up the value of this variable for each combination of id variables.

```
## Use count-function in plyr-package
#install.packages("plyr", dependencies = TRUE)
library(plyr)
# Count produces a cross table
count(mtcars, 'gear')
# Main advantage: result is data frame
gear.count <- count(mtcars, 'gear')
class(gear.count)
```

(Cumulative) relative frequencies: prop.table()

```
prop.table(x, margin = NULL)
```

```
prop.table()
```

Express Table Entries as Fraction of Marginal Table

This is really `sweep(x, margin, margin.table(x, margin), "/")` for newbies, except that if `margin` has length zero, then one gets $x/\text{sum}(x)$.

- **x:** table
- **margin:** index, or vector of indices to generate margin for

cumsum() function is used to calculate cumulative relative frequencies!

```
#####  
## Relative Frequencies  
prop.table(gear.table)*100  
# Change number of digits  
options(digits = 3)  
prop.table(gear.table)  
prop.table(gear.table)*100  
# Cumulative frequencies  
cumsum(prop.table(gear.table))  
cumsum(prop.table(gear.table)*100)
```

Contingency tables for two or more variables

```
table(...,  
  exclude = if (useNA == "no") c(NA, NaN),  
  useNA = c("no", "ifany", "always"),  
  dnn = list.names(...), deparse.level = 1)
```

table()

table can be used to generate crosstables

- `table(<rows>, <columns>)`

```
## Contingency tables for two or more variables  
str(mtcars) # Use help() to get further information  
# Gear by Transmission (0 = automatic, 1 = manual)  
ctab.gear.am <- table(mtcars$am, mtcars$gear)  
ctab.gear.am  
barplot(ctab.gear.am, beside=TRUE, legend.text=rownames(ctab.gear.am), ylab="absolute frequency")  
# Chi2-Test  
summary(ctab.gear.am)
```

Cross Tabulation: xtabs()-function

```
xtabs(formula = ~., data = parent.frame(), subset, sparse = FALSE,  
      na.action, exclude = c(NA, NaN), drop.unused.levels = FALSE)
```

xtabs()

Create a contingency table (optionally a sparse matrix) from cross-classifying factors, usually contained in a data frame, using a formula interface.

- **formula:** a formula object with the cross-classifying variables (separated by +) on the right hand side (or an object which can be coerced to a formula).
- **data:** an optional matrix or data frame
- **subset:** an optional vector specifying a subset of observations to be used.
- **na.action:** a function which indicates what should happen when the data contain NAs.
- **exclude:** a vector of values to be excluded when forming the set of levels of the classifying factors.
- **drop.unused.levels:** a logical indicating whether to drop unused levels in the classifying factors.

```
#####  
## xtabs: Use formula interface  
xtabs(~ am + gear, data = mtcars)
```

R: gmodels::CrossTable

```
CrossTable(x, y, digits=3, max.width = 5, expected=FALSE, prop.r=TRUE, prop.c=TRUE,  
  prop.t=TRUE, prop.chisq=TRUE, chisq = FALSE, fisher=FALSE, mcnemar=FALSE,  
  resid=FALSE, sresid=FALSE, asresid=FALSE,  
  missing.include=FALSE,  
  format=c("SAS","SPSS"), dnn = NULL, ...)
```

CrossTable()

An implementation of a cross-tabulation function with output similar to **S-Plus crosstabs()** and **SAS Proc Freq (or SPSS format)** with Chi-square, Fisher and McNemar tests of the independence of all table factors.

- **x**: A vector or a matrix. If y is specified, x must be a vector
- **y**: A vector in a matrix or a dataframe
- **digits**: Number of digits after the decimal point for cell proportions
- **max.width**: In the case of a 1 x n table, the default will be to print the output horizontally. If the number of columns exceeds max.width, the table will be wrapped for each successive increment of max.width columns. If you want a single column vertical table, set max.width to 1
- **expected**: If TRUE, chisq will be set to TRUE and expected cell counts from the Chi-Square will be included
- **prop.r**: If TRUE, row proportions will be included
- **prop.c**: If TRUE, column proportions will be included
- **prop.t**: If TRUE, table proportions will be included
- **prop.chisq**: If TRUE, chi-square contribution of each cell will be included

R: gmodels::CrossTable

```
CrossTable(x, y, digits=3, max.width = 5, expected=FALSE, prop.r=TRUE, prop.c=TRUE,  
  prop.t=TRUE, prop.chisq=TRUE, chisq = FALSE, fisher=FALSE, mcnemar=FALSE,  
  resid=FALSE, sresid=FALSE, asresid=FALSE,  
  missing.include=FALSE,  
  format=c("SAS","SPSS"), dnn = NULL, ...)
```

CrossTable()

An implementation of a cross-tabulation function with output similar to **S-Plus crosstabs()** and **SAS Proc Freq (or SPSS format)** with Chi-square, Fisher and McNemar tests of the independence of all table factors.

- **chisq**: If TRUE, the results of a chi-square test will be included
- **fisher**: If TRUE, the results of a Fisher Exact test will be included
- **mcnemar**: If TRUE, the results of a McNemar test will be included
- **resid**: If TRUE, residual (Pearson) will be included
- **sresid**: If TRUE, standardized residual will be included
- **asresid**: If TRUE, adjusted standardized residual will be included
- **missing.include**: If TRUE, then remove any unused factor levels
- **format**: Either SAS (default) or SPSS, depending on the type of output desired.
- **dnn**: the names to be given to the dimensions in the result (the dimnames names).
- ...: optional arguments

R: gmodels::CrossTable

```
CrossTable(x, y, digits=3, max.width = 5, expected=FALSE, prop.r=TRUE, prop.c=TRUE,  
  prop.t=TRUE, prop.chisq=TRUE, chisq = FALSE, fisher=FALSE, mcnemar=FALSE,  
  resid=FALSE, sresid=FALSE, asresid=FALSE,  
  missing.include=FALSE,  
  format=c("SAS","SPSS"), dnn = NULL, ...)
```

CrossTable()

An implementation of a cross-tabulation function with output similar to **S-Plus crosstabs()** and **SAS Proc Freq (or SPSS format)** with Chi-square, Fisher and McNemar tests of the independence of all table factors.

- ...

```
#####  
## SAS and SPSS-Style Cross-Tables  
#install.packages("gmodels")  
library(gmodels)  
data(infert, package = "datasets")  
?infert  
data(infert)  
View(infert)  
# Examples:  
CrossTable(infert$education, infert$induced, expected = TRUE)  
CrossTable(infert$education, infert$induced, expected = TRUE, format="SAS")  
CrossTable(infert$education, infert$induced, expected = TRUE, format="SPSS")
```

Marginal Sums and Means

`apply()`, `colMeans()` and `addmargins()`

```
ctab.gear.am
# Row Sum
apply(ctab.gear.am, MARGIN = 1, FUN = sum)
# Column Sum
apply(ctab.gear.am, MARGIN = 2, FUN = sum)
# Alternative: colSums colMeans
colSums(ctab.gear.am)
colMeans(ctab.gear.am)

## Add Margins
addmargins(ctab.gear.am, c(1,2), FUN = sum)
## Add Margins: Means
addmargins(ctab.gear.am, c(1,2), FUN = mean)
```

Relative and conditional frequencies

`prop.table()`

```
#####  
## Relative Frequencies and conditional frequencies  
# Check if table is still available  
ctab.gear.am  
## Relative frequencies  
# Total percentages  
relFreq <- prop.table(ctab.gear.am)  
relFreq  
## Conditional relative frequencies  
# row based  
prop.table(ctab.gear.am, margin = 1)  
# column based  
prop.table(ctab.gear.am, margin = 2)
```

Recovering the original data from contingency tables

Package DescTools::Untable()

```
#####  
## Recovering the original data from contingency tables  
#install.packages("DescTools")  
library(DescTools)  
Untable(ctab.gear.am)
```

Percentile rank

ecdf()-function

```
#####  
## Percentile ranks  
# Generate dataset  
(vec <- round(rnorm(10), 2))  
# Empirical Cumulative Distribution Function  
Fn <- ecdf(vec)  
Fn(vec)  
100 * Fn(0.1)  
Fn(sort(vec))  
# Plot empirical cumulative distribution function  
plot(Fn, main="cumulative frequencies")
```

Basic Statistical Analysis in R

STATISTICAL TESTS

Pearson's Chi-squared Test for Count Data

```
chisq.test(x, y = NULL, correct = TRUE,  
           p = rep(1/length(x), length(x)), rescale.p = FALSE,  
           simulate.p.value = FALSE, B = 2000)
```

chisq.test()

chisq.test performs chi-squared contingency table tests and goodness-of-fit tests.

- **x**: a numeric vector or matrix. x and y can also both be factors.
- **y**: a numeric vector; ignored if x is a matrix. If x is a factor, y should be a factor of the same length.
- **correct**: a logical indicating whether to apply continuity correction when computing the test statistic for 2 by 2 tables: one half is subtracted from all $|O - E|$ differences; however, the correction will not be bigger than the differences themselves. No correction is done if `simulate.p.value = TRUE`.

```
## From Agresti(2007) p.39  
M <- as.table(rbind(c(762, 327, 468), c(484, 239, 477)))  
dimnames(M) <- list(gender = c("F", "M"),  
                    party = c("Democrat", "Independent", "Republican"))  
(Xsq <- chisq.test(M)) # Prints test summary  
str(Xsq)  
Xsq$observed # observed counts (same as M)  
Xsq$expected # expected counts under the null  
Xsq$residuals # Pearson residuals  
Xsq$stdres   # standardized residuals
```

Student's -Test

```
t.test(x, y = NULL,  
       alternative = c("two.sided", "less", "greater"),  
       mu = 0, paired = FALSE, var.equal = FALSE,  
       conf.level = 0.95, ...)
```

t.test()

Performs one and two sample t-tests on vectors of data.

- **x:** a (non-empty) numeric vector of data values.
- **y:** an optional (non-empty) numeric vector of data values.
- **alternative:** a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
- **mu:** a number indicating the true value of the mean (or difference in means if you are performing a two sample test).
- **paired:** a logical indicating whether you want a paired t-test.
- **var.equal:** a logical variable indicating whether to treat the two variances as being equal. If TRUE then the pooled variance is used to estimate the variance otherwise the Welch (or Satterthwaite) approximation to the degrees of freedom is used.

```
t.test(1:10, y = c(7:20))    # P = .00001855  
t.test(1:10, y = c(7:20, 200)) # P = .1245  -- NOT significant anymore  
## Classical example: Student's sleep data  
plot(extra ~ group, data = sleep)  
## Traditional interface  
with(sleep, t.test(extra[group == 1], extra[group == 2]))  
## Formula interface  
t.test(extra ~ group, data = sleep)
```


F Test to Compare Two Variances

```
var.test(x, y, ratio = 1,  
         alternative = c("two.sided", "less", "greater"),  
         conf.level = 0.95, ...)
```

```
var.test()
```

Performs an F test to compare the variances of two samples from normal populations.

- **x, y:** numeric vectors of data values, or fitted linear model objects (inheriting from class "lm").
- **ratio:** the hypothesized ratio of the population variances of x and y.
- **alternative:** a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
- **conf.level:** confidence level for the returned confidence interval.
- **formula:** a formula of the form lhs ~ rhs where lhs is a numeric variable giving the data values and rhs a factor with two levels giving the corresponding groups.
- **data:** an optional matrix or data frame (or similar: see model.frame) containing the variables in the formula formula. By default the variables are taken from environment(formula).
- **subset:** an optional vector specifying a subset of observations to be used.
- **na.action:** a function which indicates what should happen when the data contain NAs. Defaults to getOption("na.action").

```
x <- rnorm(50, mean = 0, sd = 2)  
y <- rnorm(30, mean = 1, sd = 1)  
var.test(x, y)           # Do x and y have the same variance?  
var.test(lm(x ~ 1), lm(y ~ 1)) # The same.
```

Test for Association/Correlation Between Paired Samples

```
cor.test(x, y,  
         alternative = c("two.sided", "less", "greater"),  
         method = c("pearson", "kendall", "spearman"),  
         exact = NULL, conf.level = 0.95, continuity = FALSE, ...)
```

cor.test()

Test for association between paired samples, using one of Pearson's product moment correlation coefficient, Kendall's tau or Spearman's rho.

```
## Hollander & Wolfe (1973), p. 187f.  
## Assessment of tuna quality.  
x <- c(44.4, 45.9, 41.9, 53.3, 44.7, 44.1, 50.7, 45.2, 60.1)  
y <- c( 2.6,  3.1,  2.5,  5.0,  3.6,  4.0,  5.2,  2.8,  3.8)  
## The alternative hypothesis of interest is that the  
## Hunter L value is positively associated with the panel score.  
cor.test(x, y, method = "kendall", alternative = "greater")  
## => p=0.05972  
cor.test(x, y, method = "kendall", alternative = "greater",  
         exact = FALSE) # using large sample approximation  
## => p=0.04765  
## Compare this to  
cor.test(x, y, method = "spearman", alternative = "g")  
cor.test(x, y, alternative = "g")
```

Shapiro Test: Testing for Normality

```
shapiro.test(x)
```

```
shapiro.test()
```

Performs the Shapiro-Wilk test of normality.

- **x**: a numeric vector of data values. Missing values are allowed, but the number of non-missing values must be between 3 and 5000.

How to interpret?

If p -Value is less than the significance level (0.05), the null-hypothesis that it is normally distributed can be rejected.

```
# Example: Test a normal distribution  
normaly_disb <- rnorm(100, mean=5, sd=1) # generate a normal distribution  
shapiro.test(normaly_disb)
```

Wilcoxon Rank Sum and Signed Rank Tests

```
wilcox.test(x, y = NULL,  
            alternative = c("two.sided", "less", "greater"),  
            mu = 0, paired = FALSE, exact = NULL, correct = TRUE,  
            conf.int = FALSE, conf.level = 0.95, ...)
```

wilcox.test()

Performs one- and two-sample Wilcoxon tests on vectors of data; the latter is also known as ‘Mann-Whitney’ test.

- **x**: numeric vector of data values. Non-finite (e.g., infinite or missing) values will be omitted.
- **y**: an optional numeric vector of data values: as with x non-finite values will be omitted.
- **alternative**: a character string specifying the alternative hypothesis, must be one of "two.sided" (default), "greater" or "less". You can specify just the initial letter.
- **mu**: a number specifying an optional parameter used to form the null hypothesis. See ‘Details’.
- **paired**: a logical indicating whether you want a paired test.
- **exact**: a logical indicating whether an exact p-value should be computed.
- ...

```
x <- c(1.83, 0.50, 1.62, 2.48, 1.68, 1.88, 1.55, 3.06, 1.30)  
y <- c(0.878, 0.647, 0.598, 2.05, 1.06, 1.29, 1.06, 3.14, 1.29)  
wilcox.test(x, y, paired = TRUE, alternative = "greater")  
wilcox.test(y - x, alternative = "less") # The same.  
wilcox.test(y - x, alternative = "less", exact = FALSE, correct = FALSE) # H&W large sample approximation
```

Kolmogorov-Smirnov Tests

```
ks.test(x, y, ...,  
        alternative = c("two.sided", "less", "greater"),  
        exact = NULL)
```

```
ks.test()
```

Perform a one- or two-sample Kolmogorov-Smirnov test.

- **x**: a numeric vector of data values.
- **y**: either a numeric vector of data values, or a character string naming a cumulative distribution function or an actual cumulative distribution function such as `pnorm`. Only continuous CDFs are valid.
- **...**: parameters of the distribution specified (as a character string) by `y`.
- **alternative**: indicates the alternative hypothesis and must be one of "two.sided" (default), "less", or "greater". You can specify just the initial letter of the value, but the argument name must be given in full. See 'Details' for the meanings of the possible values.
- **exact**: NULL or a logical indicating whether an exact p-value should be computed. See 'Details' for the meaning of NULL. Not available in the two-sample case for a one-sided test or if ties are present....

```
x <- rnorm(50)  
y <- runif(30)  
# Do x and y come from the same distribution?  
ks.test(x, y)  
# Does x come from a shifted gamma distribution with shape 3 and rate 2?  
ks.test(x+2, "pgamma", 3, 2) # two-sided, exact  
ks.test(x+2, "pgamma", 3, 2, exact = FALSE)  
ks.test(x+2, "pgamma", 3, 2, alternative = "gr")
```

Fisher's Exact Test for Count Data

```
fisher.test(x, y = NULL, workspace = 200000, hybrid = FALSE,  
            control = list(), or = 1, alternative = "two.sided",  
            conf.int = TRUE, conf.level = 0.95,  
            simulate.p.value = FALSE, B = 2000)
```

fisher.test()

Performs Fisher's exact test for testing the null of independence of rows and columns in a contingency table with fixed marginals.

- **x**: either a two-dimensional contingency table in matrix form, or a factor object.
- **y**: a factor object; ignored if x is a matrix.
- **workspace**: an integer specifying the size of the workspace used in the network algorithm. In units of 4 bytes. Only used for non-simulated p-values larger than 2 by 2 tables.
- **hybrid**: a logical. Only used for larger than 2 by 2 tables, in which cases it indicates whether the exact probabilities (default) or a hybrid approximation thereof should be computed.
- ...

```
## A r x c table Agresti (2002, p. 57) Job Satisfaction  
Job <- matrix(c(1,2,1,0, 3,3,6,1, 10,10,14,9, 6,7,12,11), 4, 4,  
             dimnames = list(income = c("< 15k", "15-25k", "25-40k", "> 40k"),  
                             satisfaction = c("VeryD", "LittleD", "ModerateS", "VeryS")))  
fisher.test(Job)  
fisher.test(Job, simulate.p.value = TRUE, B = 1e5)
```

Friedman Rank Sum Test

```
friedman.test(y, groups, blocks, ...)
```

```
friedman.test()
```

Performs a Friedman rank sum test with unreplicated blocked data.

- **y**: either a numeric vector of data values, or a data matrix.
- **groups**: a vector giving the group for the corresponding elements of y if this is a vector; ignored if y is a matrix. If not a factor object, it is coerced to one.
- **blocks**: a vector giving the block for the corresponding elements of y if this is a vector; ignored if y is a matrix. If not a factor object, it is coerced to one.
- **formula**: a formula of the form $a \sim b \mid c$, where a, b and c give the data values and corresponding groups and blocks, respectively.
- ...

```
## A r x c table Agresti (2002, p. 57) Job Satisfaction
Job <- matrix(c(1,2,1,0, 3,3,6,1, 10,10,14,9, 6,7,12,11), 4, 4,
dimnames = list(income = c("< 15k", "15-25k", "25-40k", "> 40k"),
satisfaction = c("VeryD", "LittleD", "ModerateS", "VeryS")))
fisher.test(Job)
fisher.test(Job, simulate.p.value = TRUE, B = 1e5)
```

Helpful Packages and References

1

lawstat package

Statistical tests widely utilized in biostatistics, public policy, and law. Along with the wellknown tests for equality of means and variances, randomness, measures of relative variability etc, the package contains new robust tests of symmetry, omnibus and directional tests of normality, and their graphical counterparts such as Robust QQ plot; a robust trend tests for variances etc. All implemented tests and methods are illustrated by simulations and real-life examples from legal statistics, economics, and biostatistics.

2

outliers package

A collection of some tests commonly used for identifying outliers.

3

idre

Choosing the correct statistical test in SAS, Stata, SPSS and R:
<https://stats.idre.ucla.edu/other/mult-pkg/whatstat/>

4

Prof. Kanji

100 statistical tests

Split-Apply-Combine approaches in R

Loops and Flow Control

Overview: Important members of apply-family

apply

Apply a function over the margins of an array

lapply

Loop over a list and evaluate a function on each element

sapply

Same as lapply but try to simplify the result

tapply

Apply a function over subsets of a vector

mapply

Multivariate version of lapply

split

An auxiliary function useful in conjunction with lapply

When to use which apply-function?

	array	data frame	list	nothing
array	apply	adply*	alply*	a_ply*
data frame	daply*	aggregate	by	d_ply*
list	sapply	ldply*	lapply	l_ply

* Package plyr by Hadley Wickham

Hinweis: tapply → Input: immer Vektor! (vs. aggregate: data.frame!)

Split > Apply > Combine Approach

apply loops

```
apply(X, MARGIN, FUN, ARGs)
```

```
apply()
```

- X: array, matrix or data.frame;
- MARGIN: 1 for rows, 2 for columns, c(1,2) for both;
- FUN: one or more functions, including user defined functions;
- ARGs: possible arguments for function

```
## Example for applying predefined mean function  
apply(iris[,1:3], 1, mean)  
# Calculate 10! Using apply  
apply(X = data.frame(c(1:10)), MARGIN = 2, FUN = sum)
```



apply loops: How it works

X

apply(X ,2, sum)

Dimension — 2 →

↓
1

-1.7189391	-1.0863995	1.0996117	-0.55559727	-0.1792310	-0.8088577
-2.2542126	-1.3201873	-2.0533779	1.29055209	0.3264156	0.5412132
1.9874737	0.6265486	-0.3684977	1.40028967	-0.7574303	-2.3241569
0.2140376	0.8850445	1.4782993	-1.28177703	-0.5015628	1.1537703
0.9637687	1.3191502	0.8000988	0.09345943	1.4535431	1.0935720

Result

sum

-0.8078717	0.4241565	0.9561342	0.9469269	0.3417346	-0.3444591
------------	-----------	-----------	-----------	-----------	------------

lapply()

```
lapply(X, FUN, ...)
```

```
lapply()
```

- X: array, matrix or data.frame;
- FUN: one or more functions, including user defined functions;
- ... further arguments, related to FUN (e.g., na.rm-status in mean-function)

INPUT: Used for objects like data-frames, lists or vectors

OUTPUT: List (the l in the function name)

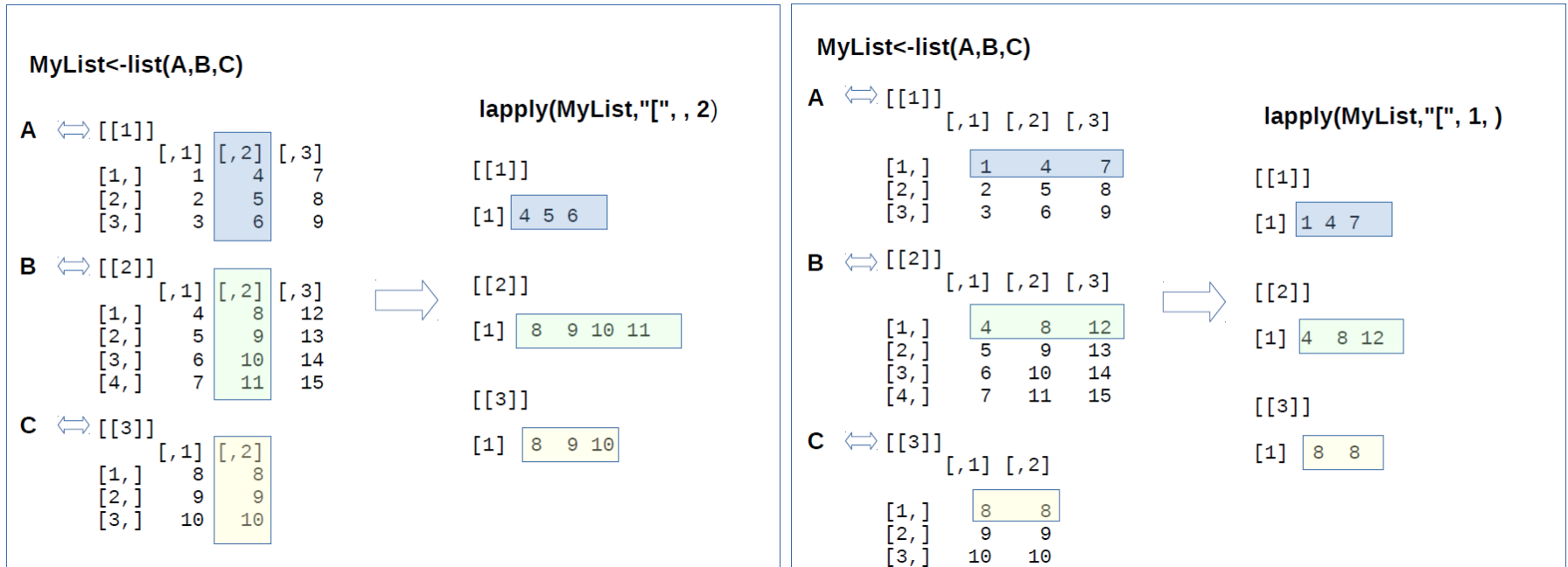
```
## Example for applying predefined mean function
```

```
apply(iris[,1:3], 1, mean)
```

```
# Calculate 10! Using apply
```

```
apply(X = data.frame(c(1:10)), MARGIN = 2, FUN = sum)
```

lapply() – How it works



apply() [Wrapper function for lapply]

```
apply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

apply()

apply works as lapply, but tries to simplify the output to the most elementary data structure that is possible.

- X: array, matrix or data.frame;
- FUN: one or more functions, including user defined functions;
- ... further arguments, related to FUN (e.g., na.rm-status in mean-function)

INPUT: Used for objects like data-frames, lists or vectors

OUTPUT: List (the l in the function name)

```
## Example for applying predefined mean function
```

```
apply(iris[,1:3], 1, mean)
```

```
# Calculate 10! Using apply
```

```
apply(X = data.frame(c(1:10)), MARGIN = 2, FUN = sum)
```

tapply

```
tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)
```

```
tapply()
```

Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

- X: an atomic object, typically a vector
- INDEX: list or one or more factors
- FUN: one or more functions, including user defined functions;
- ... further arguments, related to FUN (e.g., na.rm-status in mean-function)

```
## Example for applying predefined mean function  
data(iris)  
# tapply function  
tapply(iris[, 1], INDEX = iris$Species ,FUN=mean)
```

aggregate

```
aggregate(x, by, FUN, ..., simplify = TRUE, drop = TRUE)
```

```
aggregate()
```

Splits the data into subsets, computes summary statistics for each, and returns the result in a convenient form.

- x: an R object
- By: a list of grouping elements
- FUN: one or more functions, including user defined functions;
- ... further arguments, related to FUN (e.g., na.rm-status in mean-function)

```
## Example for aggregate
```

```
aggdata <- aggregate(mtcars, by=list(mtcars$cyl,mtcars$vs), FUN=mean, na.rm=TRUE)  
aggdata
```

A short Introduction to PLYR-package by H. Wickham

The PLYR package is a tool for doing split-apply-combine (SAC) procedures.

- SQL analogy: GROUP BY statement
- PLYR adds very little new functionality to R. What it does do is take the process of SAC and make it cleaner, more tidy and easier.

Split data frame, apply function, and return results in a data frame: `ddply()`

```
ddply(.data, .variables, .fun = NULL, ..., .progress = "none",  
      .inform = FALSE, .drop = TRUE, .parallel = FALSE, .paropts = NULL)
```

ddply()

For each subset of a data frame, apply function then combine results into a data frame. To apply a function for each row, use `adply` with `.margins` set to 1.

- **.data**: data frame to be processed
- **.variables**: variables to split data frame by, as as.quoted variables, a formula or character vector
- **.fun**: function to apply to each piece
- **...**: other arguments passed on to `.fun`

```
## Example for ddply  
# iris-dataset  
data(iris)  
ddply(iris[,1:4], c("iris$Species"), function(df)mean(df[,1]))
```


Programming in R: Control Structures and Functions

Overview Control Structures

Control structures in R allow you to control the flow of execution of the program, depending on runtime conditions.

if, else

Testing a condition

for

Execute a loop a fixed number of times

while

Execute a loop while a condition is true

repeat

Execute an infinite loop

break

Break the execution of a loop

next

Skip an iteration of a loop

return

Exit a function

Most control structures are **not used in interactive sessions**, but rather when writing functions or longer expressions.

Programming in R

FLOW CONTROL

Control Structures

Function	
if()...else	Determine which set of expressions to run depending on whether or not a conditions is TRUE
ifelse()	Do something to each element in a data structure depending on whether or not a condition is TRUE for that particular element
switch()	Evaluate different expressions depending on a given value

See ?Control for the R documentation on control structures.

Control Flow: if

if

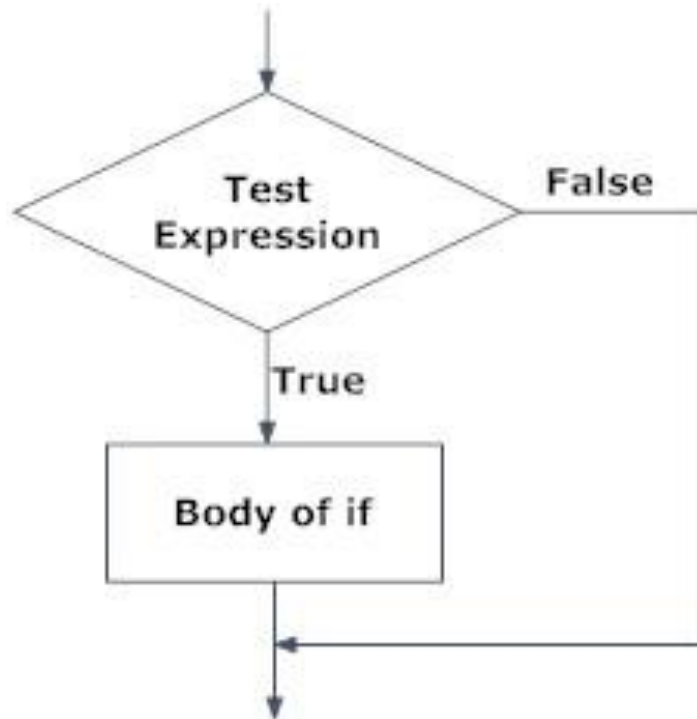


Fig: Operation of if statement

if... else

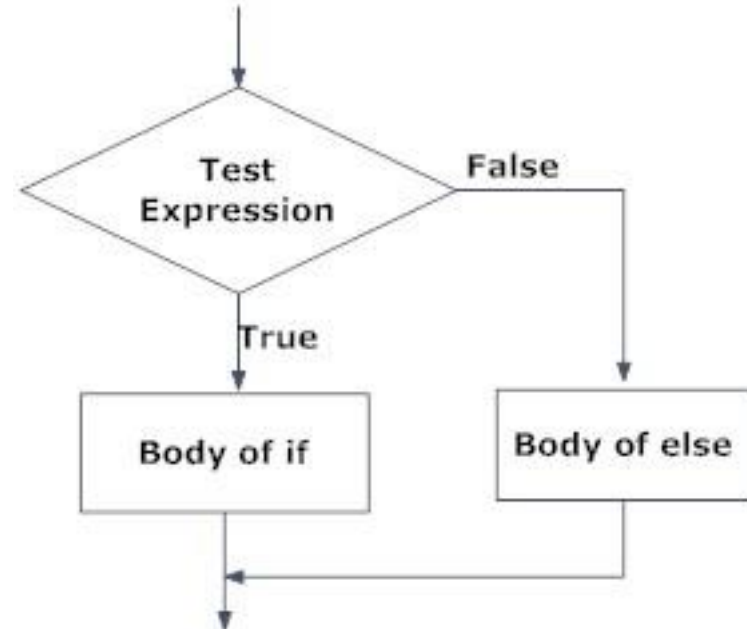


Fig: Operation of if...else statement

Beispiel

```
x <- 1
```

```
if (x == 1) {print("x ist 1!")}
```

```
x <- 2
```

```
if (x == 1) {print("x ist 1!")} else {print("Andere Zahl!")}
```

```
x <- 3
```

```
if (x == 1) {print("x ist 1!")} else if (x == 2) {print("x ist 2!")} else {print("Andere Zahl!")}
```

Control Flow: if

```
if(cond) expr
```

```
if()
```

- **cond** A length-one logical vector that is not NA. Conditions of length greater than one are accepted with a warning, but only the first element is used. Other types are coerced to logical if possible, ignoring any class.
- **expr, cons.expr, alt.expr**
An expression in a formal sense. This is either a simple expression or a so called compound expression, usually of the form { expr1 ; expr2 }.

```
x <- 5  
if(x > 0){  
  print("Positive number")  
}
```

Control Flow: if...else

```
if(cond) cons.expr else alt.expr
```

```
if...else()
```

- **cond** A length-one logical vector that is not NA. Conditions of length greater than one are accepted with a warning, but only the first element is used. Other types are coerced to logical if possible, ignoring any class.
- **expr, cons.expr, alt.expr**
An expression in a formal sense. This is either a simple expression or a so called compound expression, usually of the form { expr1 ; expr2 }.

```
x <- -5
if(x > 0){
  print("Non-negative number")
} else {
  print("Negative number")
}
```

if()...else

- The **condition** needs to evaluate to a single logical value
- Brackets **{}** are not necessary if you only have one expression and/or the **if()...else** statement are on one line.
- To avoid a syntax error, you should NOT have a newline between the closing bracket of the if statement and the else statement

```
if(condition) expression if TRUE
```

```
if(condition) {  
    expressions if TRUE  
}
```

```
if(condition) expression if TRUE else expression if FALSE
```

Conditional Element Selection

```
ifelse(test, yes, no)
```

```
ifelse()
```

Ifelse is a vectorised function, taking vectors as all its arguments.

- **test:** an object which can be coerced to logical mode
- **yes:** return values for true elements of test
- **no:** return values for false elements of test

```
x <- c(6:-4)
sqrt(x) #- gives warning
sqrt(ifelse(x >= 0, x, NA)) # no warning
```

```
## Note: the following also gives the warning !
ifelse(x >= 0, sqrt(x), NA)
```


Part II: Control Structures and Functions

FUNCTIONS

Functions

```
function name <- function(argument list) {  
    body  
    return  
}
```

function()

- The argument list is a comma-separated list of formal argument names
- Generally, the body is a group of R expressions contained in curly brackets {}.
- If the body is only one expression the curly brackets are optional.
- Functions are usually assigned names, but the names are optional (i.e. the FUN argument of `apply()`).

```
## Calculate mode  
Mode <- function(x) {  
  ux <- unique(x)  
  ux[which.max(tabulate(match(x, ux)))]  
}
```

Functions in R

Functions in R are created using the `function()` directive and stored as R objects.



Functions in R are objects of class “function”

- Functions can be passed as arguments to other functions;
- Functions can be nested: a function can be defined inside another function



The return value of a function is the last expression in the function body to be evaluated!

Function Arguments

Functions have named arguments which potentially have default values:

- The formal arguments are the arguments included in the function definition

```
## List of all formal arguments of a function:  
formals(mean.default)
```

Not every function call in R makes use of all the formal arguments:

- Function arguments can be missing or might have default values

Argument Matching

R functions arguments can be matched positionally or by name.

```
## Argument Matching:  
x1 <- rbinom(100, 10, 0.5)  
mean(x1)  
mean(x = x1)  
mean(x = x1, na.rm = FALSE)  
mean(na.rm = FALSE, x = x1)  
mean(na.rm = FALSE, x1)
```



Positional matching can be combined with matching by name

When an argument is matched by name, it is „taken out“ of the argument list and the remaining unnamed arguments are matched in the order that they are listed in the function definition

Argument List of a Function

```
args(name)
```

```
args()
```

- Displays the argument names and corresponding default values of a function or primitive.

```
## Argument List of a Function  
args(mean.default)
```

Lazy Evaluation

Arguments to functions are evaluated lazily → only evaluated as needed

```
## Lazy Evaluation  
f <- function(a, b) {  
  a^2  
}  
  
f(2)
```



In this function f, argument b will be never used, so calling f(2) will not produce an error because 2 gets positionally matched to a!

Lazy Evaluation (2) – Another Example

```
f <- function(a, b) {  
  print(a)  
  print(b)  
}  
  
f(45)
```



Notice what happens: 45 got printed first before the error was triggered.

This is because `b` did not have to be evaluated until after `print(a)`.
Once the function tried to evaluate `print(b)` it prints an error.

The “...” Argument

The ... argument indicate a variable number of arguments that are usually passed on to other functions.

... is often used when extending another function and copying the entire argument list of the original function should be avoided.

```
myplot <- function(x, y, type = "l", ...) {  
  plot(x, y, type = type, ...)  
}
```

Generic functions use ... so that extra arguments can be passed to methods

```
mean
```

The “...” Argument

The ... argument is also necessary when the number of arguments passed to the function cannot be known in advance

```
args(paste)  
args(cat)
```

Arguments that appear after ... on the argument list must be named explicitly and cannot be partially matched

```
args(paste)  
paste("a", "b", sep = ":")  
paste("a", "b", se = ":")
```

Byte Code Compiler in Package compiler

```
cmpfun(f, options = NULL)
```

```
cmpfun()
```

- f: a closure (function)
- options: list of named compiler options

```
# a simple example  
f <- function(x) x+1  
fc <- cmpfun(f)  
fc(2)  
disassemble(fc)
```

Control Structures

Function	
for()	Loop for a fixed number of iterations
while()	Loop until a conditions is FALSE
repeat	Repeat until iterations are halted with a call to break
break	Break out of a loop
next	Stop processing the current iteration and advance the looping index
apply	Applying functions to margins in data frames

See ?Control for the R documentation on control structures.

for loops

```
for(var in seq) {  
    expressions  
}
```

for()

- **var** is the **name of the loop variable** that changes with each iterations
- **seq** is an expression evaluating to any type of vector
- With each iteration **var** takes on the **next value in seq**.
 - At the **end of the loop var** will equal the **last value of seq**
- The number of iterations equals the length of seq
- The **{}** are only necessary if there is more than one expression
- By default R buffers the output created during any loop.

```
## DON'T TRY THIS AT HOME KIDS!  
mydf <- iris  
myve <- NULL # Creates empty storage container  
for(i in seq(along=mydf[,1])) {  
    myve <- c(myve, mean(as.numeric(mydf[i, 1:3])))  
}  
myve
```

Exercise 1: Calculate 10! using a for loop

```
# Calculate 10! using a for loop
f <- 1
for(i in 1:10) {
  f <- f*i
  cat(i, f, "\n")
}
f
factorial(10)
```

for-loops

Control structures in R allow you to control the flow of execution of the program, depending on runtime conditions.

These three loops have the same behavior:

```
x <- c("a", "b", "c", "d")
for(i in 1:4) {
  print(x[i])
}
for(i in seq_along(x)) {
  print(x[i])
}
for(letter in x) {
  print(letter)
}
for(i in 1:4) print(x[i])
```

while loops

```
while(cond) {  
    expressions  
}
```

while()

- **cond** is a single logical value that is not NA
- The {} are only necessary if there is more than one expression
- If you are going to use a while loop, you need to have an indicator variable i and change its value within each iteration.
 - Otherwise you will have an infinite loop

```
## while loop in R  
z <- 0  
while(z < 5) {  
  z <- z + 2  
  print(z)  
}
```


Exercise 1: Calculate 10! using a while loop

```
i <- 10
f <- 1
while(i>1) {
  f <- i*f
  i <- i-1
  cat(i, f, "\n")
}
f
factorial(10)
```

repeat loops

```
repeat {  
    expressions  
    if(cond) break  
}
```

repeat()

Loop is repeated until a break is specified. This means there needs to be a second statement to test whether or not to break from the loop.

- The repeat loop does not contain a limit.
- Therefore it is necessary to include an if statement with the break command to make sure you do not have an infinite loop.

```
## repeat loop in R  
z <- 0  
repeat {  
    z <- z + 1  
    print(z)  
    if(z > 100) break()  
}
```

Exercise 1: Calculate 10! using a repeat loop

```
i <- 10  
f <- 1  
repeat {  
  f <- i*f  
  i <- i-1  
  cat(i, f, "\n")  
  if(i<1) break  
}  
f  
factorial(10)
```

Loops and Flow Control

AVOIDING LOOPS

Avoiding Loops

- With R it is a good idea to try and avoid including loops in programs.
- Code that takes a „whole object“ approach versus an iterative approach will often be faster in R.
 - For example, `ifelse()` or `cumsum()`
- The `apply()` family of functions does not reduce the number of function calls.
 - So using `apply()` may not necessarily improve efficiency.
 - However, `apply()` is still a great function for making code more transparent and compact
- When developing code it may be easier to write an algorithm using a loop.
 - That's fine, but try and replace the loop with a more efficient expression after you have a draft program.
- Loops are still useful tools, such as with simulation studies

Example – Avoiding Loops

- Consider an extreme example, suppose we want the sum of 10 million random standard normal numbers.
- The **sum()** function, which uses the whole vector, is noticeably faster than the for loop, which uses each element

```
z <- rnorm(1E7)
system.time(sum(z))
```

```
total=0
system.time(for(i in 1:length(z))
  total <- total + z[i])
```

Speed comparison of for loops with an append versus an inject step

```
myMA <- matrix(rnorm(1000000), 100000, 10, dimnames=list(1:100000, paste("C",  
1:10, sep=""))))
```

```
results <- NULL
```

```
system.time(for(i in seq(along=myMA[,1])) results <- c(results, mean(myMA[i,])))
```

```
user system elapsed
```

```
39.156 6.369 45.559
```

```
results <- numeric(length(myMA[,1]))
```

```
system.time(for(i in seq(along=myMA[,1])) results[i] <- mean(myMA[i,]))
```

```
user system elapsed
```

```
1.550 0.005 1.556
```

Speed comparison of apply loop versus rowMeans for computing meanf for each row in large matrix

```
system.time(myMAmean <- apply(myMA, 1, mean))
```

```
user system elapsed
```

```
1.452 0.005 1.456
```

```
system.time(myMAmean <- rowMeans(myMA))
```

```
user system elapsed
```

```
0.005 0.001 0.006
```


Speed comparison of apply loop versus vectorized approach for computing the standard deviation of each row

```
system.time(myMAsd <- apply(myMA, 1, sd))
```

```
  user  system elapsed
```

```
3.707  0.014  3.721
```

```
myMAsd[1:4]
```

```
      1      2      3      4
```

```
0.8505795 1.3419460 1.3768646 1.3005428
```

```
system.time(myMAsd <- sqrt((rowSums((myMA-rowMeans(myMA))^2)) /  
(length(myMA[1,])-1)))
```

```
  user  system elapsed
```

```
0.020  0.009  0.028
```

```
myMAsd[1:4]
```

```
      1      2      3      4
```

```
0.8505795 1.3419460 1.3768646 1.3005428
```


Graphics in R

masem research institute GmbH
Bereich masem training

Unter den Eichen 5, Gebäude G,
D-65195 Wiesbaden
Registergericht Wiesbaden HRB 25700
Geschäftsführer: Dr. Guido Möser

Wissenschaftlicher Beirat:
Prof. Dr. Peter Schmidt

www.masem-training.com

R Graphics

- R is capable of creating **high quality graphics**
 - **plot()** function is *working horse*
- Graphs are typically created using a series of **high-level** and **low-level plotting commands**.
 - **High-Level Functions** create new plots, e.g. **plot()**, e.g. **scatterplot**
 - **Low-Level Functions** add information to an existing plot, e.g., **fit line**
- **Customize graphs** (line style, symbols, color, etc) by specifying graphical parameters
 - Specify graphic options using the **par()** function
 - Can also include graphic options as additional arguments to plotting functions

Graphic Parameters: `par()`

- The function **`par()`** is used to set or get graphical parameters
 - This function contains **70** possible settings and allows to adjust almost any feature of a graph.
- Graphic parameters are **reset** to defaults with each new graphic device
- To **extract** a graphic parameter, **`par("tag")`** or **`par()$tag`**
- To **set** a graphic parameter, **`par(tag = value)`**
- Most elements of **`par()`** can be set as additional arguments to a plot command, however, there are some that can only be set by a call to **`par()`**, **`mar`**, **`oma`**, **`mfrow`**, **`mfcoll`** etc. – see documentation for others

High-Level Plot Functions

Function	
plot()	Scatterplot / Line Graphs
hist()	Histogram
boxplot()	Boxplot
qqplot(), qqnorm(), qqline()	Quantile plots
interaction.plot	Interaction plot
sunflowerplot()	Sunflower scatterplots
pairs()	Scatter plot matrix
symbols()	Draw symbols on a plot
dotchart(), barplot(), pie()	Dot chart, bar chart, pie chart
curve()	Draw a curve from a given function
image()	Create a grid of colored rectangles with colors based on the values of a third variable
contour(), filled.contour()	Contour plot
persp()	Plot 3-D surface

Generic X-Y Plotting

```
plot(x, y, ...)
```

```
plot()
```

- Generic function for **plotting** of R objects.
- For more details about the graphical parameter arguments, see **par**.
- For simple scatter plots, **plot.default** will be used.
- However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use **methods(plot)** and the documentation for these.

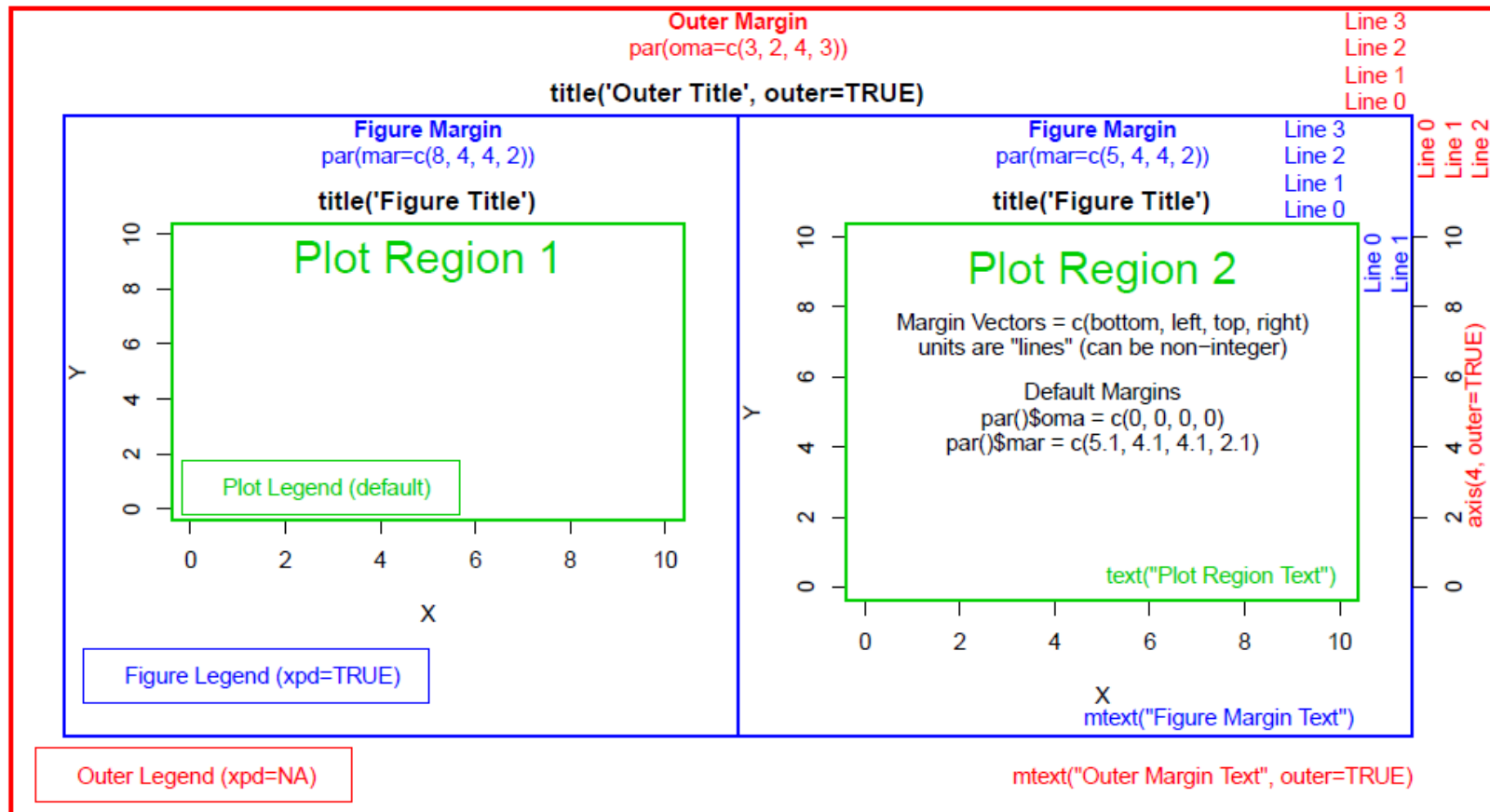
```
## high-level plot  
plot(cars)  
## low-level plot  
lines(lowess(cars))  
## Discrete Distribution Plot:  
plot(table(rpois(100, 5)), type = "h", col = "red", lwd = 10,  
      main = "rpois(100, lambda = 5)")
```

Low-Level Plot Functions

Function	
<code>points()</code>	Add points to a figure
<code>lines()</code>	Add lines to a figure
<code>text()</code>	Insert text in the plot region
<code>mtext()</code>	Insert text in the figure and outer margins
<code>title()</code>	Add figure title or outer title
<code>legend()</code>	Insert legend
<code>axis()</code> , <code>axis.Date()</code>	Customize axes
<code>abline()</code>	Add horizontal and vertical lines or a single line
<code>box()</code>	Draw a box around the current plot
<code>rug()</code>	Add a 1-D plot of the data to the figure
<code>polygon()</code>	Draw a polygon
<code>rect()</code>	Draw a rectangle
<code>arrows()</code>	Draw arrows
<code>segments()</code>	Draw line segments
<code>trans3d()</code>	Add 2-D components to a 3-D plot



Margins



Set or Query Graphical Parameters

```
par(..., no.readonly = FALSE)
```

```
par()
```

- **par** can be used to set or query graphical parameters.
- Parameters can be set by specifying them as arguments to par in tag = value form, or by passing them as a list of tagged values.

```
## Show whole list of parameters
op <- par(no.readonly = TRUE) # the whole list of settable par's.
## do lots of plotting and par(.) calls, then reset:
par(op)
## Change settings and reset after plotting
op <- par(mfrow = c(2, 2), # 2 x 2 pictures on one plot
         pty = "s")      # square plotting region,
                        # independent of device size
## At end of plotting, reset to previous settings:
par(op)
```

Graphics in R

A SHORT INTRODUCTION INTO GGLOT2

Main Ressource: Cheat Sheet by Rstudio:
<https://www.rstudio.com/wp-content/uploads/2016/11/ggplot2-cheatsheet-2.1.pdf>

ggplot2: Basics

ggplot2 → based on the idea of the grammar of graphics.
A graphic consists of **three components**:

- | | | |
|---|-------------------|--|
| 1 | Data set | Specified in ggplot()-function using the data-argument |
| 2 | Coordinate system | |
| 3 | geoms | Geoms are visual marks that represent data points. |

To display variables, map variables in the data to visual properties of the geom (aesthetics) like color, size, x and y locations etc.

ggplot2: Template

```
ggplot(data = <DATA>) +  
  <GEOM_FUNCTION> (  
    mapping = aes (<MAPPINGS>),  
    stat = <STAT>,  
    position = <POSITION>  
  ) +  
  <COORDINATE_FUNCTION> +  
  <FACET_FUNCTION> +  
  <SCALE_FUNCTION> +  
  <THEME_FUNCTION>
```

REQUIRED

NOT REQUIRED

```
ggplot(data = cars, aes(x = speed, y = dist)) + geom_point()
```

ggplot2: Major functions

1

```
ggplot(data = cars, aes(x = speed, y =  
dist)) + geom_point()
```

- **ggplot()** begins a plot: specification of data and coordinate systems (in aes).
- **geom_points()** requests a scatter plot

2

```
qplot(x = speed, y = dist, data = cars,  
geom = "point")
```

Creates a complete plot with data, geom, and mappings.

3

```
last_plots()
```

Returns the last plot

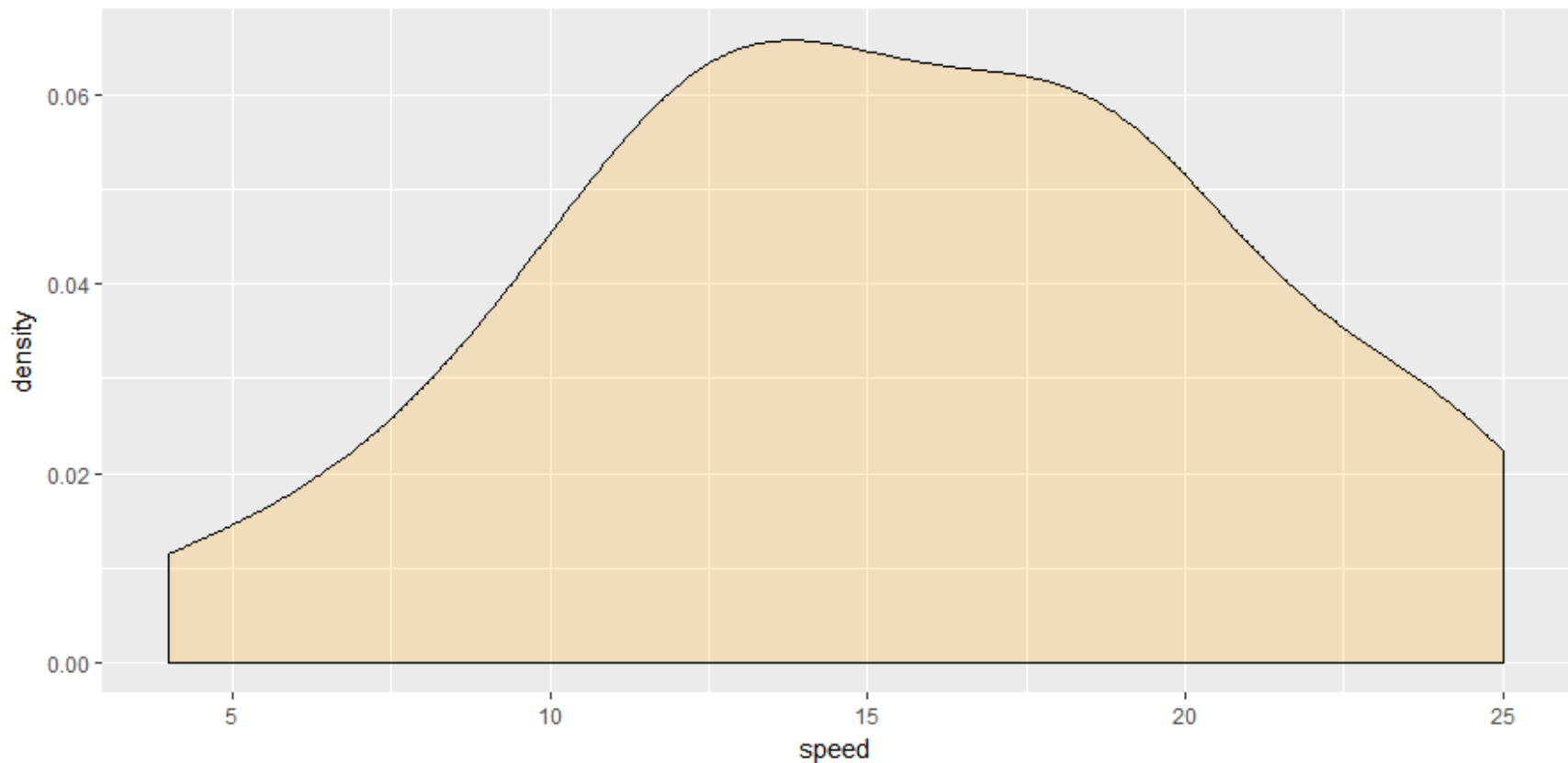
4

```
ggsave("plot.png", width = 10, height =  
10)
```

- Save the last plot as 10 x 10 file names plot.png in working directory.
- Matches file type to file extension

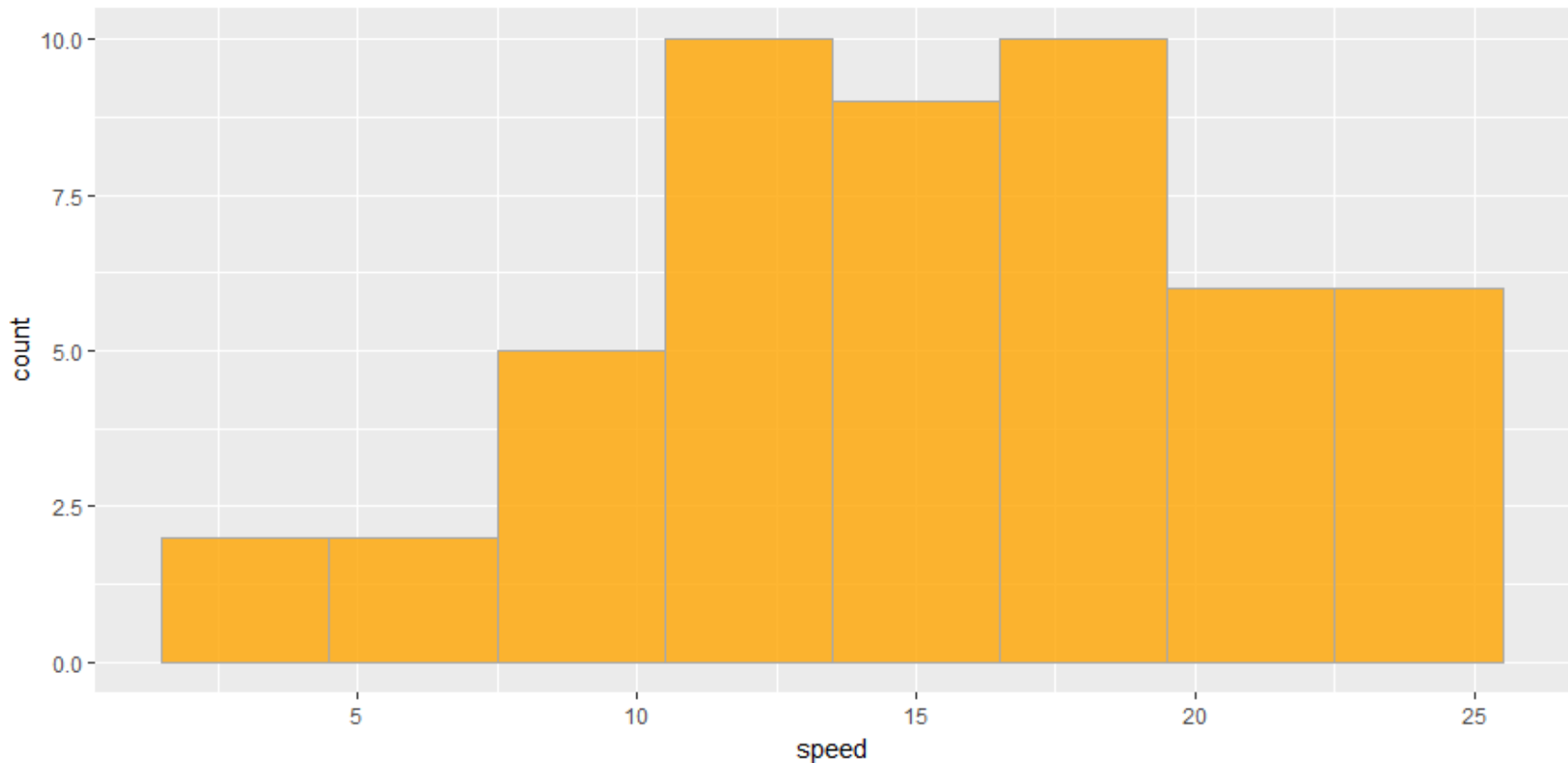
One Continuous Variable: Density Plot

```
ggplot(data = cars, aes(x = speed)) + geom_density(alpha = 0.2,  
fill = "orange")
```



One Continuous Variable: Histogram

```
ggplot(data = cars, aes(x = speed)) + geom_histogram(binwidth =  
3, alpha = 0.8, color = "darkgrey", fill = "orange")
```



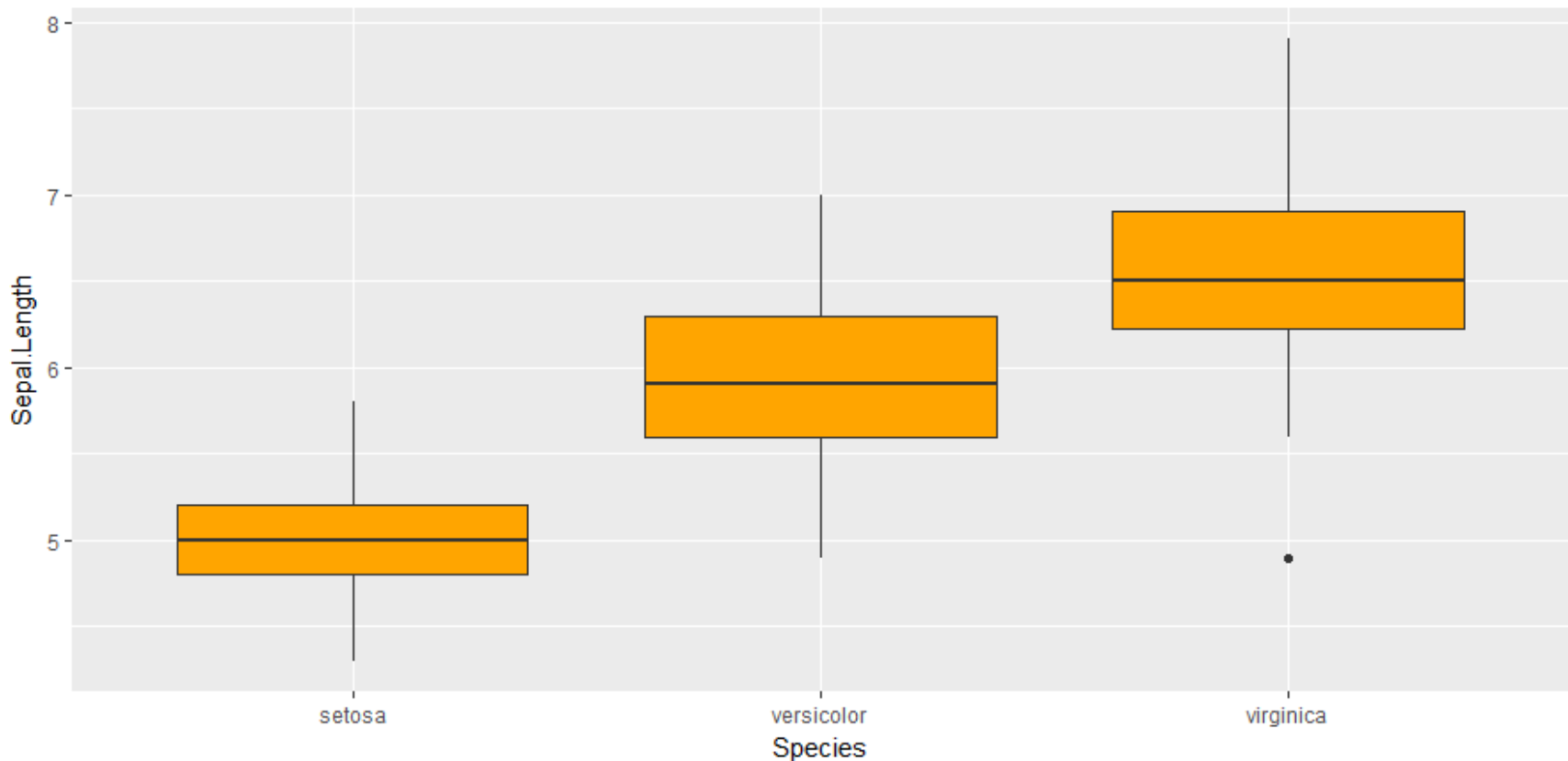
One Discrete Variable: Barchart

```
ggplot(data = iris, aes(x = Species)) + geom_bar(fill = "orange",  
color = "darkgrey")
```



Two Variables: Discrete x, Continuous y: Boxplot

```
ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +  
geom_boxplot(fill = "orange")
```



Two Variables: Continuous x, Continuous y: Boxplot

```
ggplot(data = cars, aes(x = speed, y = dist)) + geom_point()
```

