Training R

Outline

Welcome to R!

A rough timeline for today:

Part I 9:00 – 10:00

About R – website, information, help

Basics and data

Part II 10:00 – 12:00

data manipulation

statistical methods

Graphics I

--- --- *Lunch break* --- ---

Part III 13:00 – 14:00

Graphics II

Part IV 14:00 – 15:00

R-related software: *Mondrian* and *iplots*

Intro

Why R?

- R provides a wide range of methods via package extensions
- Quick and professional graphics
- R as a language: intuitive code for mathods and data transformation
- R is free

R is a language!

In order to learn a language properly you have to study

- grammar (syntax, conventions)
- vocabulary (names of methods, functions and packages)
- how to look up things

And most of all you have to practice.

Good practice in R

Some rules to follow while working with R

- Assign names to new objects
- Test commands and options step by step
- Check the defaults in operation
- Combine functions like phrases in a language
- Store sessions in textfiles
- Annotate your code

Information

Where to retrieve…

… packages?

The packages are available on the CRAN-R website
http://www.cran.r-project.org
or can be installed via the **JGR package installer**

... help?

- The comprehensive R-help (accessible e.g. via `?`) offers information about the fucntionality of the methods as well as examples, references and more.

- **Task Views** are available on CRAN to help with the selection of the packages for different fields of interest.

- Moreover there are vignettes, R-Wiki,…

About R – websites and help

The official R-website is

http://www.R-project.org

For downloads, packages, binaries, manuals and more visit:

http://cran.r-project.org

(Comprehensive R Archive Network)

Of most importance are

Task Views (CRAN)	Task Views are guidelines for the selection of packages for different fields of interest, e.g. finance.
Packages(CRAN)	Packages can be found and downloaded from CRAN
Manuals FAQs Wiki Books **...**	There are some excellent manuals for different fields of working with R. For every package there exists a **reference** manual. Moreover there are specific manuals called **vignettes** for some packages. Further help can be found in the FAQ, Wiki, Books and mailing lists!

The R-help

R offers a comprehensive help with information about the functionalities of the methods as well as examples and references.

- The R help can be accessed via the **JGR help button** or by calling *help.start().*
- Help for a particular function can be found via *?topic* (e.g. *?summary*)

The help pages consist of the following parts:

`Description`	A short description of the method
`Usage`	The complete syntax
`Arguments`	Description of the arguments
`Details`	More detailed information
`Value`	Information on the return values or objects
...	...
`See Also`	Links to related methods
`References`	Listing of references
`Examples`	Examples for the method

The R-help

example: *?Normal*

Keyword and package

Short description

Syntax:
Parameters without default value amongst the first arguments necessarily need to be specified!

Arguments:
A short description of the parameters

Normal {stats} R Documentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

Usage

```
dnorm(x, mean=0, sd=1, log = FALSE)
pnorm(q, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean=0, sd=1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean=0, sd=1)
```

Arguments

`x,q`	vector of quantiles.
`p`	vector of probabilities.
`n`	number of observations. If `length(n) > 1`, the length is taken to be the number required.
`mean`	vector of means.
`sd`	vector of standard deviations.
`log, log.p`	logical; if TRUE, probabilities p are given as log(p).
`lower.tail`	logical; if TRUE (default), probabilities are $P[X <= x]$, otherwise, $P[X > x]$.

Notation

```
<- or =        Assignment
;              command separator
{ }            code block
[ ]            array index
[[ ]]          list index
$              (dataset)$variable
&              AND
|              OR
==             exact equality
!              NOT
" "            string
#              comments

%*%            matrix multiplication
%in%           is element
?              R help
```

Data import and export

The first step of working with R is to load the data.

Almost any software offers some way of saving the data in textfiles.
Textfiles can be read by ***read.table()*** or via the JGR menu.

However there are also import functions for almost all important formats:

read.spss()	reads SPSS data files
read.ssd()	reads SAS data files via read.xport()
read.xls(), ***read.csv()***	reads excel spreadsheets resp. *csv* files
…	

```
> CARS <- read.table("C:/traffix/CARS.txt",header=TRUE,sep="\t",quote="",dec=".",na.strings ="NA")
```

These functions work very similar. The most important parameter is the separator ***sep***.

In the other direction it is recommendable to save the data **in (tab-delimited) textfile format**.

```
> write.table(CARS,"C:/traffix/CARS.txt",append=F,quote=F,sep="\t",row.names=F)
```

Other programs will usually be able to import such files!

Data import and export II

The size of the datasets which R is able to handle depends on

- the system
- the functions which shall be applied to it

For large datasets it is not efficient (or even possible) to load the data completely into R.

In R there are different packages available for the connection to **databases**.

For ODBC the package **RODBC** is a good choice:

```
> library(RODBC)
> con = odbcConnect("ODBCone")
> query = sqlQuery(con,"SELECT * FROM mydatabase")
```

For MySQL the package **RMySQL** can be used:

```
> library(RMySQL)
> drv <- dbDriver("MySQL")
> con = dbConnect(drv,dbname="mydatabase1",user="root",password="",host="localhost")
> mydata=dbGetQuery(con,"SELECT * FROM mydatabase1.table1")
```

Access and Excel can be connected via **RODBC** and for Oracle there is another package called **ROracle**, which should work similar to the others.

Data import and export III

Errors and problems with data import arise from:

- wrong separator specification
- wrong specification for quotes and decimal separator
- wrong specification of missing values
- problematic characters and symbols in the data file
  e.g. \ # ß äöü ? ...
- missing or too many separators in the data file
- for databases: problems with the driver
- special encodings for numeric variables, e.g. 2.718*e-04

Dataframes

A data.frame consists of rows (cases) and columns (variables).

In R objects with such a build-up can be accessed in the following way

object[x,]	for row x
object[,y]	for column y
object[x,y]	for a single element

If an attribute of an objects has a name, this can also be used to access it, e.g.

Var1 <- eco09$kW will save the *kW* variable from the *eco09* dataset in an object called *Var1*.

The dimensions of an object can be obtained by

ncol*(object)*, ***nrow****(object)*

length*(object)*

dim*(object)*

Objects

R allows different types of objects which are in general differentiated by their **class** attribute.

For instance the results of a simple linear regression are of class „lm“.
The function ***attributes()*** returns the class as well as the names of the attributes:

```
> m1 = lm(Horsepower~Engine.Size*Cylinders, data = CARS)
> m1$coefficients
          (Intercept)           Engine.Size            Cylinders Engine.Size:Cylinders
            81.665178              9.655677             7.843083              2.891979
> attributes(m1)
$names
 [1] "coefficients"  "residuals"     "effects"      "rank"          "fitted.values"
 [6] "assign"        "qr"            "df.residual"  "xlevels"       "call"
[11] "terms"         "model"

$class
[1] "lm"
```

Many objects have the class *„list“*.
Their elements are adressed by **object[[i]]** and can themselves be of different types.

```
> m1 = lm(Horsepower~Drive)
> L = list( c("a","b","c"), summary(m1), 5)
> L[[1]]
[1] "a" "b" "c"
```

Objects

If an object can be accessed by object[i], object[[i]] or object[i,j],…
i and ***j*** can also be **vectors of indices** or **logical vectors**.

```
> (M <- CARS[1:2,1:5])
                   Vehicle.Name  Type Drive Retail.Price Dealer.Cost
1           Chevrolet Aveo 4dr Sedan front        11690       10965
2 Chevrolet Aveo LS 4dr hatch Sedan front        12585       11802
> M[ c(T,F), c(T,F,F,T) ]
          Vehicle.Name Retail.Price Dealer.Cost
1 Chevrolet Aveo 4dr        11690       10965
```

Therefor logical operators and the function ***which****(expression)* are very useful.

```
> which( names(M) %in% c("Vehicle.Name","Drive") )
[1] 1 3
> M2 = CARS[ CARS$Retail.Price > 50000, ]
> dim(M2)
[1] 52 14
```

Reordering objects is done by using ***order****(x)* as an index vector or by the function ***sort****(x):*

```
> sort( c(1,4,3,2) )
[1] 1 2 3 4
> CARS <- CARS[order(CARS$Horsepower),]
```

Classes

The class of a dataframe is ***data.frame*** and its variables are either ***factor, integer, double, logical*** or ***character***.

Categorical variables are of class *factor* and they have a different number of ***levels***.

```
> levels(Drive)
[1] "AWD"   "front" "rear"
> nlevels(Drive)
[1] 3
```

Functions for conversion and querying exist for many classes.

as.<class>*(Objekt)*	changes the class of the object to the specified one (if possible)
is.<class>*(Objekt)*	cheacks whether the object is of a particular class
class*(Object)*	simply returns the classes of the object

```
> X = as.integer(Drive)
> table(Drive,X)
       X
Drive     1   2   3
  AWD    92   0   0
  front   0 226   0
  rear    0   0 110
> Y = as.factor(X)
> levels(Y)
[1] "1" "2" "3"
> is.integer(Y)
[1] FALSE
> class(Drive)
[1] "factor"
```

Generic functions

Classes are also the backbone of the conept of **generic** functions.

A generic function *fun(x,…)* checks the classes of *x* and tries to call a function *fun.class1(x,…).*

Should that fail it tries *fun.class2(x,…)* and so on.

That allows functions like ***plot, summary*** and ***print*** to work for different types of input.

```
> summary(Drive)
  AWD front  rear
   92   226   110
> summary(Horsepower)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   73.0   165.0   210.0   215.9   255.0   500.0
```

There are many more generic functions available.
It is possible to implement more variations for them based on other classes.

Some important statistics

R offers a lot of basic statistics:

mean()
median()
min(), max()
var(),cov(),cor()
sd()
summary()
quantile()
length()
range()

```
> range(X)
[1]   659 5666
```

```
> summary(X)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    659    1587    1910    1983    2148    5666
> var(X)
[1] 470063.4
> sd(X)
[1] 685.6117
> quantile(X,probs=c(0.2,0.5,0.8))
 20%  50%  80%
1493 1910 2204
```

(rounded) correlation matrix:

```
> round(  cor(cbind(kW,CCM,FuelCons))  ,digits=2)
           kW  CCM FuelCons
kW       1.00 0.91     0.69
CCM      0.91 1.00     0.68
FuelCons 0.69 0.68     1.00
```

Drawing **samples** from a set with or without replacement:

```
> sample( 1:9, size = 4, replace = FALSE )
[1] 9 5 8 7
> sample( 1:3, size = 4, replace = TRUE )
[1] 2 3 2 1
```

Utility functions

Several functions allow to generate special vectors comfortably:

rep(*x,times,each)* generates repetitions of *x* in a vector
seq(*from,to,by)* generates a sequence *[from, to]* with stepwidth *by*

```
> rep(4.3,5)
[1] 4.3 4.3 4.3 4.3 4.3
> rep( c("a","b","c"), each = 2)
[1] "a" "a" "b" "b" "c" "c"
> seq(1,3,0.5)
[1] 1.0 1.5 2.0 2.5 3.0
```

Different **concatenation** functions are provided:

c(*x1,…xn)* unites *x1,…,xn* in a vector
list(*x1,…xn)* unites *x1,…,xn* in a list

paste(*x, y, z,…)* concatenates *x,y,z,…* as a string (vectors: elementwise!)

cbind(*c1,...,cn)* appends vectors *c1,...,cn* as columns
rbind(*x1,...,xn)* appends vectors *c1,...,cn* as rows

```
> a = c(1,2,5)
> b = list("a","b","e")
> paste(a,b,sep=":")
[1] "1:a" "2:b" "5:e"
> cbind(a,b)
     a b
[1,] 1 "a"
[2,] 2 "b"
[3,] 5 "e"
> rbind(a,b)
  [,1] [,2] [,3]
a 1    2    5
b "a"  "b"  "e"
```

Utility functions

Several functions in R allow to create or handle **tables**.

Three of the most important commands are:

ftable() computes flat contingency (frequency) tables for categorical variables.

```
> ftable(CARS[,2:3],row.vars = 1:2)
Type       Drive
Mini Van   AWD       3
           front    16
           rear      1
Pickup     AWD      12
           front     0
           rear     12
Sedan      AWD      25
           front   166
           rear     54
Sports Car AWD       5
           front     8
           rear     36
SUV        AWD      38
           front    22
           rear      0
Wagon      AWD       9
           front    14
           rear      7
```

table() is the standard command for tables

```
> table(Type,Drive)
             Drive
Type          AWD front rear
  Mini Van      3    16    1
  Pickup       12     0   12
  Sedan        25   166   54
  Sports Car    5     8   36
  SUV          38    22    0
  Wagon         9    14    7
```

xtabs() is specified by a formula and allows weights

```
> xtabs(Horsepower~Type+Drive)
             Drive
Type           AWD front  rear
  Mini Van     590  3309   190
  Pickup      2952     0  2444
  Sedan       5659 29276 14086
  Sports Car  1317  1953 10654
  SUV         9057  5092     0
  Wagon       2054  2257  1509
```

Utility functions

It is usually not recommendable to use loops in R, because they are very slow.

Instead the **apply** functions can be used:

command	**computes…**
tapply*(var,group,fun)*	… *fun(var)* for each level of group
by*(data,group,fun)*	… *fun(var)* for each variable in data and for each level of group
apply*(M,i,fun)*	… *fun(x)* for each vector *x* of *M*'s *i*-th dimension
sapply*(D,fun)*	… *fun(x)* for each list element of *D*
lapply*(D,fun)*	… *fun(x)* for each list element of *D*
mapply*(fun,x,y,z)*	… *fun(x,y,z)* for the k-th elements of *x* and *y*

examples:

```
> tapply(Horsepower,Type,mean)
  Mini Van      Pickup       Sedan  Sports Car         SUV       Wagon
  204.4500    224.8333    200.0857    284.1633    235.8167    194.0000
> M = matrix(1:16,ncol=4)
> apply(M,1,sum)
[1] 28 32 36 40
```

Distributions and random samples

Probability distributions, densities, quantile functions and random samples are all obtained from the same type of commands.

These commands consist of
-> a character **p,d,q** or **r** (distribution, density, quantiles and sample)
-> the **keyword** of the distribution (e.g. *norm, binom, chisq*,...)

```
> pnorm(q=0.75, mean=0, sd=1)
[1] 0.7733726
> rbinom(n=4, size=1, prob=0.5)
[1] 1 0 1 0
```

pnorm(...): the $q = 0.75$ quantile of the standard normal distribution

rbinom(...): tossing a coin 4 times

some more:

keyword	**distribution**
norm, lnorm	(log-) normal
exp	exponatial
gamma	gamma
binom, nbinom	(neg.) binomial
pois	Poisson
chisq	Chi²

Tests

R also offers statistical tests. Not unlike the distributions many are quite similar to handle.

Four popular representatives:

t.test()	classical t-Test for one sample or two paired /independent samples
chisq.test()	Chi^2 -Test for independence of contingency tables
ks.test()	Kolmogorov-Smirnov-Test for equality of distributions
wilcox.test()	Nonparametric Signed-Rank and Rank-Sum Tests

Examples:

t-Test:
Is mean(Horsepower) equal for cars with front resp. rear drive?

```
> front = Horsepower[which(Drive == "front")]
> rear = Horsepower[which(Drive == "rear")]
> t.test(front,rear)

      Welch Two Sample t-test

data:  front and rear
t = -9.1642, df = 167.175, p-value < 2.2e-16
alternative hypothesis: true difference in
means is not equal to 0
95 percent confidence interval:
 -93.87030 -60.59374
sample estimates:
mean of x mean of y
 185.3407  262.5727
```

Chi²-Test:
Are Drive and Type independent?

```
> tt = table(Drive,Type)
> chisq.test(tt)

      Pearson's Chi-squared test

data:  tt
X-squared = 187.2532, df = 10, p-value <
2.2e-16
```

Linear models

A simple linear regression is available in R through the function ***lm****(Y~X1+…+Xn).*

```
> LR <- lm(Horsepower~Engine.Size,data=CARS)
```

The function ***glm****(Y~X1+…+Xn, family = „gaussian")* works quite the same way:

```
> linreg <- glm(Horsepower~Engine.Size,data=CARS, family="gaussian")
```

But it also offers **log-linear** models and **logistic regression**!
Therefore the *family* argument has to be defined:

```
> pois <- glm(Freq~Survived*Class+(Sex+Age)^2, data=Titanic, family="poisson")
> binom <- glm(Survived~Class*Sex+Age,data=Titanic, family="binomial")
```

Many important informations about the model are contained in the ***summary()*** of the model object, e.g.:

```
> summary(LR)$r.squared
[1] 0.6200538
```

An analysis of variance for the model can be computed by ***aov****(model)* or ***anova****(model).*

The functions ***polr()*** and ***multinom()*** provide **proportional odds** logistic regression and **multinomial** logistic regression.

Predictions

The fitting values of a model or a smoother are either contained in the returned objects or have to be predicted.

Moreover some models are also meant for the prediction of **new data**.

Therefor the generic function ***predict(object,...)*** is implemented for almost all types of models (***object***s):

lm, glm, trees (rpart), neural networks (nnet), smoothers (loess),
principal components analysis (prcomp), ... and many more

Specifics for each model/object type can be found in the corresponding help page, e.g. ***?predict.loess***

```
> LR <- lm(Horsepower~Engine.Size,data=CARS)
> LRfit <- predict(LR, newdata=CARS2, type="response")
```

Custom functions

Writing your own functions in R is of a simple form which shall be demonstrated by a simple example:

```
myownfun <- function(x,y,...){
   stopifnot( is.numeric(x) & is.integer(y) )
   if( x > y ){
       z = y/x
   }else{
       z = 1
       for( i in 1:3 ){
           z = z*x/y
       }
   }
   return(z)
}
```

Graphics I - basics

One of the strenghs of R is that it provides functions to generate **professional graphics.**

The (generic) default command ***plot(Object)*** has already been mentioned and is available for many classes.

Moreover there are a lot of other commands to create a particular type of graphic:
Some examples are:

hist()	histogram
boxplot()	boxplot
barplot()	barchart
pairs()	scatterplot matrix
parcoord()	parallel coordinates plot (MASS)
densityplot() / histogram()	density plots (lattice)
ibar()/ihist()/ipcp()/…	interactive graphics similar to Mondrian (iplots)
mosaicplot()	mosaicplots
xyplot(), bwplot(),…	Trellisplots (lattice)
rmb()	Relative Multiple Barchart (extracat)

Graphics I - basics

Graphics usually have a large number of possible layout parameters.

They can either be found in the description of the corresponding help file or they are part of a family of graphic parameters.

One of the most important parameters families is **par** (see *?par*).

xlim/ylim	*= c(min,max)*	axis intervals
xaxp/yaxp	*= c(min,max,steps)*	tickmarks for the axes
pch	*= integer, „symbol“*	symbol for data points
lwd	*= double*	witdh of lines and points
main	*= „text“*	header
xlab/ylab	*= „text“*	axis labeling

Moreover there exists a wide range of color palettes. The color is usually set by ***col*** *= …*

```
> plot(Engine.Size,Horsepower, col = 2)
> plot(Engine.Size,Horsepower, col = "red")
> plot(Engine.Size,Horsepower, col = unclass(Type))
> plot(Engine.Size,Horsepower, col = rgb(red=1, green=0, blue=0, alpha = 0.3) )
> plot(Engine.Size,Horsepower, col = hcl(h=1, c=0, l=0, alpha = 0.3) )
> plot(Engine.Size,Horsepower, col = hsv(h=1, s=0.7, v=0.7, alpha = 0.3) )
> plot(Engine.Size,Horsepower, col = rainbow(n=3, s=0.7, v=0.7, gamma = 1, alpha = 0.3) )
```

the simple way

by another variable

palettes

Graphics I - basics

It is possible to add information to a graphic and to arrange multiple plots in one graphic device.

Additional points, lines or **rectangles** can be added by

points(*x,y*)	adds additional points similar to ***plot***(*x,y*)
lines(*data*)	connects datapoints with lines (in their given order!)
abline(*a,b,v,h*)	draws lines **a + b*x** and/or vertical/horizontal lines
rect(*xleft, ybottom, xright, ytop*)	draws a rectangle in the plot

-> The graphic parameters are those of **par**()

The function ***curve(fun(x,...), from, to,..., add=T)*** can draw function graphs **fun(x)~x**

To open a new graphics device call

dev.new *(width,height)*	*or*	***JavaGD***(*width,height) on JGR*
dev.new()	*or*	***JavaGD***() *on JGR*

For multiple plots in one window the **par**-commands ***mfrow()*** / ***mfcol()*** are useful:

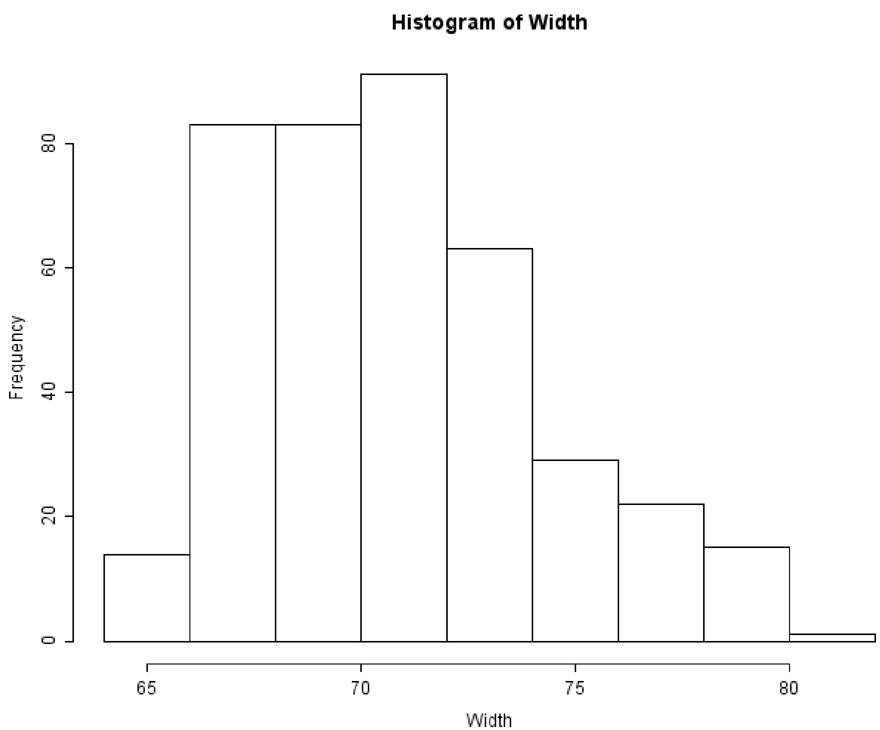
*par(**mfrow**=c(rows,cols))*

Graphics I - histograms

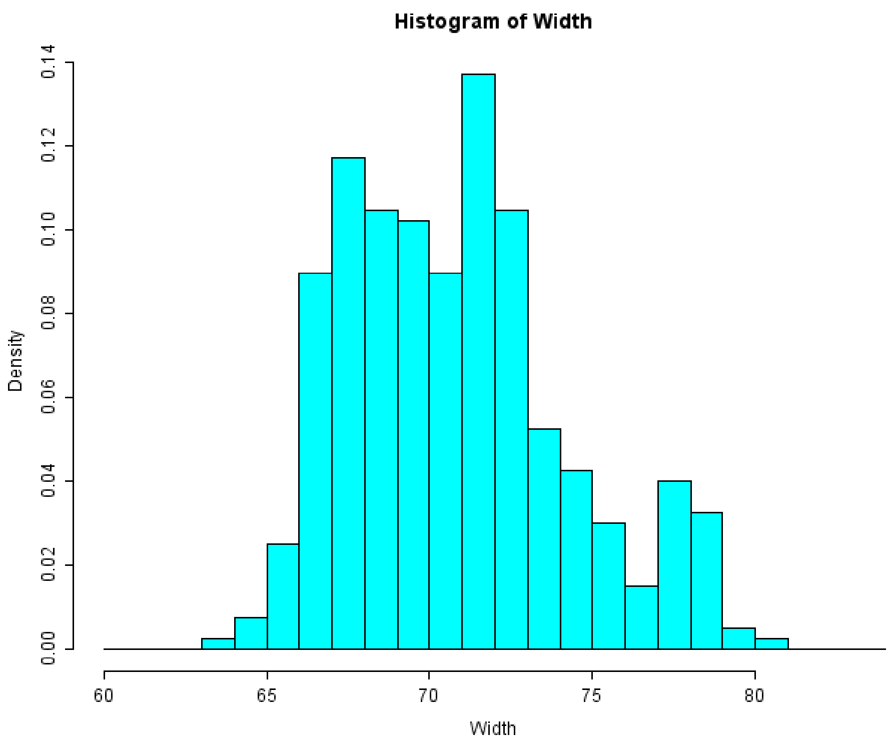
Histograms are generated using ***hist***(*x, breaks, freq,...)* or the ***histogram***(*x,data, breaks,...)* command from the package *lattice*.

x should be a numeric or integer variable and ***breaks*** is either the number of breakpoints or a point sequence (see *?seq*).

```
> hist(Width)
```




```
> hist(Width, freq=F, breaks = seq(60,84,1), col=5)
```

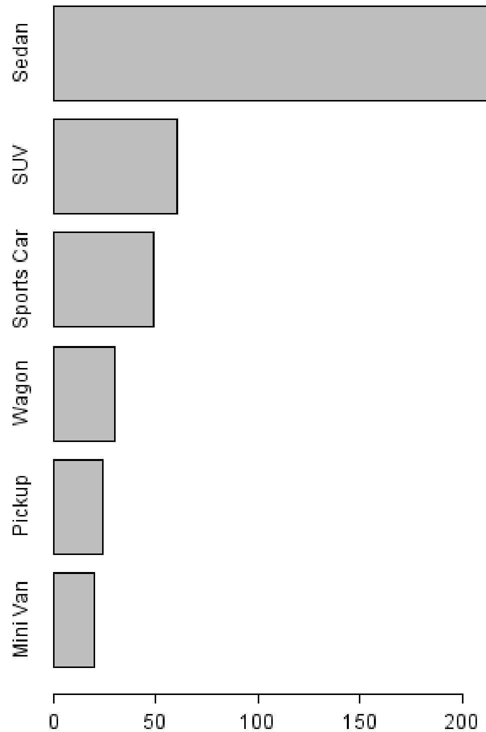


Graphics II – barcharts

Barcharts are generated using the ***barplot****(height)* command where height is either a **vector of values**, a **matrix** or a **table**. Examples:

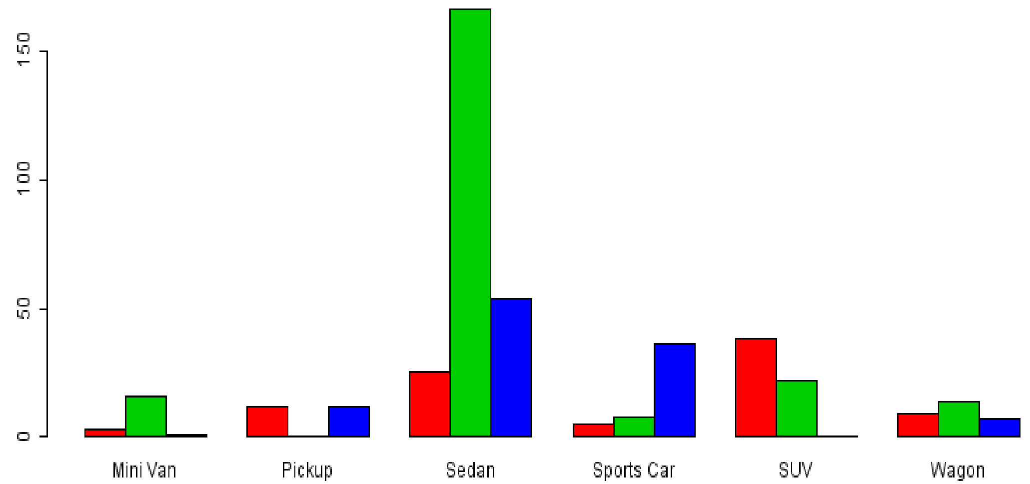
For a single variable in horizontal direction, sorted by counts:

```
> barplot( sort( table(Type) ), horiz=T)
```



For two variables with new colors and without stacked bars:

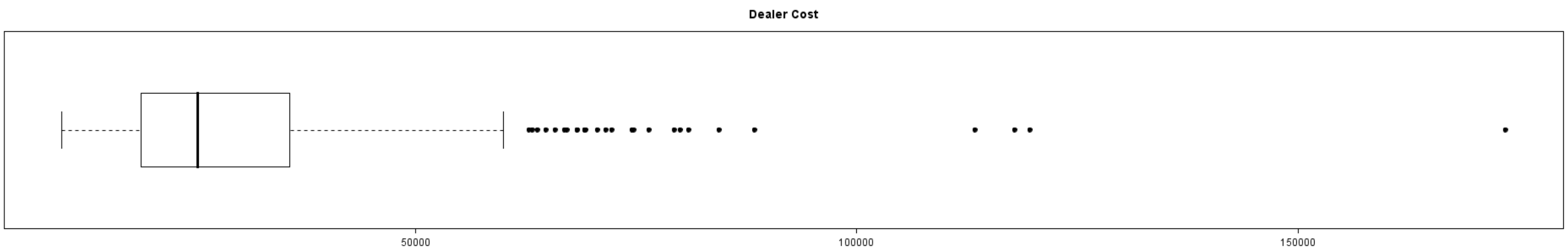
```
> barplot( table(Drive,Type), col=2:4, beside=T)
```



Graphics II – boxplots

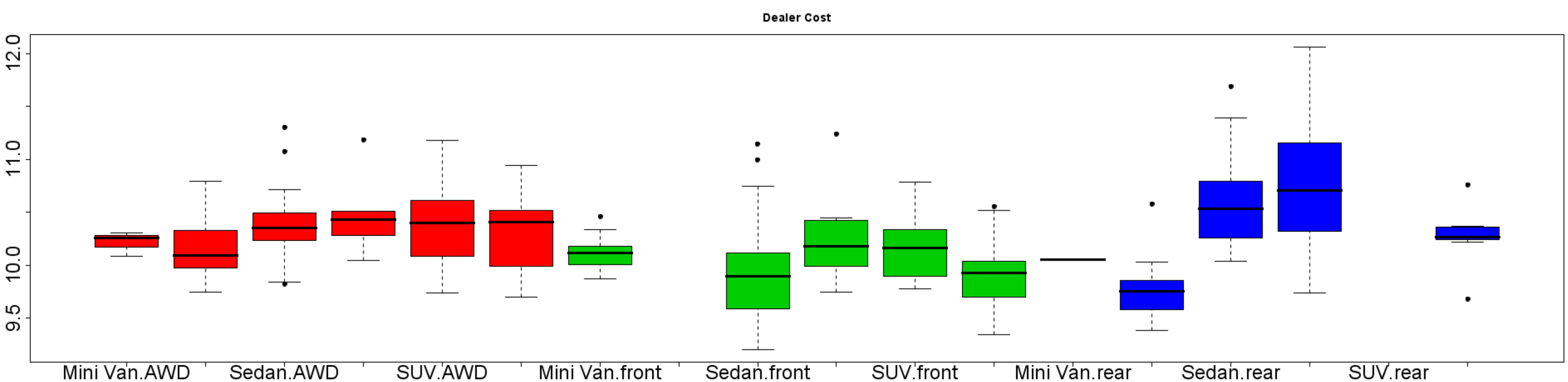
Single boxplots are generated using either the ***plot****(y)* command where *y* is a single numeric variable or via ***boxplot****(x)*.

```
> boxplot(Dealer.Cost, horizontal=T,pch = 19, main ="Dealer Cost")
```



Multiple (parallel) boxplots are generated by ***plot****(y~x)* or ***boxplot****(y~grp),* where *x* and *grp* are factor variables.

```
> boxplot( log(Dealer.Cost)~Type+Drive,pch = 19, main ="Dealer Cost", col = rep(2:4,each=6))
```

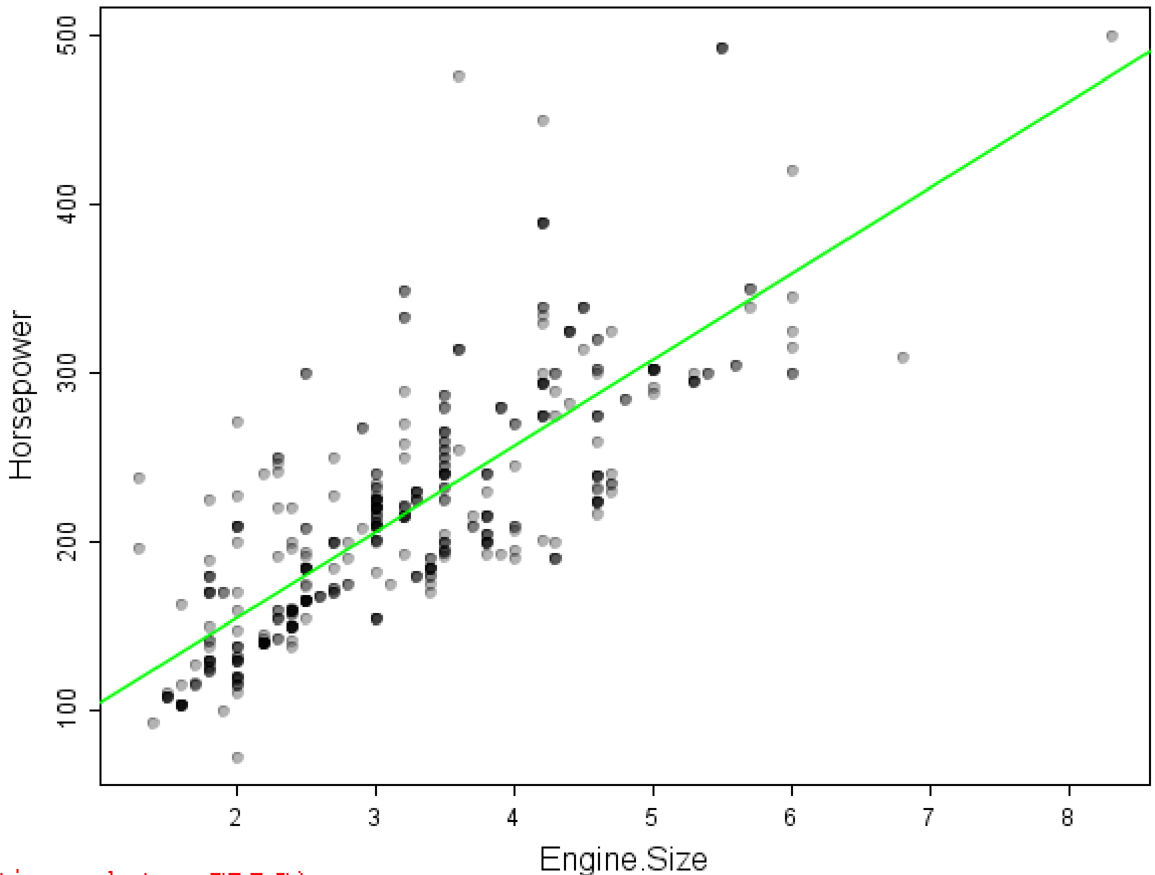


Graphics II - scatterplots

One of most popular amongst the basic graphics is the scatterplot.
In R it is sufficient to call the ***plot****(x,y,..)* function for two numeric variables *x* and *y* (gerneric!)

A simple example:

```
> plot(Engine.Size,Horsepower,
+ col = hsv(0,0,0,alpha=0.3),
+ pch = 19, cex.lab=1.3)
```



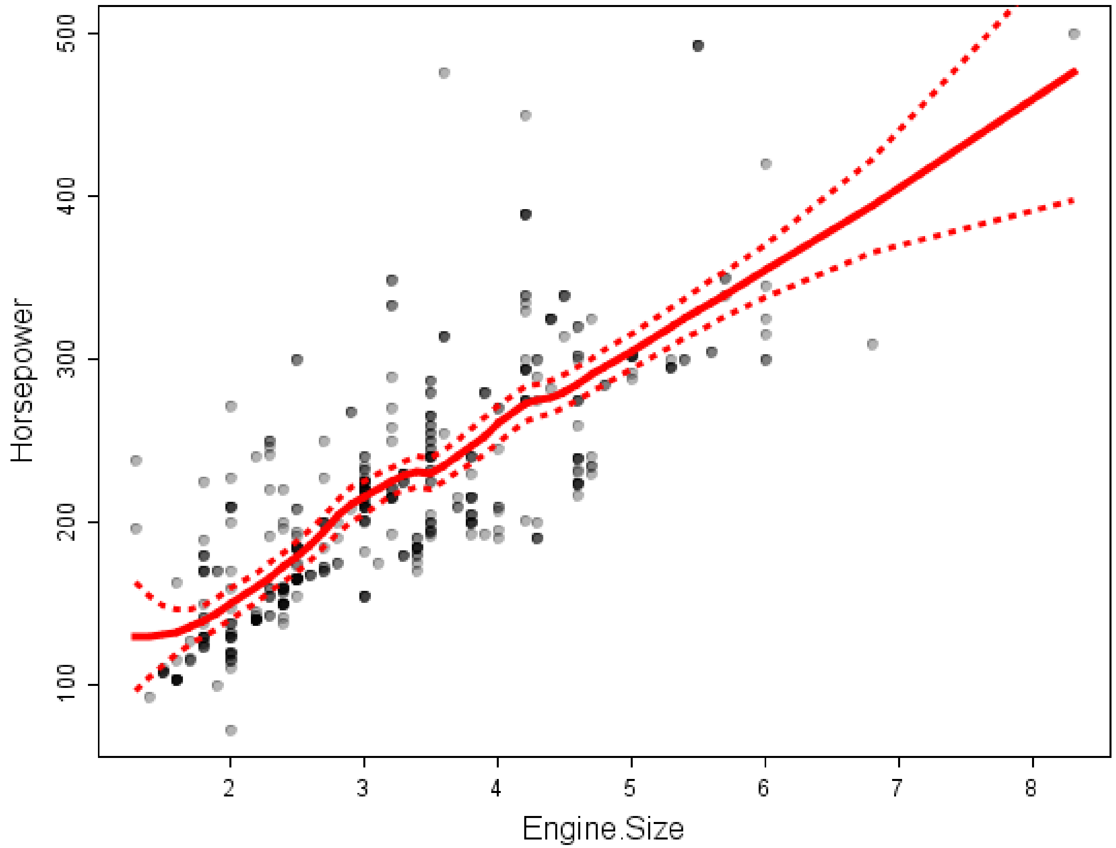
A regression line has been added to the plot.

```
> m1 = lm(Horsepower~Engine.Size,data=CARS)
> abline(m1,col="green", lwd = 1.5)
```

Graphics II - scatterplots

Again the same example:

```
> plot(Engine.Size,Horsepower,
+ col = hsv(0,0,0,alpha=0.3),
+ pch = 19, cex.lab=1.3)
```



In this second example a ***loess smoother*** with confidence bands has been added.

```
> L1 = loess(Horsepower~Engine.Size,data=CARS, span = 0.5)
> p1 = predict(L1, se=TRUE)
> lines(p1$fit            ~sort(Engine.Size),col="red", lwd=4)
> lines(p1$fit-1.96*p1$se~sort(Engine.Size),col="red", lwd=3, lty="dashed")
> lines(p1$fit+1.96*p1$se~sort(Engine.Size),col="red", lwd=3, lty="dashed")
```

Graphics II – time series plots

For time series there exist several different classes in R.

Often the date variable contains strings which have to be converted.
Two examples for date time classes are:

POSIXct / POSIXlt / POSIXt	for dates and time of day
Date	for dates only

```
> as.Date("16.04.2010", format="%d.%m.%Y")
[1] "2010-04-16"
> as.POSIXct("16/04/2010 12:01:53", format="%d/%m/%Y %H:%M:%S", tz="CET")
[1] "2010-04-16 12:01:53 CEST"
> (ss = strptime("16/04/2010 12:01:53", format="%d/%m/%Y %H:%M:%S", tz="CET"))
[1] "2010-04-16 12:01:53 CEST"
> class(ss)
[1] "POSIXt"  "POSIXlt"
```

POSIXt and **POSIXct** start from the years 1900 resp. 1970.
POSIXlt is a list with components *year, month, wday, hour,* … and so on.

The ***plot(x,y,…)*** command is generic and recognizes the class of the variables **x** and **y**.
The same holds for the functions ***lines(), points(), seq(), cut()*** and several others.

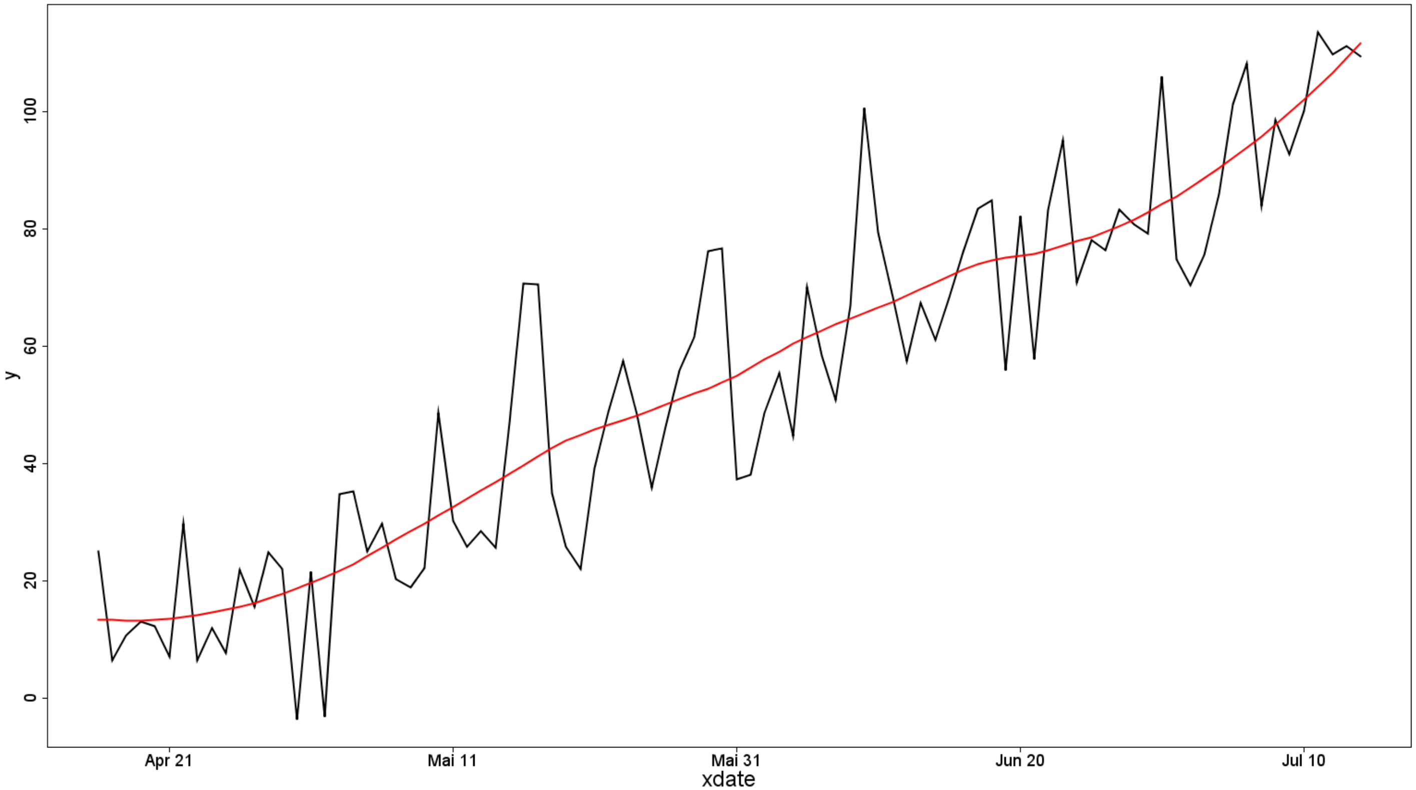
Graphics II – time series plots

A **simulated** time series example with a linear trend and a ***loess*** smoother.

```
> y = a+b*x+e
> xdate = seq(as.Date("2010/04/16"),by="day",along.with=y)
> plot(y~xdate, type="l",lwd=2, cex.axis=1.5)

> L1 = loess(y~x, span=0.5)
> lines(predict(L1)~xdate,lwd=2, col=2)
```

a sequence of class **Date**



Graphics II - Trellis

Trellis graphics are a powerful graphical tool for **exploration** as well as **presentation**.

The idea is to draw a plot for one or more variables grouped by some other variables.

All trellis plots in the package **lattice** are specified by a formula of the form

x ~ y | g1 + g2 or **~ x | g1 + g2**

meaning: plot *x* and *y* for every combination of *g1* and *g2*.

g1 and *g2* are either *factors* or *shingles*.

The most important commands are

xyplot*(x, data, ...)*	conditional scatterplots
barchart*(x, data, ...)*	conditional barcharts
bwplot*(x, data, ...)*	conditional boxplots
splom*(x,data,…)*	lattice scatterplot matrix
parallel*(x,data,…)*	lattice parallel coordinates plot
levelplot*(x,data,…)*	heatmap of a matrix
contourplot*(x,data,…)*	plotting contours in scatterplots
histogram*(x,data,…)*	conditional histograms
densityplot*(x,data,…)*	conditional density estimators

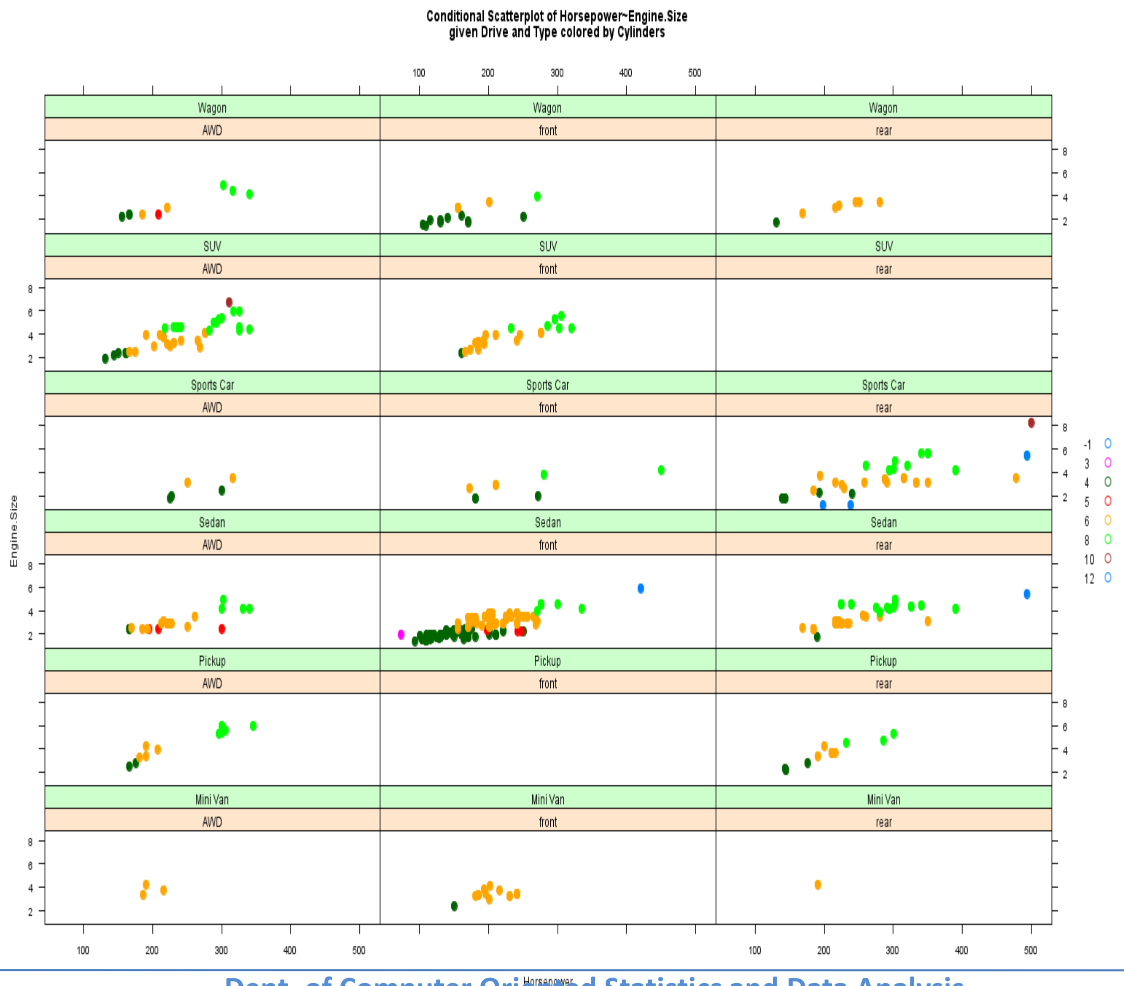
Graphics II - Trellis

Some of the most important parameters:

groups	defines additional grouping variables (factors or shingles). For instance this results in different colors of the groups in ***xyplot()***
horizontal	defines the direction of barcharts or boxplots
stack	defines whether or not to stack barcharts
panel	redefines the plot function manually, e.g. in order to add smoothers
layout	defines the grid for the plot quite similar to **mfcol** or **mfrow**
pch, col, xlab, ylab,…	define some layout options similar to **par**

Graphics II - Trellis

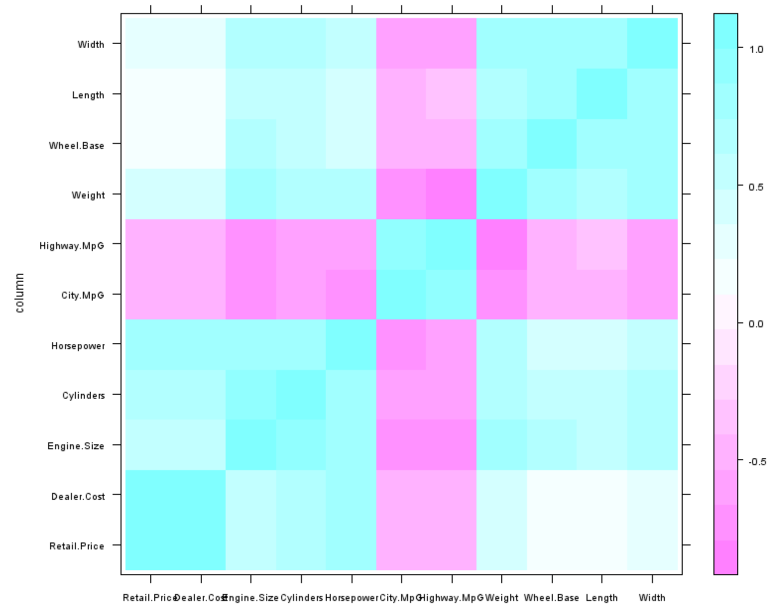
```
> xyplot(Engine.Size~Horsepower|Drive+Type, data=CARS, groups=Cylinders,
+ pch=19, auto.key = list(space="right"),
+ main = "Conditional Scatterplot of Horsepower~Engine.Size
+ given Drive and Type colored by Cylinders")
```



levelplot() visualizes the entries of a data matrix using different colors and alpha-levels.

```
> CM <- cor(CARS[,4:14],use="pairwise.complete")
> round(CM,digits=2)
```

	Retail.Price	Dealer.Cost	Engine.Size	Cylinders	Horsepower	City.MpG	Highway.MpG	Weight	Wheel.Base	Length	Width
Retail.Price	1.00	1.00	0.57	0.63	0.83	-0.46	-0.43	0.45	0.15	0.22	0.33
Dealer.Cost	1.00	1.00	0.56	0.62	0.82	-0.46	-0.42	0.44	0.15	0.21	0.32
Engine.Size	0.57	0.56	1.00	0.90	0.79	-0.70	-0.71	0.81	0.64	0.61	0.72
Cylinders	0.63	0.62	0.90	1.00	0.78	-0.63	-0.62	0.72	0.53	0.54	0.63
Horsepower	0.83	0.82	0.79	0.78	1.00	-0.67	-0.64	0.63	0.39	0.38	0.52
City.MpG	-0.46	-0.46	-0.70	-0.63	-0.67	1.00	0.94	-0.74	-0.50	-0.47	-0.59
Highway.MpG	-0.43	-0.42	-0.71	-0.62	-0.64	0.94	1.00	-0.79	-0.51	-0.39	-0.59
Weight	0.45	0.44	0.81	0.72	0.63	-0.74	-0.79	1.00	0.76	0.67	0.81
Wheel.Base	0.15	0.15	0.64	0.53	0.39	-0.50	-0.51	0.76	1.00	0.87	0.76
Length	0.22	0.21	0.61	0.54	0.38	-0.47	-0.39	0.67	0.87	1.00	0.75
Width	0.33	0.32	0.72	0.63	0.52	-0.59	-0.59	0.81	0.76	0.75	1.00



Graphics II - rmb

The ***rmb***-plot is a multiple-barcharts-like plot for categorical variables.
It is available in the package **extracat.**

It visualizes the conditional relative frequencies of a target variable and the corresponding weights of the explanatory variables.

The default call has the form:

rmb(~V1+V2+V3+TV, dset = mydata)

It can also be used as a generalization of spineplots:

rmb(~V1+V2+V3+TV, dset = mydata, spine = T)

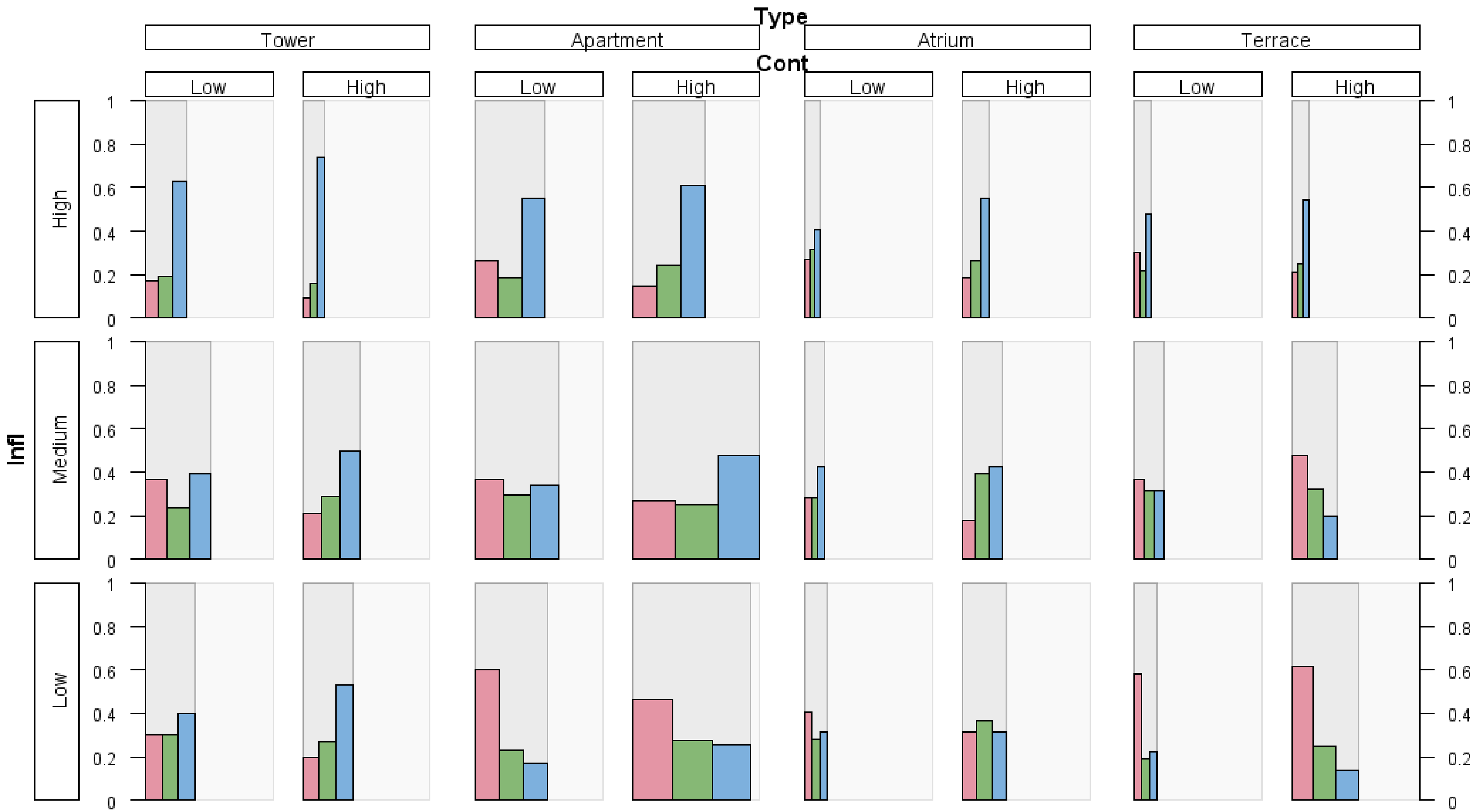
Another interesting option is set by the parameter **eqwidth**:

rmb(~V1+V2+V3+TV, dset = mydata, eqwidth = T)

Graphics II - rmb

An ***rmb***-plot for the Copenhagen housing dataset:

```
> library(colorspace)
> rmb(f = ~Type+Infl+Cont+Sat, dset = housing, colv = rainbow_hcl(3))
```



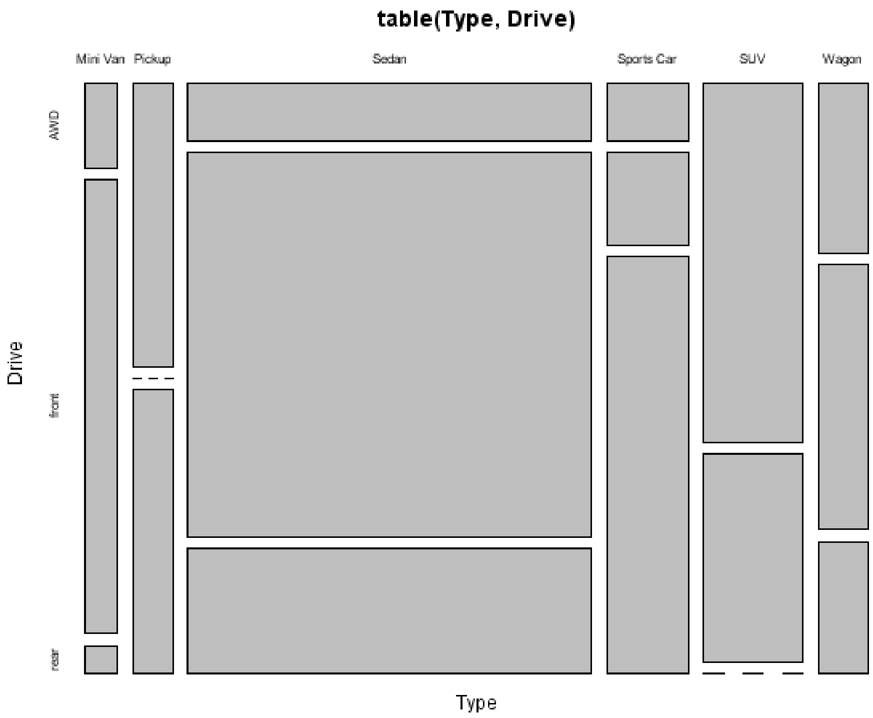
Graphics II - mosaicplots

Mosaicplots are a visualization of contingency tables and thus are generated from them in R. As an alternative formulas can be used.

```
> mosaicplot(table(Type,Drive))
```

leads to the same result as

```
> mosaicplot(Drive~Type)
```



A more sophisticated way to generate mosaicplots is provided by the package **vcd.**

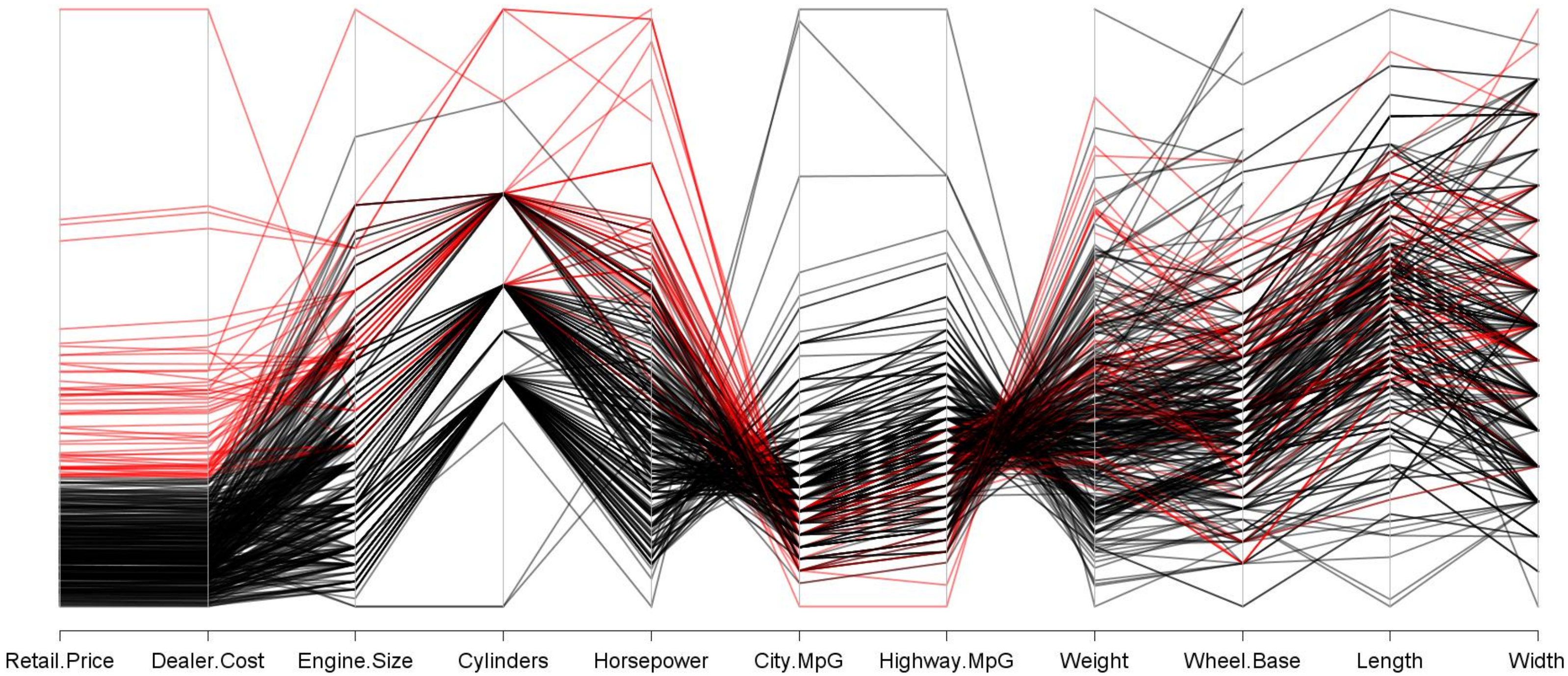
Although it is the recommended choice it needs some experience to use it properly.

Graphics II – PCP

Parallel Coordinates Plots (PCP) display many continuous variables in one graphic.

The standard command to generate them is ***parcoord(x)*** in the **MASS** package.
x has to be a *data.frame* or *matrix* with numeric variables.

```
> cols = rgb(red=c(0,1),green=0,blue=0,alpha=0.5)
> parcoord(CARS[,4:14],col= cols[ (Dealer.Cost > 45000)+1 ],lwd=2)
```



Graphics II – saving images

The simplest way of saving graphics is the file menu.
It offers to save the current graphic either as ***.pdf*** or as ***.eps***

Usually the first option is preferrable.

Another option is to change the graphical output device to a file.

This is done by one of the functions

bmp(file, width, height, ...)
png(file, width, height, ...)
jpeg(file, width, height, ...)
tiff(file, width, height, ...)

Instead of displaying the plot in a graphics device such as JavaGD it will be **saved to the specified file**.

Packages

Which packages to use?

- **packages from task views**
- **recommended packages**
- **packages with vignettes**
- **packages from known authors**
- **packages with a good R help including**
  - **understandable and informative (parameter) descriptions**
  - **meaningful examples**

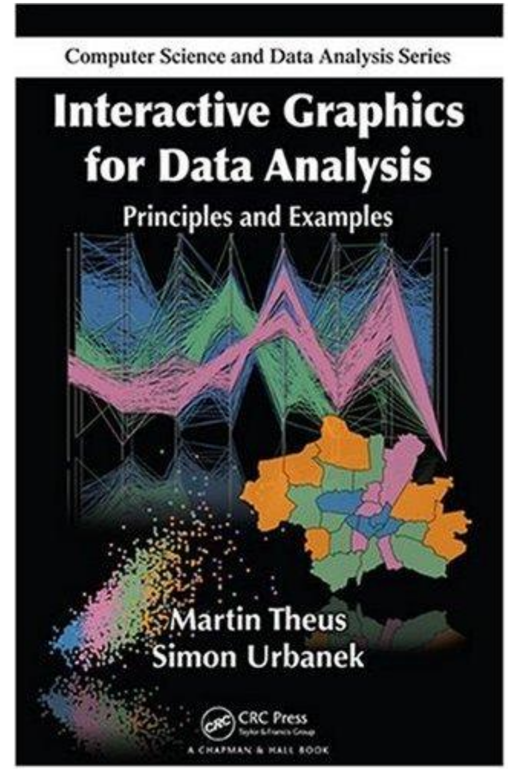
Mondrian

Mondrian and **iplots** provide interactive graphics for explanatroy data analysis.
Currently a new and very fast version of iplots is under development.

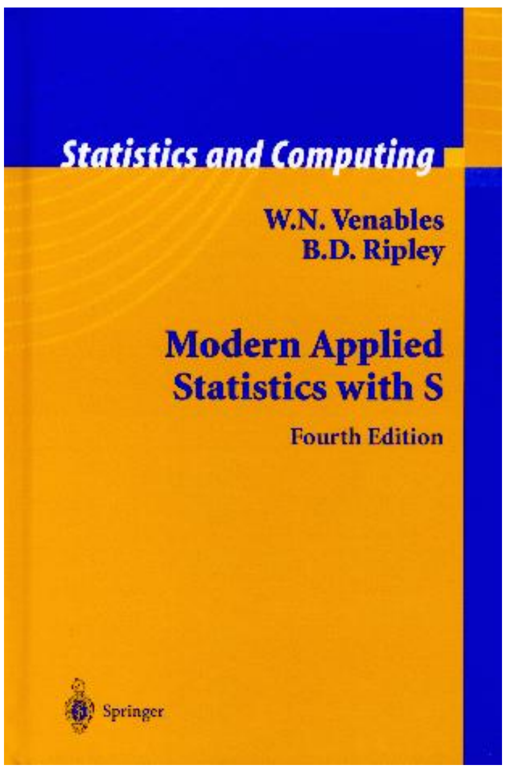
Interactivity means:

Highlighting and Color Brushing	Selections in one plot also affect the others
Zooming	Zooming in plot areas via mouse drag or ceiling censored zooming for MB and fluctuation diagrams
Querying	Additional information about points and cases are available via mouse-over query
Alpha-blending	All point and polyline plots offer interactive alpha blending via arrow keys. The same works for point sizes!
Models and smoothers	log-linear models for residual shadings and smoothers for scatterplots are available (Rserve)
Axis rotations	Plots can be rotated in whole or variable-wise
Orderings	The order of variables in mosaicplots as well as the order of the categories in barcharts can be changed via keyboard and mouse-drag respectively

Further reading



The authors of **iplots** and **Mondrian** have published a very easy-to-understand and well-written book with many examples, graphics and lessons about **Interactive Data Analysis** using Mondrian.



Modern Applied Statistics with S is a standard reference for working with R.