DescTools

A Hardworking Assistant for Descriptive Statistics

<preliminary blueprint version>

by Andri Signorell

Helsana Versicherungen AG, Health Sciences, Zurich
HWZ University of Applied Sciences in Business Administration, Zurich

andri@signorell.net

June, 29h, 2018

R sometimes makes ordinary tasks difficult. Virtually every data analysis project starts with describing data. The first thing to do will often be calculating summary statistics for all variables while listing the occurrence of nonresponse and missing data and producing some kind of graphics. This is a three-click process in SPSS, but regardless of the normality of this task, base R does not contain higher level functions for quickly describing huge datasets (meant regarding the number of variables, not records) adequately in an automated way. Sure, there are facilities like summary (base), describe (Hmisc), stat.desc (library pastecs), but all of them are lacking some functionality or flexibility we would have expected. What we in particular had missed ever since was a combination of numerical and graphical description of data.

R comes with a considerable collection of basic functions for computing summary statistics, including mean, var, median, range and others. But then there are quite a few commonly used functions, which curiously are missing in the stats package, think of e.g. skewness, kurtosis but also the Gini-coefficent, Cohen's Kappa or Somers' delta. This led to a rank growth of libraries implementing just one specific missing thing. There are plenty of "misc"-libraries out there, containing versions of such functions and tests. We often would end up using a dozen libraries, each time using just one single function out of it and suffering huge variety concerning NA-handling, recycling rules and so on.
R has been developed in a university environment. This will be clear at the latest when you find yourself working in a corporate environment, where Word document format is ubiquitous, and you realize that only MS-Office (and no LATEX) is installed on your system (and the IT guys won't give you admin rights). We were forced in this situation to write code for creating our reports in MS-Word. (This works quite well for Windows, but not for Mac unfortunately.)
The origins of "DescTools" go back to a project in which we had to describe a huge dataset under time pressure. Time ran out and we wanted to be faster next time. We then started to gather our newly created functions and put them together. This collection has meanwhile grown to a considerably versatile toolset for descriptive statistics, providing rich univariate and bivariate descriptions of data without expecting the user to say much. There are numerous basic statistic functions and tests included, possibly flexible and enriched with different approaches (if existing). Special attention was paid to providing confidence intervals for all key figures as far as possible.
Recognizing that most problems can be satisfactorily visualized with bar-, scatter- and dotplots, still some more specific plot types are used in special cases and thus included in the package. Some of them are rather new, and some of them are based on types found scattered in the myriads of R packages found out there (partly rewritten to meet the design goals of the package).

This document shows how data description can be accomplished with DescTools.

Users, even expert statisticians, do not always screen the data.
B. D. Ripley, Robust statistics (2004)

1 Introduction

The analyst's sacred duty before beginning any sort of statistical analysis is to take a preliminary look at the data with three main goals in mind: first, to check for errors and anomalies; second, to understand the distribution of each of the variables on its own; and third, to begin to understand the nature and strength of relationships among variables.
Errors should, of course, be corrected, since even a small percentage of erroneous data values can drastically influence the results and might completely invalidate the analysis. Understanding the distribution of the variables, especially the outcomes, is crucial to choosing the appropriate multipredictor regression method. Finally, understanding the nature and strength of relationships is the first step in building a more formal statistical model from which to draw conclusions.
To prevent the analyst to bypass these steps the describing process must be quick and simple. So the principal goal of DescTools is to make data description easier, less costly, less time consuming and less error-prone. One outstanding feature of the package is the combination of numerical results and graphical representation which can mostly be automated and reported to the console, but as well quite easily be exported to a Word Document.

The proper description of data depends on the nature of the measurement. The key distinction for statistical analysis is between numerical and categorical variables. The temperature of the pizza is a numerical variable, while the driver delivering it is categorical. The delivery time is numerical, whereas the area of the customer is categorical.
A secondary but sometimes important distinction within numerical variables is whether the variable can take on a whole continuum or just a discrete set of values. So the temperature would be continuous, while number of pizzas ordered (count) would be discrete.
Categorical variables can be divided in two groups depending on whether the categories are ordered or unordered. So, for example, categories of quality (low, medium, high) would be ordered, while the operator would be unordered. A categorical variable is ordinal if the categories can be logically ordered from smallest to largest in a meaningful; otherwise it is unordered or nominal. Some overlap between types is possible. For example, we may break a numerical variable (such as exact total amount) into ranges or categories. Conversely, we may treat a categorical variable as a numerical score, for example, by assigning values one to three to the ordinal responses Low, Medium, High. Most of the basic analysis methods for numerical scores (e.g., linear regression or t-tests) have interpretations based on average scores. So assigning scores to a categorical variable is effective if average scores are readily interpretable. [3]

A describing procedure must take all these types and properties into account. The function Desc has been designed to describe variables depending on their type with some reasonable statistic measures and an adequate graphic representation. It includes code for describing logical variables, factors (ordered and unordered), integer variables (typically counts), numeric variables, dates and tables and matrices.
Data frames will be split into their variables and the single variable will be described. A formula interface is implemented to easily describe variables in dependence of others.

The output can either be sent to the R-console or as well directly redirected to a MS-Word document. The latter works only in Windows with MS-Office installed, but Mac users can leave the wrd argument away and add a plotit = TRUE argument to have the full results in the console.

Note: For all the following examples in this document, `library(DescTools)` must be declared.

2 Categorical Variables

The first variable to be described is an unordered `factor`. Factors are typically best detailed by a frequency table of their levels. It's not clear how the level order should be set. Especially the given order of levels is rarely helpful in practice. So we choose a pareto rule, the most frequent levels first as the default order for the output table.
Ordered factors would be sorted after their natural order by default. The default order can always be changed by setting the `ord` argument to either `"desc"` (for descending frequency order), `"asc"` (ascending order), `"name"` (alphabetical order) or `"level"` (order of the levels).

```
Desc(d.pizza$driver, plotit=TRUE)
```

Desc

```
  length      n    NAs levels unique  dupes
   1'209  1'204      5      7      7      y


      level freq  perc cumfreq cumperc
1 Carpenter  272  .226     272    .226
2    Carter  234  .194     506    .420
3    Taylor  204  .169     710    .590
4    Hunter  156  .130     866    .719
5    Miller  125  .104     991    .823
6    Farmer  117  .097   1'108    .920
7   Butcher   96  .080   1'204   1.000
```

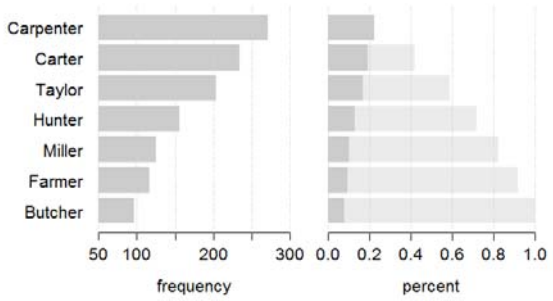


Figure 10.2 Frequency plot of a categorical variable

The argument `plotit` can be set to produce a plot in the same step. It is possible to define this value as an option to save the hassle of having to specify it explicitly each time.

If there are missing values, they will be listed in the first row, together with the length of the vector and the number of levels.

Synopsis

`length`	total number of elements in the vector, NAs are included here
`n`	number of valid cases, NAs, NaNs, Inf etc. are not counted here
`NAs`	number of missing values
`levels`	number of levels
`unique`	number of unique (observed) values. **Note:** This is not necessarily the same number as levels, as there might be empty levels. Thus, the number of levels might be higher than the number of unique values (but not conversely).
`dupes`	y(es) or n(o), reporting if there are any duplicate values in the vector. If "n" (for no) is reported then there are only unique values in the variable. This might typically be the case for identifiers. Factors usually contain duplicates.
`freq`	the number of observations (absolute frequency) of a specific level. The order of a factors frequency table is by default chosen as "absolute frequency-decreasing".
`perc`	the relative frequency of the specific level
`cumfreq`	the cumulative frequencies of the levels
`cumperc`	the same for the percentage values

If the labels of the factor exceed a certain length, they will be truncated. The length where this happens can be controlled with the argument `maxlablen`. The cumulative bars can be blown off with `ecdf=FALSE`. The other arguments follow the meaning of those in the function `barplot`.

Factors sometimes tend to have lots of levels. Listing all of them might not be informative. Thus the frequency table is by default truncated in the case that there are more than a dozen values. This can be avoided by setting the argument `maxrows=Inf`. The same argument can also be used to list either only a defined number of levels. Say we wanted only to have 4 levels included we can set `maxrows=4`. We can also restrict the maximum number of levels by defining the maximum cumulative percentage. If we set e.g. `maxrows=0.7`, then as much levels will be displayed as are needed to just exceed the cumulative percentage of 70%.

The number formats can be controlled by the `DescToolsOptions` "`fmt.abs`" and "`fmt.perc`". These formats define the representation of the counts and of the percentages. (See Chapter `DescToolsOptions` and the helpfile `?Desc.factor` for more details)

The graphical representation consists of two horizontal barplots. The left one is displaying the absolute frequencies with truncated x-axis. The left plot will always display the percentages with fixed x-axis limits set to 0 and 1. The cumulative frequencies can be displayed or be left away. The plot width is adapted to the length of the labels. If the labels get too long, they will be truncated and displayed with ellipsis (...).

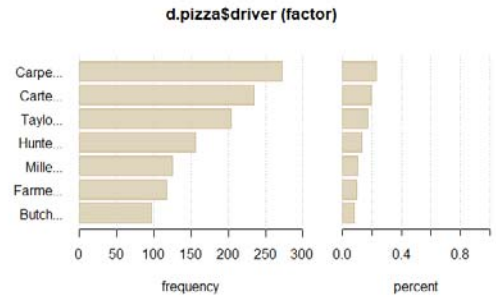
The plot can be customized with several arguments:

```
plot(Desc(d.pizza$driver), main = NULL, maxlablen = 25, type = c("bar", "dot"),
     col = NULL, border = NULL, xlim = NULL, ecdf = TRUE))
```

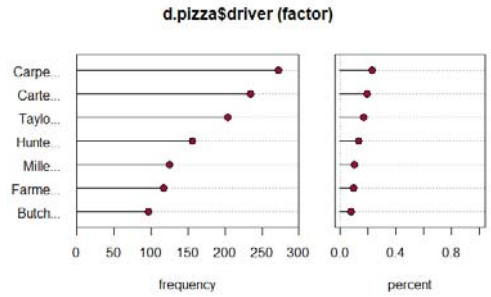
We can e.g. change colors and omit the cumulative distribution as in A):

```
plot(Desc(d.pizza$driver), main=NULL, maxlablen = 5,
     type="bar", col=SetAlpha(hecru, 0.6), border=hecru,
     xlim=c(0, 300), ecdf=FALSE)
```

A)



B)



Or we can choose a "hist"-type plot as in B).

```
plot(Desc(d.pizza$driver), main=NULL, maxlablen = 5,
     type="dot", col=hred, pch=16, xlim=c(0, 300))
```

3 Numerical Variables

3.1 Numeric

The temperature of the delivered pizza is a numeric variable. Numeric variables contain the most information of all variable types and are typically described by a selection of statistical measures for location, variation and shape.

Several features of the output are worth some consideration. The largest and smallest values should be scanned for outlying or incorrect values. In real world erroneous (or awkwardly coded) values are often found at the ends of a variable. Think of e.g. 999 or -256 for NA etc. So the values and their frequencies (numbers in brackets) are reported. 0 plays a special role and is therefore reported individually. In the example below "(2)" means that the value 20.2 can be found twice in the variable.

The mean (or median) and standard deviation (or interquartile range IQR, resp. the median absolute deviation mad) should be assessed as general measures of the location and spread of the data. The quantiles deliver a good overall impression of the distribution. In the current example we note that 90% of the data lie between 26 and 60 degrees and the inner 50% between 42 and 55.

The skewness and kurtosis are usually more easily assessed by graphical means, though their numerical values are included in the output. A large difference between the mean and median is another cue for the skewness. In right-skewed data with a positive value of the skewness, the mean is larger than the median, while in left-skewed data (skewness < 0), the mean is smaller than the median.

```
Desc(d.pizza$temperature, main="", plotit=TRUE)
```

```
  length      n    NAs unique     0s   mean meanSE
   1'209  1'170     39    375      0 47.937  0.291

     .05    .10    .25 median    .75    .90    .95
  26.700 33.290 42.225     50 55.300 58.800 60.500

   range     sd  vcoef    mad    IQR   skew   kurt
  45.500  9.938  0.207  9.192 13.075 -0.842  0.051

lowest : 19.3, 19.4, 20, 20.2 (2), 20.35
highest: 63.8, 64.1, 64.6, 64.7, 64.8
```

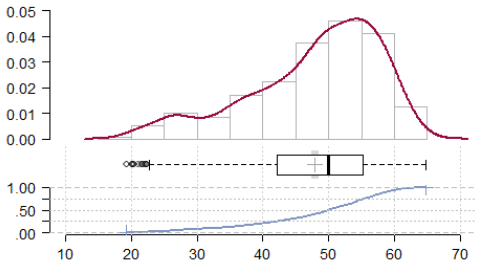


Figure 3.1 Distribution of a numeric variable.

PlotFdist

The plot in figure 3.1 as produced by the function `PlotFdist` combines a histogram with a density plot, a boxplot and the plot of the empirical distribution function (ECDF). The scale for the x-axis is synchronized over all plots. The median can thus be found on the boxplot as also in the ecdf-plot.
The maximum and the minimum value are tagged with a tiny vertical dash upon the ecdf-line. The mean is shown in the boxplot as grey cross, the grey bar is its confidence interval.

Let's enumerate the features in detail. The first measures length, n, NAs, unique have again the same meaning as above. NAs are silently removed from all subsequently calculations.

`0s`	total number of zero values.
`mean`	the arithmetic mean of the vector.
`meanSE`	standard error of the mean, sd(x) / sqrt(n). (See also: function `MeanCI(...)`) This can be used to construct the confidence intervals for the mean, defined as `qt(p = 0.025, df = n-1) * sd(x) / sqrt(n)`.
`.05, .., .95`	quantiles of x, starting with 5%, 10%, 1. quartile, median etc.
`rng`	range of x, `max(x) - min(x)`

`sd`	standard deviation
`vcoef`	variation coefficient, defined as `sd(x) / mean(x)`
`mad`	median absolute deviation
`IQR`	inter quartiles range
`skew`	skewness of x
`kurt`	kurtosis of x
`lowest`	the smallest 5 values. If there are bindings, the frequency of each value will be reported in brackets.
`highest`	same as lowest, but on the other end

Transformations can easily be entered in place.

```
Desc(1/d.pizza$temperature, digits=3, main="")
title(expression(frac(1,x)))
```

```
 length      n    NAs unique     0s   mean meanSE
  1'209  1'170     39    375      0  0.022  0.000

    .05    .10    .25 median    .75    .90    .95
  0.017  0.017  0.018  0.020  0.024  0.030  0.037

  range     sd  vcoef    mad    IQR   skew   kurt
  0.036  0.006  0.289  0.004  0.006  2.027  4.244

lowest : 0.015, 0.015, 0.015, 0.016, 0.016
highest: 0.049, 0.050 (2), 0.050, 0.052, 0.052
```

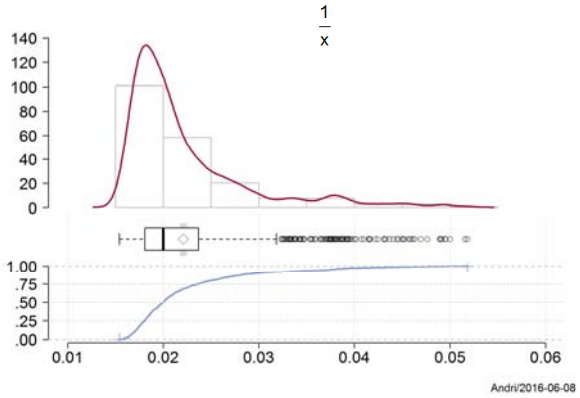


Figure 3.2 Distribution of a numeric variable.

There are several approaches commonly used for graphical comparing the variable's distribution to a reference distribution. The two most seen are firstly superposing the reference density curve over the variable's histogram and the second using a Q-Q-plot. A Q-Q-plot is used to compare the shapes of distributions, providing a graphical view of how properties such as location, scale, and skewness are similar or different in the two distributions.

```
z <- LinScale(z, newlow=0, newhigh = 32)[,1]

PlotFdist(z, args.curve = list(expr="dchisq(x, df=5)", col="darkgreen"),
          args.boxplot=NA, args.ecdf=NA)

legend(x="topright", legend=c("kernel density", expression(chi["df=5"]^2-distribution)),
       fill=c(getOption("col1", hred), "darkgreen"), text.width = 5)
```

LinScale

We get

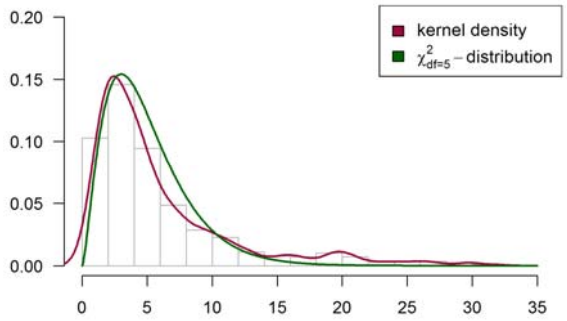


Figure 3.3 Overlay of fitted χ^2-function.

This makes it clear, that this is not the best way to decide, whether the red curve follows our hypothesized distribution or not. Where does randomness begin and where does it end?

PlotQQ

The better approach is to use a QQ-plot, which by the way solves the x-axis scaling problem we had in the overlay solution. The function `PlotQQ` is a wrapper for plotting QQ-plots with other than normal distributions.
A `qqline` is inserted on which the points are likely to lie (approximately) if the two distributions being compared are similar.
It sometimes might be hard to judge, if the points are (too) far away from the `qqline` or not.
An idea to check the general variability is to use simulated sets with the desired distribution. If our points exceed the confidence intervals, something is likely to be wrong.
In our example everything's fine, of course, as we sampled from the tested distribution.

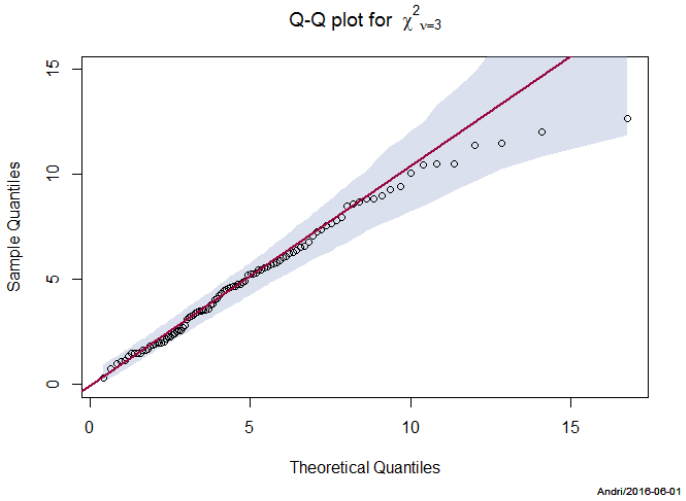


Figure 3.4 QQ plot for a χ^2-distributed variable.

```
set.seed(159)
z <- rchisq(100, df=5)

PlotQQ(z, function(p) qchisq(p, df=5), type="n", main=NA, args.qqline = NA)

x <- qchisq(ppoints(z), df=5)
y <- replicate(1000, sort(rchisq(100, df=5)))
ci <- apply(y, 1, quantile, c(0.025,0.975))

DrawBand(x = c(x, rev(x)), y = c(ci[1,], rev(ci[2,])), col=SetAlpha(hblue, 0.3))
PlotQQ(z, function(p) qchisq(p, df=5), add=TRUE,
       args.qqline=list(col=hred,lwd=2, probs=c(0.1, 0.6)))

title(main=expression("Q-Q plot for" ~~ {chi^2}[nu == 5]))
```

What do the tests say about ozone being gamma distributed?

```
AndersonDarlingTest(na.omit(ozone), "pgamma", shape = m^2/v, scale = v/m)

##         Anderson-Darling test of goodness-of-fit
##         Null hypothesis: Gamma distribution
##         with parameters shape = 1.6310, scale = 25.8300
##
## data:  na.omit(ozone)
## An = 0.66365, p-value = 0.5896
```

The observation seems compatible with the hypothesis.
Let's superpose the model distribution curve to both, the histogram and the cumulative distribution function.

```
ozone <- airquality$Ozone; m <- mean(ozone, na.rm = TRUE); v <- var(ozone, na.rm = TRUE)

PlotFdist(ozone, args.hist = list(breaks=15),
```

```
  args.curve = list(expr="dgamma(x, shape = m^2/v, scale = v/m)", col=hecru),
  args.curve.ecdf = list(expr="pgamma(x, shape = m^2/v, scale = v/m)", col=hecru),
  na.rm = TRUE, main = "Airquality - Ozone")

legend(x="topright",
       legend=c(expression(plain("gamma:  ") * Gamma * " " * bgroup("(", k * " = " *
           over(bar(x)^2, s^2) * " , " * theta * plain(" = ") * over(s^2, bar(x)), ")") ),
                "kernel density"),
       fill=c(hecru, getOption("col1", hred)), text.width = 0.25)
```

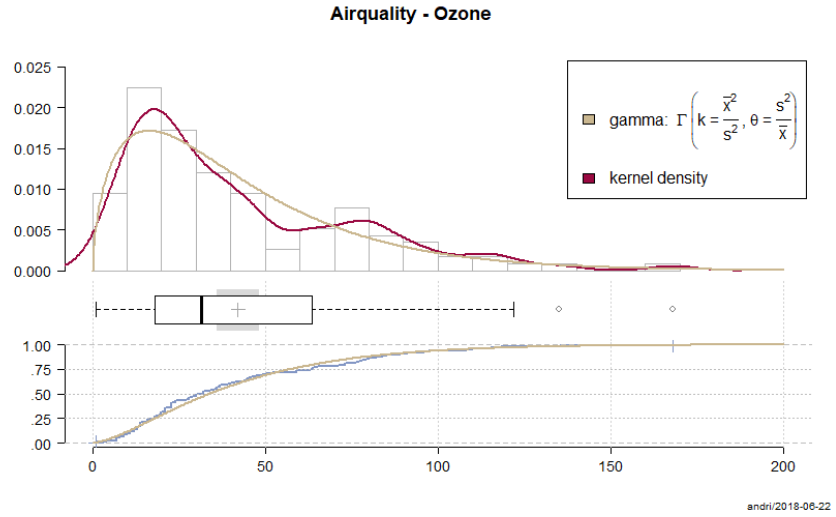


Figure 3.5 Compare empirical distribution with a gamma distribution.

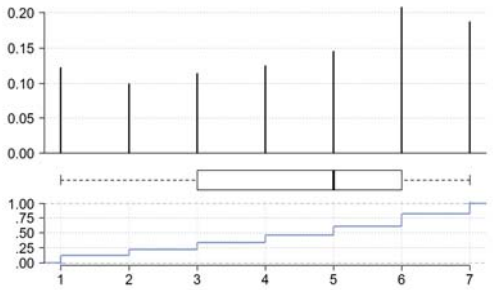
3.2 Count data (discrete) or numeric data with few unique values

The next variable is a discrete variable, whose nature is somewhat between numeric and factors as far as descriptive measures are concerned. In fact, if there are only just a few unique values, then the factor representation (frequencies) might be more appropriate than the numeric description (with densities etc.). We draw the line between factor and numeric representation at a dozen of unique values in x. Beyond that number, the numeric description will be reported and for fewer values the factor representation will be used.

In the numerical results scheme the extreme values will be replaced by a full frequency representation with absolute values and percentages. The number of rows can be controlled via `maxrows` in the same manner as with factor levels (see above).

```
Desc(d.pizza$weekday, plotit=TRUE)
```

```
length      n    NAs  unique     0s   mean  meanSE
 1'209  1'177     32       7      0   4.44    0.06

   .05    .10    .25  median    .75    .90    .95
  1.00   1.00   3.00    5.00   6.00   7.00   7.00

 range     sd  vcoef     mad    IQR   skew   kurt
  6.00   2.02   0.45    2.97   3.00  -0.34  -1.17


   level  freq   perc  cumfreq  cumperc
1      1   144  12.2%      144    12.2%
2      2   117   9.9%      261    22.2%
3      3   134  11.4%      395    33.6%
4      4   147  12.5%      542    46.0%
5      5   171  14.5%      713    60.6%
6      6   244  20.7%      957    81.3%
```

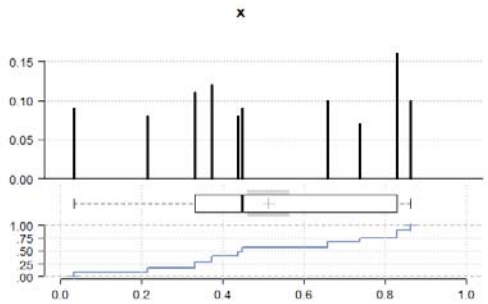


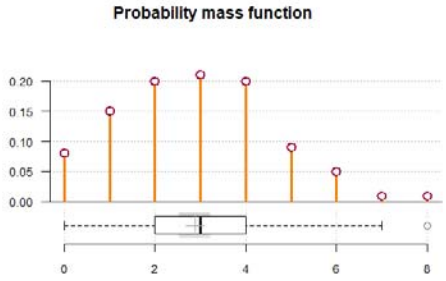
```
7      7   220  18.7%    1'177   100.0%
```

Figure 3.6 Distribution of a discrete variable.

The above assessment also applies to numeric variables that contain only a few different values. If there are only a handful of unique values, then a description by means of a histogram and a density curve is not adequate. The density curve would start oscillating and the bins in the histograms would lose their continuous nature.
Therefore, we change the graphic representation in such cases from a histogram to a histogram like h-type plot leaving the density curve off.

```
set.seed(1984)
x <- sample(runif(10), 100, replace = TRUE)
PlotFdist(x, args.hist=list(type="mass"))
```





The same representation also makes sense for displaying some Poisson distributed values:

```
x <- sample(runif(10), 100, replace = TRUE)
PlotFdist(x, args.hist=list(type="mass"))

pp <- rpois(n = 100, lambda = 3)
PlotFdist(pp, args.hist = list(type="mass", pch=21, col=horange,
                               cex.pch=2.5, col.pch=hred, lwd=3, bg.pch="white"),
          args.boxplot = NULL, args.ecdf = NA, main="Probability mass function")
```

4 Logical values

Dichotomous variables do not have real dense (univariate) information. The variable `wine_ordered` for example contains only two values, 0 and 1. Still it is usually interesting to know, how many NAs there are, besides the frequencies. In DescTools the individual frequencies are reported together with a confidence interval, calculated by `BinomCI` using the option `"Wilson"`, which provides a good default in most cases.

```
Desc(d.pizza$wine_ordered, plotit=TRUE)
```

```
  length      n    NAs unique
   1'209  1'197     12      2

  freq  perc lci.95 uci.95¹
0 1010  .844   .822   .863
1  187  .156   .137   .178

¹ 95%-CI Wilson
```

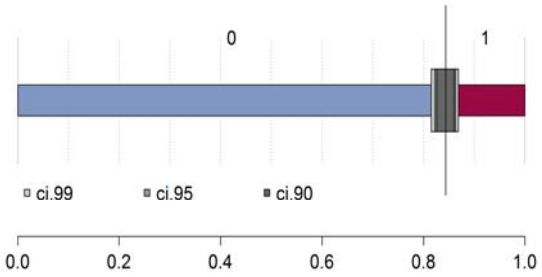


Figure 4.1 Distribution of a numeric variable.

This is basically a univariate horizontal stacked `barplot`, with confidence intervals on the confidence levels of 0.90, 0.95 and 0.99. The vertical line denominates the point estimator.

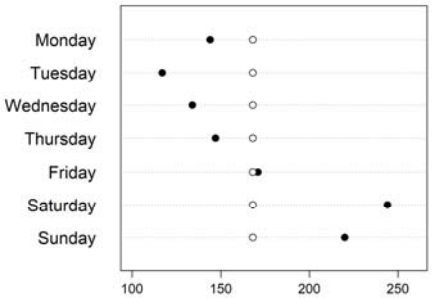
5 Time variables

5.1 Dates

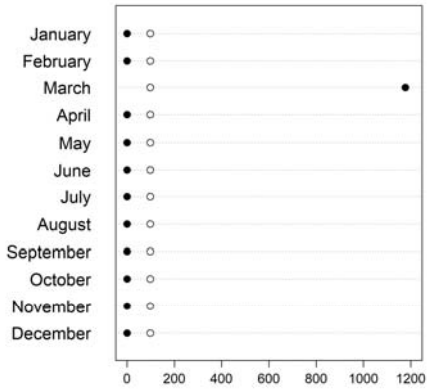
A date variable is rarely described in a univariate context regarding its calendar properties. Nevertheless, the distribution of weekdays or months can sometimes provide us with surprising insights into seasonal structures of the variables. We would normally choose a description similar to numeric values, supplemented by an analysis of the weekday and month for grasping anomalies concerning extreme, invalid or missing values.

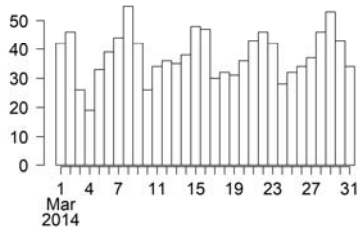
```
Desc(d.pizza$date, plotit=TRUE)
```

```
  length      n    NAs unique
   1'209  1'177     32     31
          97.4%   2.6%

lowest : 2014-03-01 (42), 2014-03-02 (46), 2014-03-03 (26), 2014-03-04 (19)
highest: 2014-03-28 (46), 2014-03-29 (53), 2014-03-30 (43), 2014-03-31 (34)

Weekday:

Pearson's Chi-squared test (1-dim uniform):
  X-squared = 78.879, df = 6, p-value = 6.09e-15

       level  freq  perc  cumfreq  cumperc
1     Monday   144  .122      144     .122
2    Tuesday   117  .099      261     .222
3  Wednesday   134  .114      395     .336
4   Thursday   147  .125      542     .460
5     Friday   171  .145      713     .606
6   Saturday   244  .207      957     .813
7     Sunday   220  .187    1'177    1.000

Months:

Pearson's Chi-squared test (1-dim uniform):
  X-squared = 12947, df = 11, p-value < 2.2e-16

        level   freq   perc  cumfreq  cumperc
1     January      0   .000        0     .000
2    February      0   .000        0     .000
3       March  1'177  1.000    1'177    1.000
4       April      0   .000    1'177    1.000
5         May      0   .000    1'177    1.000
6        June      0   .000    1'177    1.000
7        July      0   .000    1'177    1.000
8      August      0   .000    1'177    1.000
9   September      0   .000    1'177    1.000
10    October      0   .000    1'177    1.000
```



```
11   November      0   .000    1'177    1.000
12   December      0   .000    1'177    1.000

By days :

         level  freq  perc  cumfreq  cumperc
1   2014-03-01    42  .036       42     .036
2   2014-03-02    46  .039       88     .075
3   2014-03-03    26  .022      114     .097
4   2014-03-04    19  .016      133     .113
5   2014-03-05    33  .028      166     .141
6   2014-03-06    39  .033      205     .174
7   2014-03-07    44  .037      249     .212
8   2014-03-08    55  .047      304     .258
9   2014-03-09    42  .036      346     .294
10  2014-03-10    26  .022      372     .316
11  2014-03-11    34  .029      406     .345
12  2014-03-12    36  .031      442     .376
13  2014-03-13    35  .030      477     .405
14  2014-03-14    38  .032      515     .438
15  2014-03-15    48  .041      563     .478
16  2014-03-16    47  .040      610     .518
17  2014-03-17    30  .025      640     .544
18  2014-03-18    32  .027      672     .571
19  2014-03-19    31  .026      703     .597
20  2014-03-20    36  .031      739     .628
21  2014-03-21    43  .037      782     .664
22  2014-03-22    46  .039      828     .703
23  2014-03-23    42  .036      870     .739
24  2014-03-24    28  .024      898     .763
25  2014-03-25    32  .027      930     .790
26  2014-03-26    34  .029      964     .819
27  2014-03-27    37  .031    1'001     .850
28  2014-03-28    46  .039    1'047     .890
29  2014-03-29    53  .045    1'100     .935
30  2014-03-30    43  .037    1'143     .971
31  2014-03-31    34  .029    1'177    1.000
```

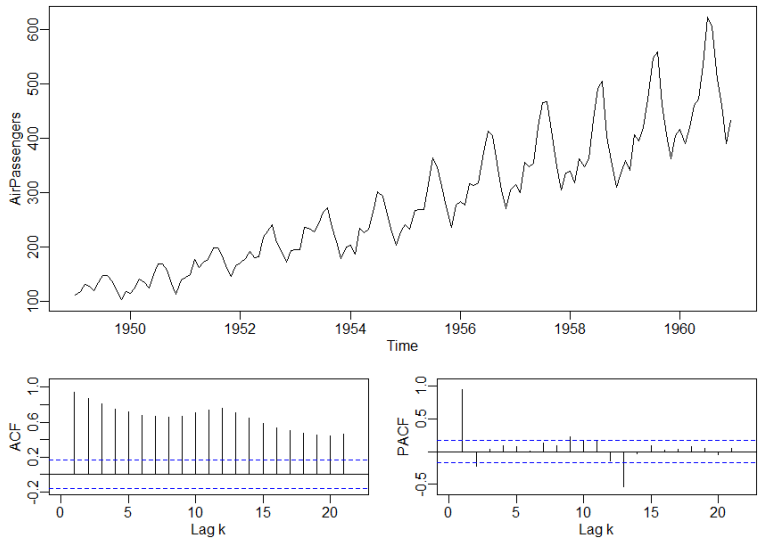




5.2 Timeseries ACF-plot

For a serious time series analysis, we need the representation of the temporal course to be able to recognize the internal structures and the autocorrelative connections. For this task base R already contains a comprehensive set of tools for this purpose. We here contribute a more condensed variation of basic plots. The function PlotACF() produces a combined plot of a time series and its autocorrelation and partial autocorrelation, which is often used in introductory course for time-series.

```
PlotACF(AirPassengers)
```



6 data.frames

6.1 Overview

For a data frame we would normally like to have a quick overview over the contained variables.

```
Desc(d.pizza)
```

This will produce the following table containing the types of the variables, as well as the levels for the factors and potential labels. Missing values are reported for each variable in absolute and relative frequencies. The overview alone can be produced with Abstract(d.pizza).

data.frame: 1209 obs. of 16 variables

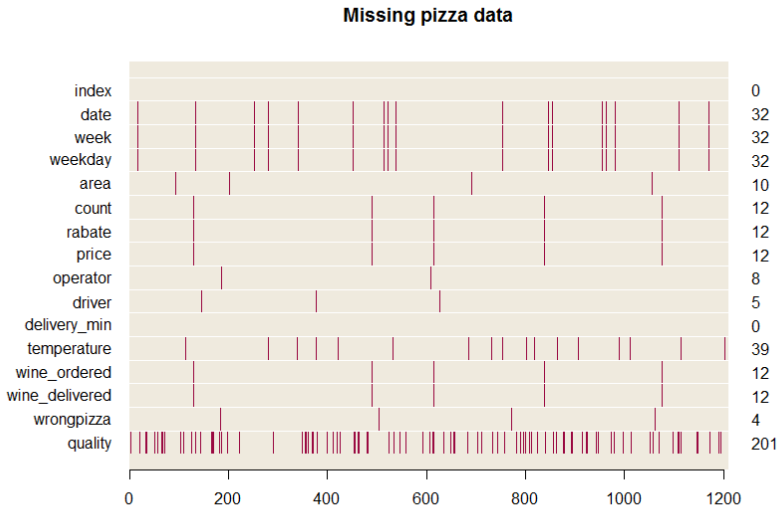
Nr	ColName	Class	NAs	Levels	Label
1	index	integer	.		-
2	date	Date	32 (2.6%)		-
3	week	numeric	32 (2.6%)		-
4	weekday	numeric	32 (2.6%)		-
5	area	factor	10 (0.8%)	(3): 1-Brent, 2-Camden, 3-Westminster	-
6	count	integer	12 (1.0%)		-
7	rabate	logical	12 (1.0%)		-
8	price	numeric	12 (1.0%)		-
9	operator	factor	8 (0.7%)	(3): 1-Allanah, 2-Maria, 3-Rhonda	-
10	driver	factor	5 (0.4%)	(7): 1-Butcher, 2-Carpenter, 3-Carter, 4-Farmer, 5-Hunter, ...	-
11	delivery_min	numeric	.		-
12	temperature	numeric	39 (3.2%)		This is the temperature in degrees Celsius measured at the time when the pizza is delivered to the

					client.
13	wine_ordered	integer	12 (1.0%)		-
14	wine_delivered	integer	12 (1.0%)		-
15	wrongpizza	logical	4 (0.3%)		-
16	quality	ordered, factor	201 (16.6%)	(3): 1-low, 2-medium, 3-high	-

Then each variable is described according to the type of its class.

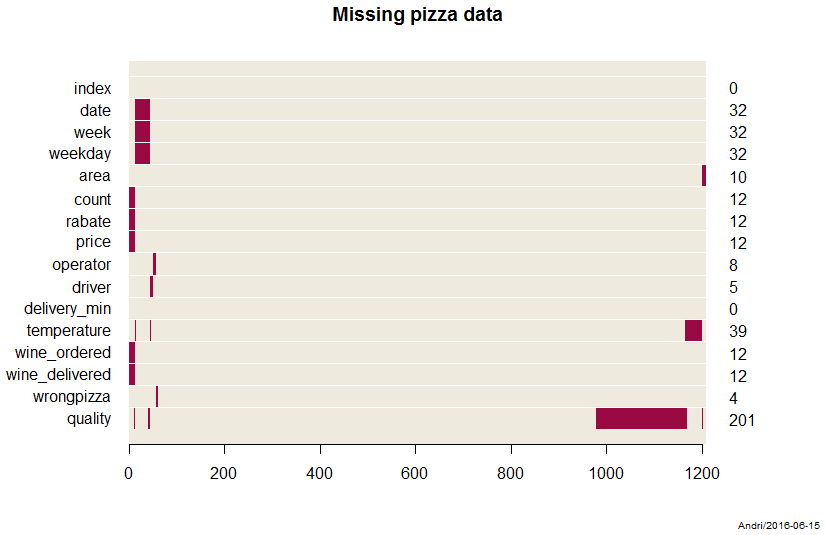
6.2 Missing data

An interesting idea for creating a visual representation of missing data was brought to my attention by Henk Harmsen. The following plot symbolizes each missing value with a vertical line. The x-axis represents the index of the record. On the right side are the numbers of missings noted.



The missing values can be clustered such as to display several areas of missing values. This can be helpful for detecting dependencies or patterns within the missings.

```
PlotMiss(d.pizza, main="Missing pizza data", clust = TRUE)
```



7 Pairwise Numeric ~ Categorical

`Desc` implements a formula interface allowing to define bivariate descriptions straight forward. The formula supports the ., meaning all remaining variables in the given data frame. This allows in particular the description of a response variable by a freely definable set of further explanatory variables. So we could describe temperature pairwise by all remaining variables defined in the data argument.

```
Desc(temperature ~ ., data=d.pizza[, c("area","driver","delivery_min","temperature")])
```

This can as well be reversed in the sense that the dot is defined as response variable and so all the variables will be plotted against one predictor variable.

```
Desc(. ~ temperature, data=d.pizza[, c("area","driver","delivery_min","temperature")])
```

7.1 Boxplot and Designplot

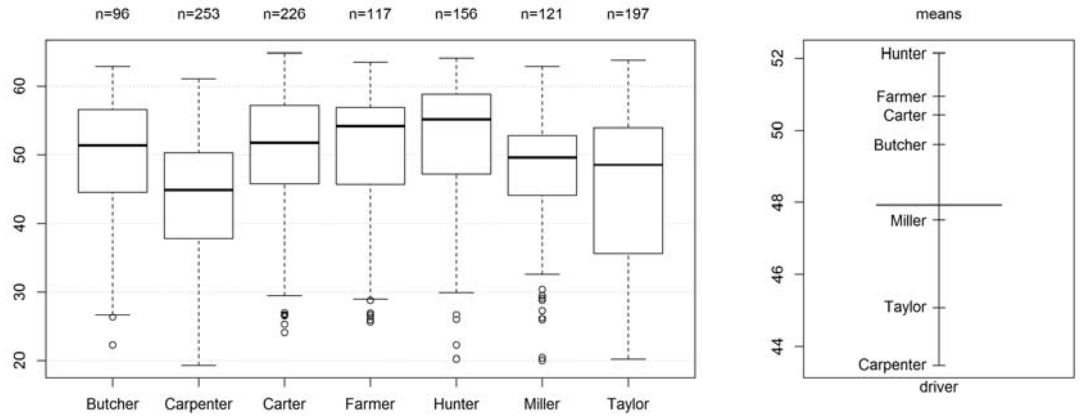
A numeric variable vs. a categorical is best described by group wise measures. Here the valid pairs are reported first. Missing values in the single groups are documented in the results table and missing values on the grouping factor are mentioned with a warning at the end of the table, if existing at all.

```
Desc(temperature ~ driver, d.pizza, digits=1, plotit=TRUE)
```

```
Summary:
n pairs: 1'209, valid: 1'166 (96%), missings: 43 (4%), groups: 7
```

	Butcher	Carpenter	Carter	Farmer	Hunter	Miller	Taylor
mean	49.6	43.5[1]	50.4	50.9	52.1[2]	47.5	45.1
median	51.4	44.8[1]	51.8	54.1	55.1[2]	49.6	48.5
sd	8.8	9.4	8.5	9.0	8.9	8.9	11.4
IQR	12.0	12.5	11.3	11.2	11.6	8.8	18.4
n	96	253	226	117	156	121	197
np	0.082	0.217	0.194	0.100	0.134	0.104	0.169
NAs	0	19	8	0	0	4	7
0s	0	0	0	0	0	0	0

```
¹ min, ² max

Kruskal-Wallis rank sum test:
  Kruskal-Wallis chi-squared = 141.9349, df = 6, p-value < 2.2e-16
Warning:
  Grouping variable contains 5 NAs (0.414%).
```

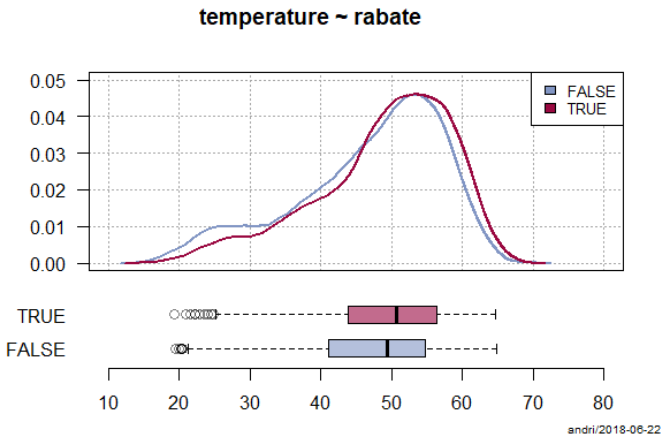


As default graphical representation a boxplot is chosen combined with a means-plot as it is often used in ANOVA analysis.

In case where we have only a few levels in the explanatory variable, we might prefer a presentation by a density plot, combined with a boxplot. The density allows a better insight in the internal distribution of the variable.
Here we also change the test being used to compare the response between the two levels (t.test() instead of kruskal.test()).

```
(z <- Desc(temperature ~ rabate, d.pizza, test=t.test, digits=1, plotit=FALSE))
plot(z, type="dens")
```

```
temperature ~ rabate

Summary:
n pairs: 1'209, valid: 1'158 (95.8%), missings: 51 (4.2%), groups: 2

        FALSE   TRUE
mean     46.9   49.0
median   49.4   50.7
sd       10.2    9.5
IQR      13.6   12.5
n         580    578
np      50.1%  49.9%
NAs        21     18
0s          0      0

Welch Two Sample t-test:
  t = -3.7159, df = 1149.2,
  p-value = 0.0002122
Warning:
  Grouping variable contains
  12 NAs (0.993%).
```

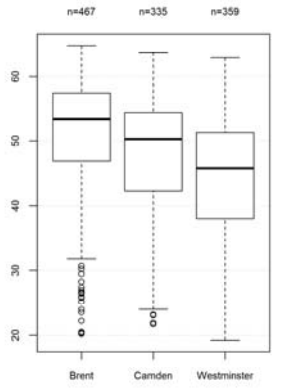


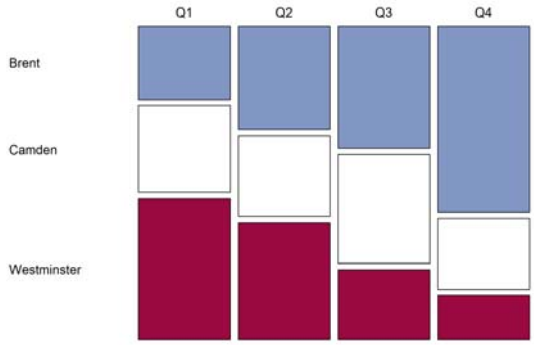
So we learn that pizzas given a rebate are also significantly warmer, when delivered to the client.

8 Pairwise Categorical ~ Numeric

No, it's not the same as numeric ~ categorical. The design is such, that the response variable is categorical and the predictor numeric. With a model one would set up a multinomial regression (or logistic in the case of 2 categories).

```
Desc(area ~ temperature, data=d.pizza, digits=1, wrd=wrd)

Summary:
n pairs: 1'209, valid: 1'161 (96%), missings: 48 (4%), groups: 3

               Brent       Camden  Westminster
mean           51.1²         47.4        44.3¹
median         53.4²         50.3        45.9¹
sd              8.7          10.1         9.8
IQR            10.5          12.2        13.2
n               467           335         359
np            0.402         0.289       0.309
NAs               7             9          22
0s                0             0           0

¹ min, ² max

Kruskal-Wallis rank sum test:
  Kruskal-Wallis chi-squared = 115.83, df = 2, p-value < 2.2e-16
Warning:
  Grouping variable contains 10 NAs (0.827%).


Proportions of area in the quantiles of temperature:

                Q1    Q2    Q3    Q4
  Brent       0.244 0.345 0.405 0.618
  Camden      0.289 0.266 0.363 0.236
  Westminster 0.467 0.389 0.232 0.146
```



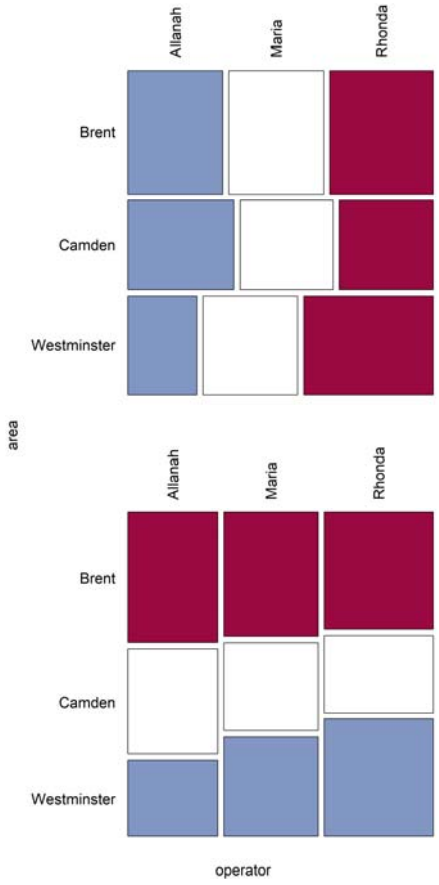


9 Pairwise Categorical ~ Categorical

Two categorical variables are described by a contingency table and a mosaicplot.

```
Summary:
n: 1'191, rows: 3, columns: 3

Pearson's Chi-squared test:
  X-squared = 17.905, df = 4, p-value = 0.001288
Likelihood Ratio:
  X-squared = 18.099, df = 4, p-value = 0.001181
Mantel-Haenszel Chi-squared:
  X-squared = 8.6654, df = 1, p-value = 0.003243

Phi-Coefficient        0.123
Contingency Coeff.     0.122
Cramer's V             0.087
```

area	operator	Allanah	Maria	Rhonda	Sum
Brent	freq	153	153	167	473
	perc	12.8%	12.8%	14.0%	39.7%
	p.row	32.3%	32.3%	35.3%	.
	p.col	41.9%	39.9%	37.7%	.
Camden	freq	123	108	109	340
	perc	10.3%	9.1%	9.2%	28.5%
	p.row	36.2%	31.8%	32.1%	.
	p.col	33.7%	28.2%	24.6%	.
Westminster	freq	89	122	167	378
	perc	7.5%	10.2%	14.0%	31.7%
	p.row	23.5%	32.3%	44.2%	.
	p.col	24.4%	31.9%	37.7%	.
Sum	freq	365	383	443	1'191
	perc	30.6%	32.2%	37.2%	100.0%
	p.row	.	.	.	.
	p.col	.	.	.	.



Further information about tables are shown in the vignette Tables.

10 Pairwise Numeric ~ Numeric

10.1 Scatterplot

Two numerical variables have no obvious standard description as their relationship can have manifold forms. Thus, we're going to report only the simple correlation coefficients (Pearson, Spearman and Kendall) and a hopefully helpful scatterplot.

The variables are plotted as xy-scatterplots with interchanging mutual dependency, supplemented with either a LOESS or a spline smoother.

```
Desc(temperature ~ delivery_min, d.pizza, plotit=TRUE)
```

```
Summary:
n pairs: 1'209, valid: 1'170 (97%), missings: 39 (3%)


Pearson corr. : -0.575
Spearman corr.: -0.573
Kendall corr. : -0.422
```

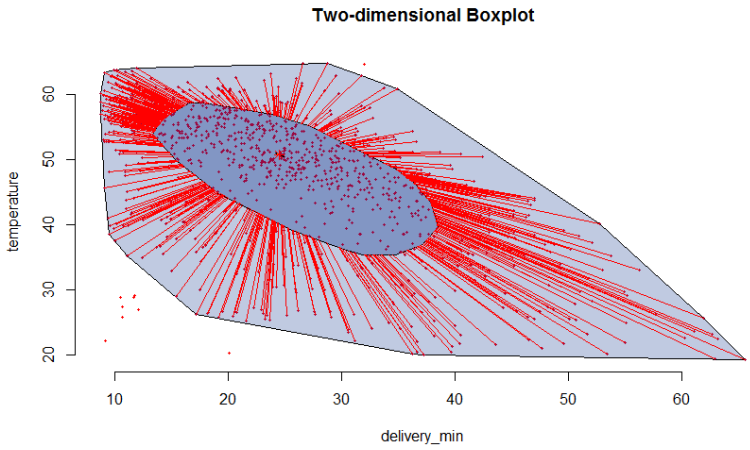
Scatterplots for two numeric variables:

Figure 10.2 Scatterplot of temperature and delivery time.

10.2 Boxplot in 2 dimensions: PlotBag

If we would want to have more information on the distribution of the two values, we can alternatively try a bagplot. This function transposes the boxplot idea in the 2-dimensional space. The points are outliers, the lightblue area is the area within the fences in a normal boxplot and the darkblue area is the inner quartile range.
The median is plotted as orange point in the middle.
This code is taken verbatim from Peter Wolf's aplpack package.

```
d.frm <- d.pizza[complete.cases(d.pizza[,c("temperature","delivery_min")]),]

PlotBag(x=d.frm$delivery_min, y=d.frm$temperature, xlab="delivery_min",
        ylab="temperature", main="Two-dimensional Boxplot")
```

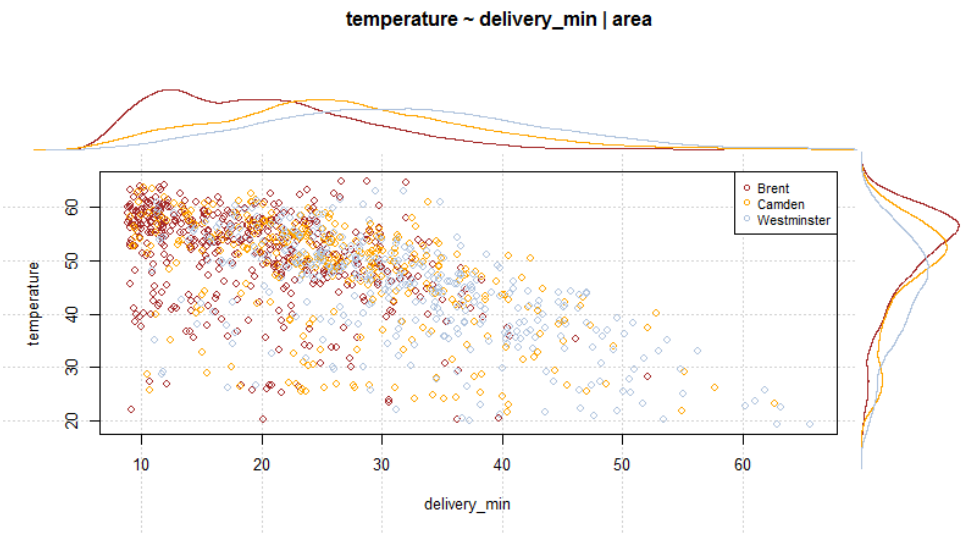


10.3 PlotMarDens

This plot shows a scatterplot of two numerical variables temperature and delivery_time, by area. On the margins the density curves of the specific variable are plotted, also stratified by area.

```
PlotMarDens(y=d.pizza$temperature, x=d.pizza$delivery_min, grp=d.pizza$area,
```

```
            xlab="delivery_min", ylab="temperature",
            col=c("brown","orange","lightsteelblue"), panel.first=grid(),
            main="temperature ~ delivery_min | area"  )
```



11 Table One

Create a table summarizing continuous, categorical and dichotomous variables, optionally stratified by one or more variables, while performing adequate statistical tests. The function will make use of the format definitions in the DescToolsOptions().

```
ToWrd(TOne(x=d.pizza[, c("temperature","delivery_min","driver","wine_ordered")],
           grp=d.pizza$quality),
      wrd=GetNewWrd())
```

will produce the following table:

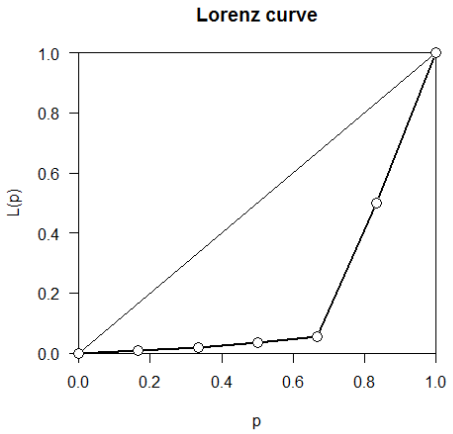
var	total	low	medium	high	
n	1'008	156 (15.5%)	356 (35.3%)	496 (49.2%)	
temperature	47.9 (9.9)	32.9 (7.8)	45.6 (7.4)	53.6 (6.5)	*** [1]
delivery_min	25.7 (10.8)	33.9 (11.7)	26.5 (10.1)	22.6 (9.5)	*** [1]
driver					*** [3]
Butcher	79 (8.0%)	10 (6.5%)	36 (10.1%)	33 (6.7%)	
Carpenter	225 (22.6%)	59 (38.1%)	90 (25.4%)	76 (15.4%)	
Carter	196 (19.4%)	11 (7.1%)	72 (20.3%)	113 (22.9%)	
Farmer	94 (9.7%)	10 (6.5%)	26 (7.3%)	58 (11.7%)	
Hunter	130 (13.0%)	8 (5.2%)	43 (12.1%)	79 (16.0%)	
Miller	109 (10.4%)	16 (10.3%)	35 (9.9%)	58 (11.7%)	
Taylor	171 (16.9%)	41 (26.5%)	53 (14.9%)	77 (15.6%)	
wine_ordered (= 1)	161 (16.1%)	32 (20.8%)	63 (17.9%)	66 (13.4%)	. [3]

[1]) Kruskal-Wallis test, [2]) Fisher exact test, [3]) Chi-Square test

12 Concentration

Lorenz-curves can be found in other libraries. This implementation starts with that from the library `ineq`, adding some value by calculating confidence intervals for the Gini coefficient.

```
x <- c(10, 10, 20, 20, 500, 560)

lc <- Lc(x)
plot(lc)
points(lc$p, lc$L, cex=1.5, pch=21, bg="white", col="black", xpd=TRUE)

Gini(x)
Gini(x, unbiased = FALSE)

Gini(x, conf.level = 0.95)
```




```
Gini(x)
[1] 0.7535714

Gini(x, unbiased = FALSE)
[1] 0.6279762

Gini(x, conf.level=0.95)
     gini    lwr.ci    upr.ci
0.7535714 0.2000000 0.8967742
```

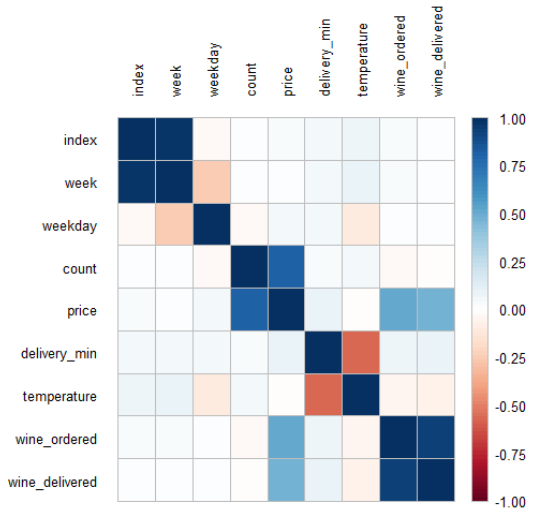
13 Multivariate graphical description

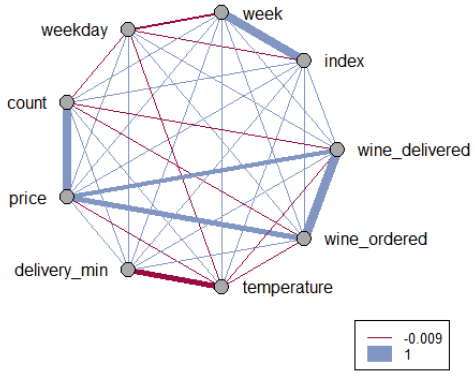
13.1 Correlation plot

These functions produce a graphical display of a correlation matrix. In the classic matrix representation, the cells of the matrix can be shaded or coloured to show the correlation value. In the right circular representation, the correlations are coded in the line width of the connecting lines. Red means a negative correlation, blue a positive one.

```
par(mfrow=c(1,2))
m <- cor(d.pizza[,which(sapply(d.pizza, is.numeric))], use="pairwise.complete.obs")

PlotCorr(m, col=PalDescTools("RedWhiteBlue1", 100), border="grey",
        args.colorlegend=list(labels=Format(seq(1,-1,-.25), 2), frame="grey"))
```

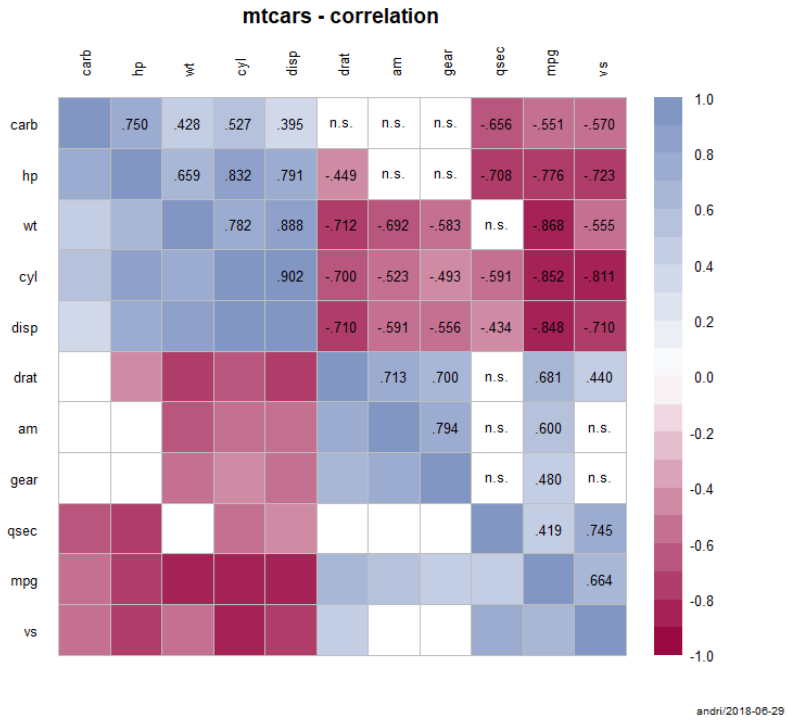
```
PlotWeb(m, col=c(hred, hblue))
```





We can combine some features and produce a plot which displays only the significant correlations, includes the correlation coefficient for significant correlations and groups variables with similar results together.

```
m <- cor(mtcars)

idx <- order.dendrogram(as.dendrogram(
  hclust(dist(m), method = "mcquitty")
))

# now let's get rid of all non significant correlations
p <- PairApply(mtcars,  function(x, y) cor.test(x, y)$p.value, symmetric=TRUE)
# ok, got all the p-values, now replace > 0.05 with NAs
m[p > 0.05] <- NA

PlotCorr(m[idx, idx], main="mtcars - correlation")

x <- matrix(rep(1:ncol(m),each=ncol(m)), ncol=ncol(m))
y <- matrix(rep(ncol(m):1,ncol(m)), ncol=ncol(m))
txt <- Format(m[idx, idx], d=3, leading = "drop", na.form = "n.s.")
idx <- upper.tri(matrix(x, ncol=ncol(m)), diag=FALSE)
text(x=x[idx], y=y[idx], label=txt[idx], cex=0.8, xpd=TRUE)
```

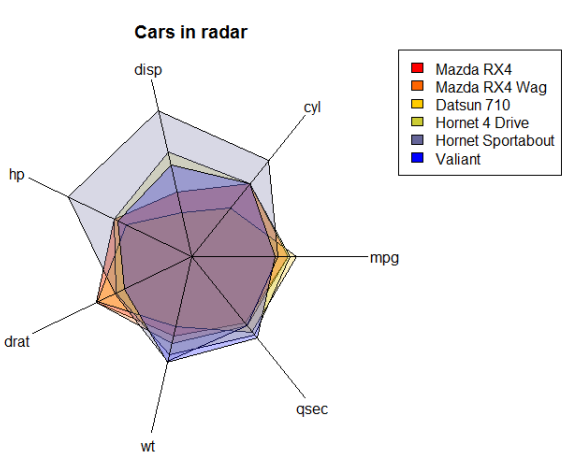


13.2 PlotPolar (Radarplot)

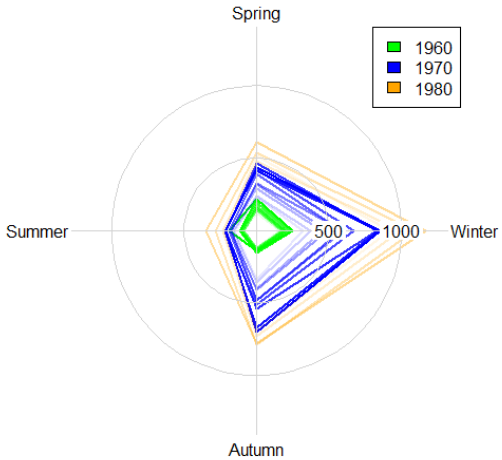
This function produces a polar plot but can also be used to draw radarplots or spiderplots.

A)

```
d.car <- scale(mtcars[1:6,1:7], center=FALSE)

# let's have a palette with thransparent colors
cols <- SetAlpha(colorRampPalette(c("red","yellow","blue"), space = "rgb")(6), 0.25)

PlotPolar(d.car, type="l", fill=cols, main="Cars in radar")
PolarGrid(nr=NA, ntheta=ncol(d.car), alabels=colnames(d.car), lty="solid", col="black")
legend(x=2, y=2, legend=rownames(d.car), fill=SetAlpha(cols, NA))
```



A)

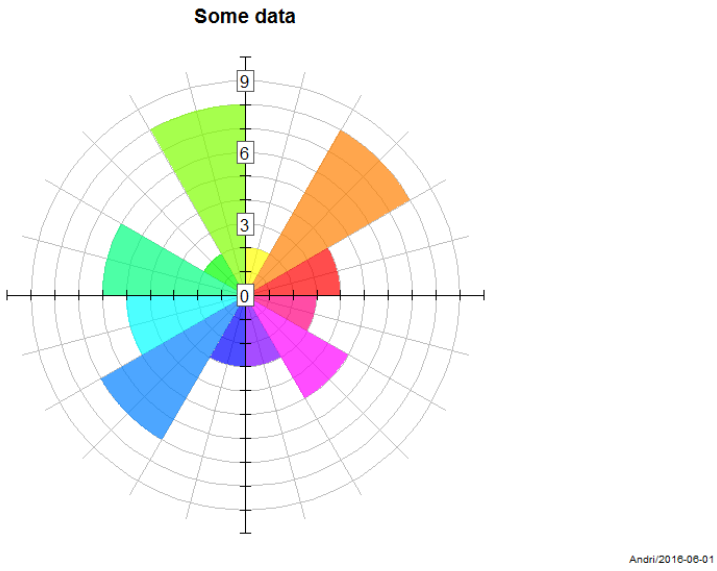


B)

B)

```
m <- matrix(UKgas, ncol=4, byrow=TRUE)

cols <- c(SetAlpha(rep("green", 10), seq(0,1,0.1)),
          SetAlpha(rep("blue", 10), seq(0,1,0.1)),
          SetAlpha(rep("orange", 10), seq(0,1,0.1)))

PlotPolar(r=m, type="l", col=cols, lwd=2 )
PolarGrid(ntheta=4, alabels=c("Winter","Spring","Summer","Autumn"), lty="solid")

legend(x="topright", legend=c(1960,1970,1980), fill=c("green","blue","orange"))
```

A barplot in polar coordinates can be produced by means of the function DrawAnnulusSector.

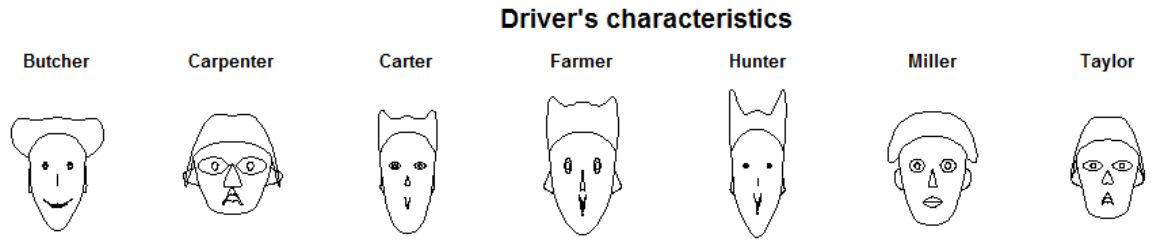



```
x <- c(4,8,2,8,2,6,5,7,3,3,5,3)
theta <- (0:12) * pi / 6
PlotPolar(x, type = "n", main="Some data")
PolarGrid(nr = 0:9, ntheta = 24, col="grey", lty=1, rlabels = NA, alabels = NA)
DrawAnnulusSector(x=0, y=0, radius.in=0, radius.out=x,
                  angle.beg = theta[-length(theta)], angle.end = theta[-1],
                  col=SetAlpha(rainbow(12), 0.7), border=NA)

segments(x0 = -10:10, y0 = -.2, y1=0.2)
segments(x0=-10, x1=10, y0 = 0)
segments(y0 = -10:10, x0 = -.2, x1=0.2)
segments(y0=-10, y1=10, x0 = 0)

BoxedText(x=0, y=c(0,3,6,9), labels = c(0,3,6,9), xpad = .3, ypad=.3, border="grey35")
```

13.3 PlotFaces

A nice idea for the concrete representation of your customer's profile is to produce a Chernoff faces plot. The rows of a data matrix represent cases and the columns the variables.

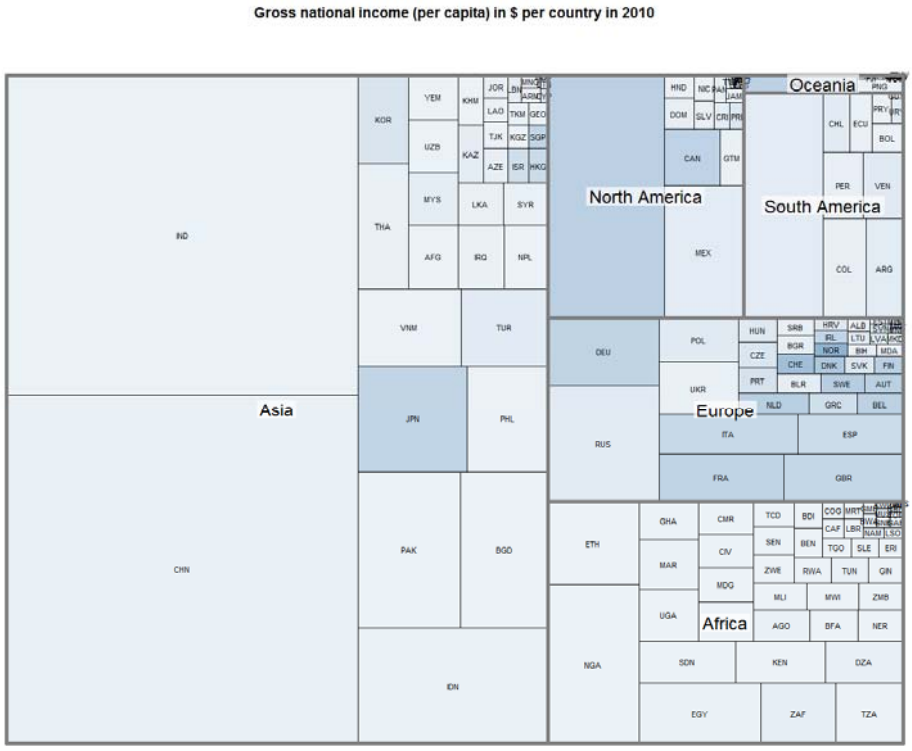
```
m <- data.frame( lapply(
d.pizza[,c("temperature","price","delivery_min","wine_ordered","weekday")]
                        , tapply, d.pizza$driver, mean, na.rm=TRUE))

PlotFaces(m, ncol=7, nrow=1, main="Driver's characteristics")
```



13.4 PlotTreemap

This function produces a treemap.

```
# get some data
data(GNI2010, package="treemap")
gn <- GNI2010[,c("iso3","population","continent","GNI")]
gn <- gn[gn$GNI!=0,]

# define a color
gn$col1 <- SetAlpha("steelblue", LinScale(gn$GNI, newlow=0.1, newhigh=0.6))


b <- PlotTreemap(x=gn$population, grp=gn$continent, col=gn$col1, labels=gn$iso3,
                 main="Gross national income (per capita) in $ per country in 2010",
                 labels.grp=NA, cex=0.7)

# get the midpoints
mid <- do.call(rbind, lapply(lapply(b, "[", 1), data.frame))

# and write the continents' text
DrawBoxedText(x=mid$grp.x, y=mid$grp.y, labels=rownames(mid), cex=1.5, bold=TRUE,
              border=NA, col=SetAlpha("white",0.7) )
```

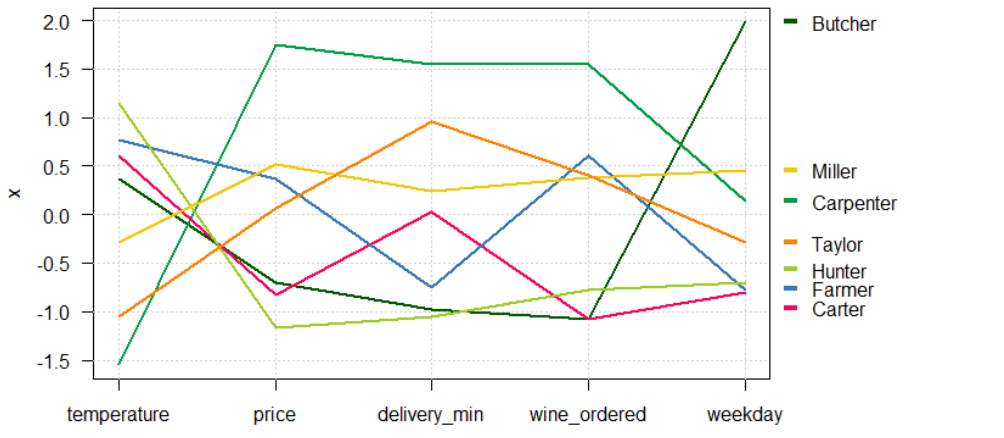


14 Supplements to base R plots

14.1 Lineplots

There are many flavours of line plots, most (all?) of which can be handled by the function `matplot()`. We would generally desist from defining own functions, that only set suitable arguments for another already existing function, as we fear we would run into a forest of new functions, loosing overview. And yet, the parametrization of `matplot()` can be such a traumatic experience that we moved away from the principle and devised our own procedure, extended by a nifty and well readable legend at the right side.

```
PlotLinesA(t(ms), col=PalTibco(), lwd=2)
```

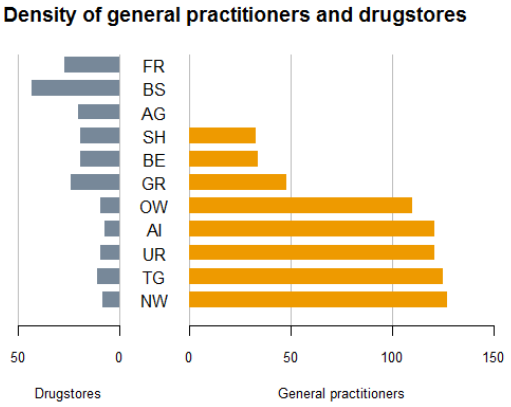


Andri/2016-04-27

14.2 PlotPyramid

A special kind of horizontal barplot is a “pyramid plot”, where the bars are plotted back to back. This is sometimes needed, when your boss has specific and strict ideas how his presentation should look like.

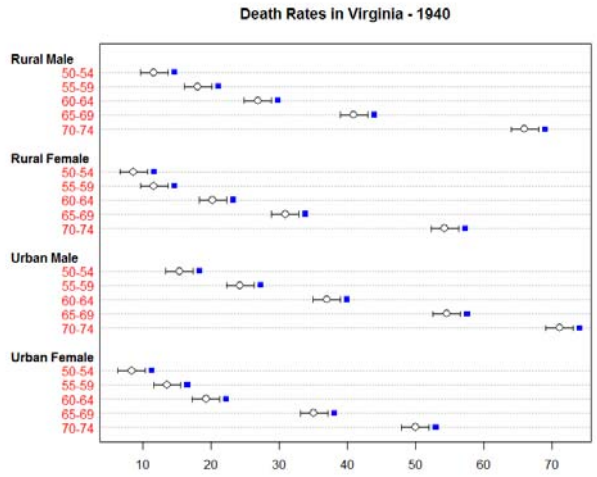
```
d.sda <- data.frame(
  kt_x  = c("NW","TG","UR","AI","OW","GR","BE","SH","AG","BS","FR"),
  apo_n = c(  8,  11,   9,   7,   9,  24,  19,  19,  20,  43,  27 ),
  sda_n = c(127, 125, 121, 121, 110,  48,  34,  33,   0,   0,   0 ))

PlotPyramid(lx=d.sda[,c("apo_n","sda_n")], ylab=d.sda$kt_x,
            col=c("lightslategray", "orange2"), border = NA, ylab.x=0, xlim=c(-110,250),
            gapwidth = NULL, cex.lab = 0.8, cex.axis=0.8, xaxt = TRUE,
            lxlab="Drugstores", rxlab="General practitioners",
            main="Density of general practitioners and drugstores",
            space=0.5, args.grid=list(lty=1))
```



14.3 PlotDot

The base function dotchart has been improved but still has some potential for extensions. Especially an add argument is sometimes useful and returning the y-coordinates for the points would allow to add elements.
PlotDot implements these extensions and allows adding error bars. This is interesting, as the calculation of the x-limits should be done with respect to the bars and not only to the points.

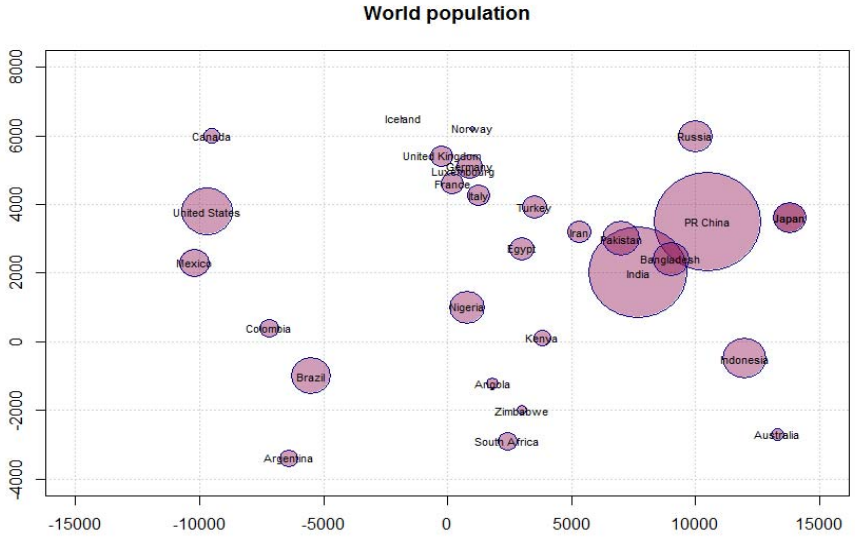
```
# add some error bars
PlotDot(VADeaths, main="Death Rates in Virginia - 1940", col="red", pch=NA,
        args.errbars = list(from=VADeaths-2, to=VADeaths+2, mid=VADeaths,
        pch=21, cex=1.4))

# add some other values
PlotDot(VADeaths+3, pch=15, col="blue", add=TRUE, labels=NA)
```



14.4 PlotBubble

Bubbles can actually easily be produced with the standard plot function. This function here helps you defining appropriate axis limits.

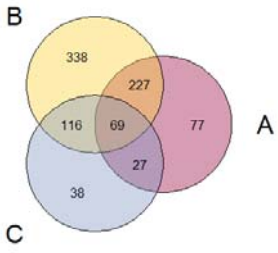
```
PlotBubble(d.world$x, d.world$y, area=d.world$pop/90, col=SetAlpha("deeppink4",0.4),
border="darkblue",
           xlab="", ylab="", panel.first=grid(), main="World population")
text(d.world$x, d.world$y, labels=d.world$country, cex=0.7, adj=0.5)
```



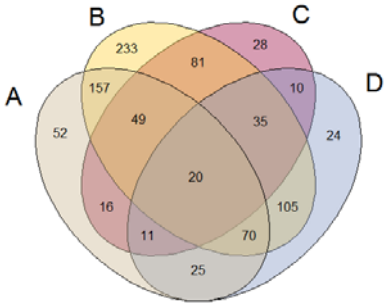
14.5 Venn plots

Now and then one might want to plot a Venn diagram. This function does this for up to 5 datasets using the simple proposed geometric representations.
(For more than 5 datasets the Venn representation loses its simplicity and other plot types become more adequate.)

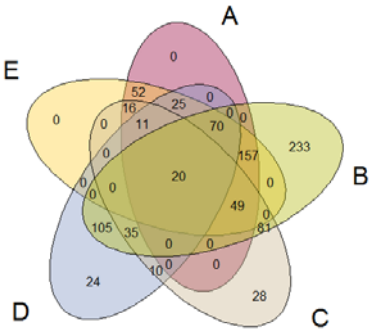
```
example(PlotVenn)
PlotVenn(x=x[1:3], col=SetAlpha(c(PalHelsana()[c(1,3,6)]), 0.4))
PlotVenn(x=x[1:4], col=SetAlpha(c(PalHelsana()[c(1,3,6,4)]), 0.4))
PlotVenn(x=x[1:5], col=SetAlpha(c(PalHelsana()[c(1,3,6,4,7)]), 0.4))
```



3 sets



4 sets



5 sets

14.6 Areaplot

Areaplots have a high "ink factor"[1], say they use much ink to display the information and are therefore rarely the best way of representing data. But again, when your boss wants it this way, here's a function to produce it easily.

```
t.oil <- t(matrix(c(13.3,11.4, 9.7,10.6,12.7,11.0,10.6,13.5,
                    5.3, 3.6, 5.8, 8.4, 9.1,14.8,10.6, 9.6,
                    4.9, 3.1, 3.0, 6.0,12.2, 7.1, 7.3,10.0,
                    2.1, 2.6, 2.7, 3.5, 4.7, 5.0, 4.4, 4.3), nrow=4, byrow=TRUE,
   dimnames = list(c("ExxonMobil","BP","Shell","Eni"),
                     c("1998","1999","2000","2001","2002","2003","2004","2005"))))
t(t.oil)

par(mfrow=c(1,2), mar=c(5,4,5,5))
col <- SetAlpha(PalHelsana(), 0.7)
PlotArea(t.oil, col = col, las = 1, frame.plot=FALSE)
mtext(side=4, text=colnames(t.oil), las=1,
      at=Midx(tail(t.oil, 1)[,], incl.zero=TRUE, cumulate=TRUE))

PlotArea(prop.table(t.oil, 1), col = col, las = 1, frame.plot=FALSE)
```

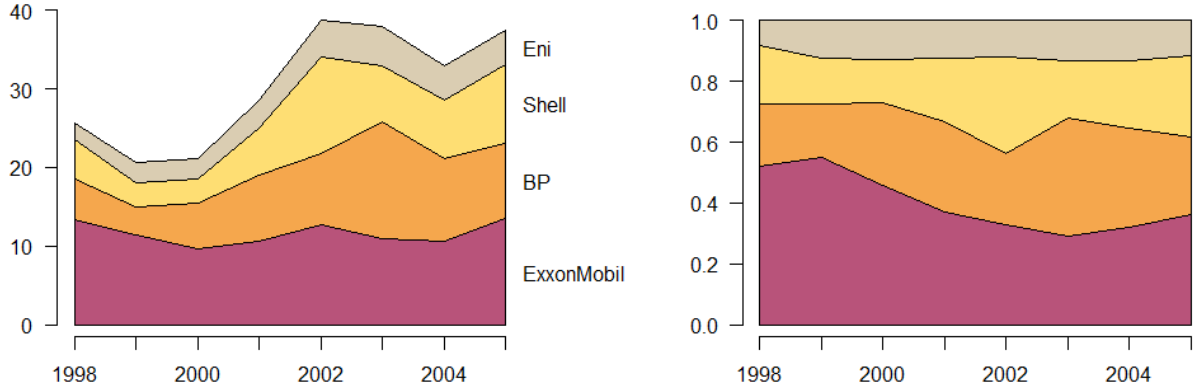
tab (absolute values)

```
> t(t.oil)
           1998 1999 2000 2001 2002 2003 2004 2005
ExxonMobil 13.3 11.4  9.7 10.6 12.7 11.0 10.6 13.5
BP          5.3  3.6  5.8  8.4  9.1 14.8 10.6  9.6
Shell       4.9  3.1  3.0  6.0 12.2  7.1  7.3 10.0
Eni         2.1  2.6  2.7  3.5  4.7  5.0  4.4  4.3
```

ptab (relative values)

```
            1998  1999  2000  2001  2002  2003  2004  2005
ExxonMobil 0.520 0.551 0.458 0.372 0.328 0.290 0.322 0.361
BP         0.207 0.174 0.274 0.295 0.235 0.391 0.322 0.257
Shell      0.191 0.150 0.142 0.211 0.315 0.187 0.222 0.267
Eni        0.082 0.126 0.127 0.123 0.121 0.132 0.134 0.115
```

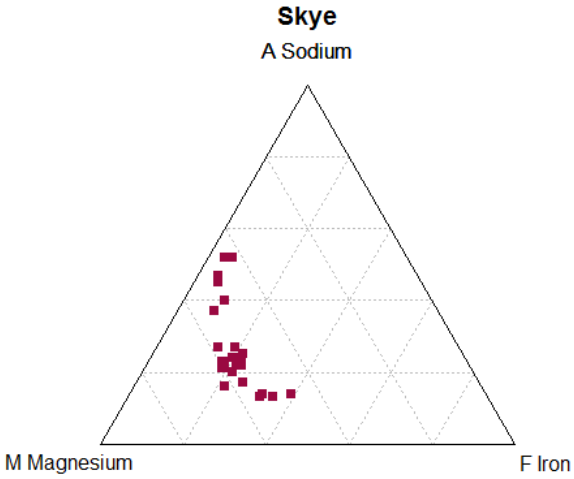
[1] Tufte, Edward R (2001) [1983], The Visual Display of Quantitative Information (2nd ed.), Cheshire, CT: Graphics Press, ISBN 0-9613921-4-2.



14.7 PlotTernary

This produces a ternary or triangular plot.

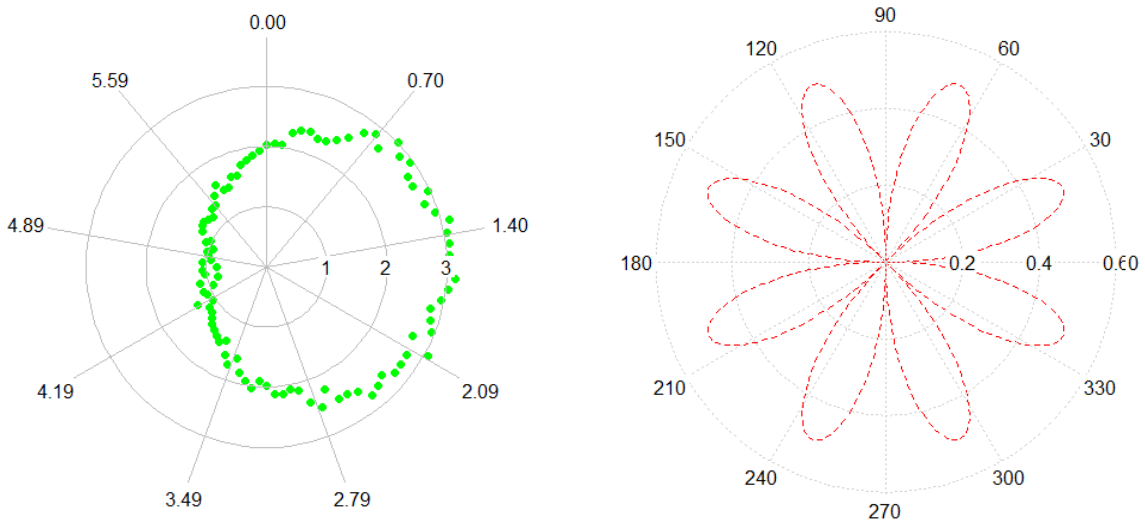
```
data(Skye, package="MASS")
PlotTernary(Skye[c(1,3,2)], pch=15, col=hred, main="Skye",
            lbl=c("A Sodium", "F Iron", "M Magnesium"))
```



14.8 Polar plots

```
testlen <- c(sin(seq(0, 1.98*pi, length=100)) + 2 + rnorm(100)/10)
testpos <- seq(0, 1.98*pi, length=100)
# start at 12 o'clock and plot clockwise
PlotPolar(testlen, -(testpos - pi/2), type="p", main="Test Polygon", col="green", pch=16)

PolarGrid(ntheta = rev(seq(0, 2*pi, by=2*pi/9) + pi/2),
          alabels=Format(seq(0, 2*pi, by=2*pi/9),2)[-10], col="grey",
          lty="solid", lblradians=TRUE)

# just because of its beauty
```

```
t <- seq(0,2*pi,0.01)

PlotPolar(r=sin(2*t)*cos(2*t), theta=t, type="l", lty="dashed", col="red")
PolarGrid()
```

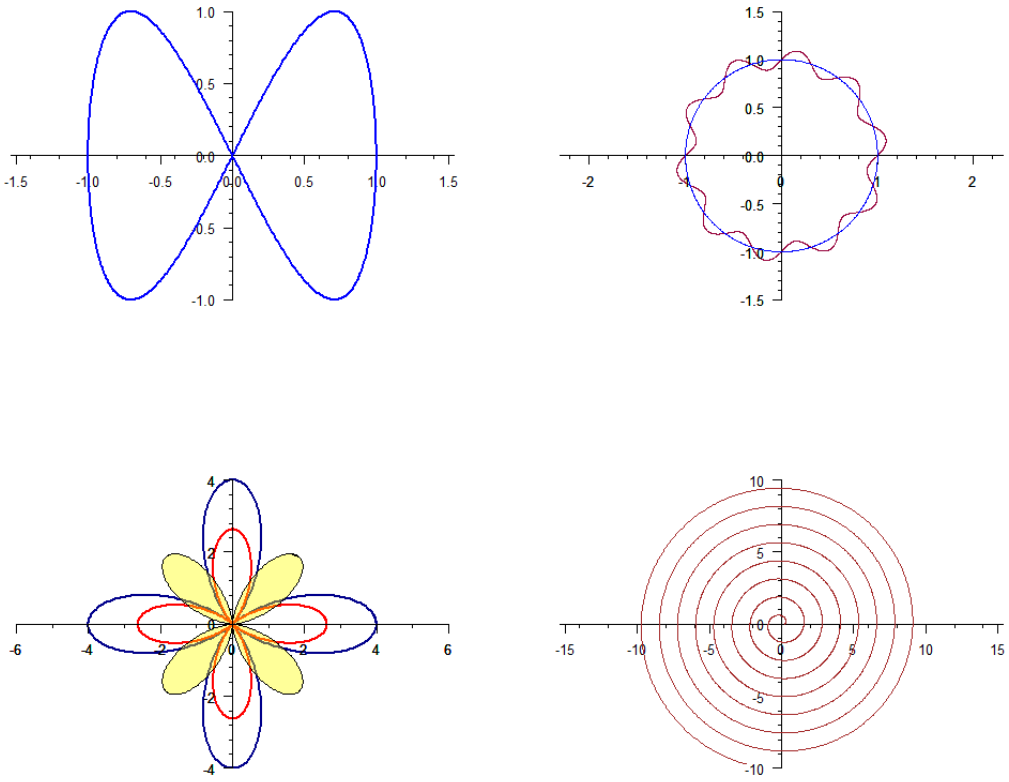


14.9 Plot Functions

Functions can be plotted a bit more comfortable by means of the function `PlotFun`. The idea behind it is to make use of the formula interface, for example x^2 ~ x, and let the function choose appropriate defaults for the rest. (This would be the best case scenario…;-).

There can as well be further parameters defined for plotting more than one function at once. Arguments as `type="n"` or `add=TRUE` are supported.
The function returns the calculated xy-coordinates as list. This can be used to modify the coordinates afterwards, e.g. rotate or translate them.

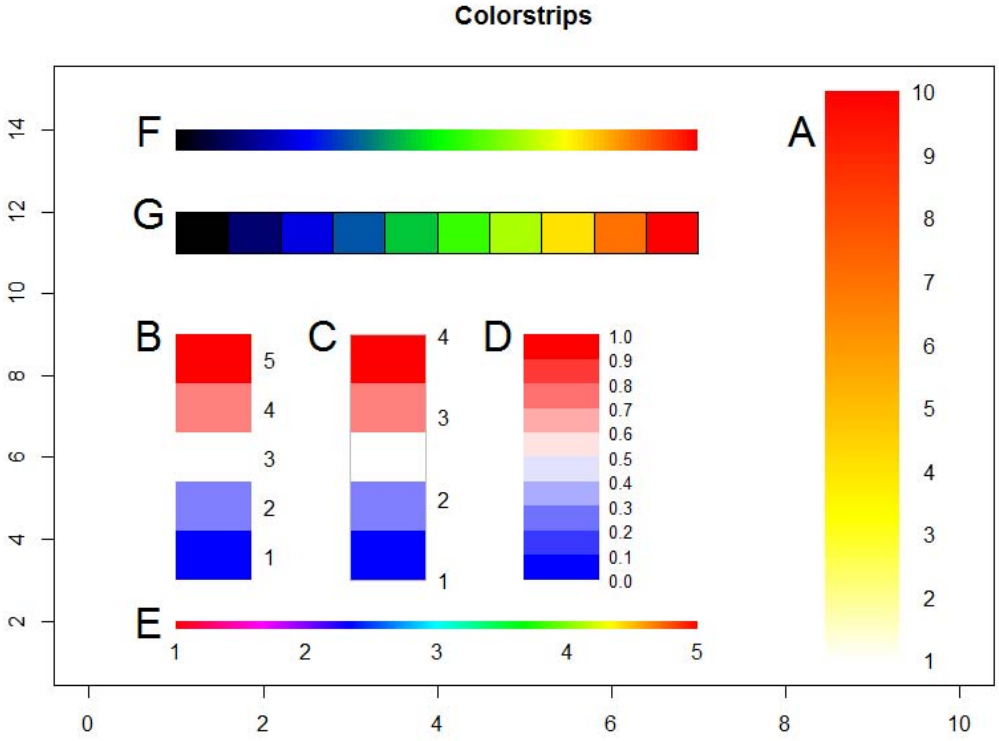
```
# get some data
par(mfrow=c(2,2))
PlotFun(sin(2*t) ~ sin(t), from=0, to=2*pi, by=0.01, col="blue", lwd=2)

PlotFun(1+ 1/10 * sin(10*x) ~ x, polar=TRUE, from=0, to=2*pi, by=0.001, col=hred)
# add a second curve with add=TRUE
PlotFun(sin(x) ~ cos(x), polar=FALSE, from=0, to=2*pi, by=0.01, add=TRUE, col="blue")


# lemniscate of Bernoulli
PlotFun((2*a^2*cos(2*t))^2 ~ t, args=list(a=1), polar=TRUE, from=0, to=2*pi+0.1, by=0.01,
        col="darkblue", lwd=2)
# add the second curve in red
PlotFun((2*a^2*cos(2*t))^2 ~ t, args=list(a=0.9), polar=TRUE, from=0, to=2*pi+0.1, by=0.01,
        col="red", lwd=2, add=TRUE)
# calculate points for a third curve, but do not yet plot it
z <- PlotFun((2*a^2*cos(2*t))^2 ~ t, args=list(a=0.9), polar=TRUE, from=0, to=2*pi+0.1,
             by=0.01, add=TRUE, type="n")
# rotate the structure by pi/4
zz <- Rotate(z$x, z$y, theta=pi/4)
# add a polygon for being able to fill it
polygon(x = zz$x, y=zz$y, col=SetAlpha("yellow", 0.4))

# evolving circle
PlotFun(a*(sin(t) - t*cos(t)) ~ a*(cos(t) + t*sin(t)), args=list(a=0.2), from=0, to=50,
        by=0.01, col="brown")
```



14.10 Legends and colour strips

The function ColorLegend produces colour strips, which often are needed for colour coded maps.

```
plot(1:15,, xlim=c(0,10), type="n", xlab="", ylab="", main="Colorstrips")

# A
ColorLegend(x="right", inset=0.1, labels=c(1:10))

# B: Center the labels
ColorLegend(x=1, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(5), labels=1:5, cntrlbl = TRUE)

# C: Outer frame
ColorLegend(x=3, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(5), labels=1:4, frame="grey")

# D
ColorLegend(x=5, y=9, height=6, col=colorRampPalette(c("blue", "white", "red"),
  space = "rgb")(10), labels=sprintf("%.1f",seq(0,1,0.1)), cex=0.8)

# E: horizontal shape
ColorLegend(x=1, y=2, width=6, height=0.2, col=rainbow(500), labels=1:5,horiz=TRUE)

# F
ColorLegend(x=1, y=14, width=6, height=0.5, col=colorRampPalette(
  c("red","yellow","green","blue","black"), space = "rgb")(100), horiz=TRUE)

# G
ColorLegend(x=1, y=12, width=6, height=1, col=colorRampPalette(c("red","yellow",
  "green","blue","black"), space = "rgb")(10), horiz=TRUE, border="black")
```


Colorstrips
F
G
B
5
4
3
2
1
C
4
3
2
1
D
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
E
1
2
3
4
5
A
10
9
8
7
6
5
4
3
2
1
2
4
6
8
10
12
14
0
2
4
6
8
10

15 Format, Strings and Date functions

15.1 Formatting numbers and dates

Number formatting can sometimes be a nightmare in base R. The function Format tries to concentrate as much as possible form the functionality of formatC, format, symbol, pval etc. into one simple interface.

The following example will use a space as big mark, align the numbers on the position of the "e", flip to scientific notation for numbers < 10^{-2} and for such > 10^{4} and use 3 fixed digits for all numbers.

```
x <- pi * 10^(-5:7)
cbind(Format(x, big.mark=" ", align="e", sci=c(5,2), digits=3))

##       [,1]
##  [1,] "     3.142e-05"
##  [2,] "     3.142e-04"
##  [3,] "     3.142e-03"
##  [4,] "     0.031    "
##  [5,] "     0.314    "
##  [6,] "     3.142    "
##  [7,] "    31.416    "
##  [8,] "   314.159    "
##  [9,] " 3 141.593    "
## [10,] "31 415.927    "
## [11,] "     3.142e+05"
## [12,] "     3.142e+06"
## [13,] "     3.142e+07"
```

Engineering format, set with fmt = "eng", will snap to powers of multiples of 3 when using scientific notation.

```
Format(x, fmt="eng", leading="00", digits=2)

##  [1] "31.42e-06"  "314.16e-06" "03.14e-03"  "31.42e-03"  "314.16e-03" "03.14e+00"
##  [7] "31.42e+00"  "314.16e+00" "03.14e+03"  "31.42e+03"  "314.16e+03" "03.14e+06"
## [13] "31.42e+06"
```

Formatting dates use format codes "d" for days, "m" for months etc.

```
Format(as.Date(c("2014-11-28", "2014-1-2")), fmt="ddd, d mmmm yyyy")
## [1] "Fri, 28 November 2014" "Thu, 2 January 2014"

Format(Today(), fmt="dddd, dd.mm.yyyy")
## [1] "Thursday, 26.05.2016"

Format(Today(), fmt="dddd, yy/mm/dd")
## [1] "Thursday, 16/05/26"

Format(Today(), fmt="dddd, yy/mm/dd", lang="loc")   # with local language
## [1] "Donnerstag, 16/05/26"
```

The format code "p" will produce formatted p-values and is a simple wrapper for format.pval.

```
Format(c(0.442, 0.02125, 4e-21), fmt="p")
## [1] "0.44200"   "0.02125"   "< 2.2e-16"
```

Significance stars mimics the function symnum.

```
Format(c(0.4, 0.02, 0.0004), fmt="*")
## [1] "   " "*  " "***"
```

When formatting percentages the function Format will multiply the numbers with 100, round them to the given number of fixed digits and append a "%"sign.
A sometimes suitable alternative format could be to drop the leading zeros.

```
Format(c(0.24534, 0.4512345, 1.347), fmt="%", digits=2)
## [1] "24.53%"  "45.12%"  "134.70%"

Format(c(0.24534, 0.4512345, 1.347), leading="drop", digits=2)
## [1] ".25"  ".45"  "1.35"
```

NAs and zeros must sometimes be formatted specially. Think eg. of sparse matrices, where one would like the 0s being displayed as "." or maybe even not at all "".

```
Format(c(3.45, 451.2345, 0, NA), digits=2, na.form="<NULL>", zero.form="-")
## [1] "3.45"   "451.23" "-"      "<NULL>"
```

Alignment can be done directly within the function. There are 3 special codes supported, left alignment with "\\l", centered with "\\c" and right with "\\r".

```
cbind(Format(cumsum(10^(0:6)), align="\\c", digits=0))
     [,1]
[1,] "   1   "
[2,] "  11   "
[3,] "  111  "
[4,] " 1111  "
[5,] " 11111 "
[6,] "111111 "
[7,] "1111111"
```

15.2 Date functions

Many date functions are presumably thought to be reached via format and some subsequent cast in base R. However in the analyst's daily life it's often convenient to be able to directly extract parts of a date. So DescTools contains the following ones:

day.name, day.abb	Defined names of the days
AddMonths, AddMonthsYM	Add a number of months to a given date
IsDate	Check whether x is a date object
IsWeekend	Check whether x falls on a weekend
IsLeapYear	Check whether x is a leap year
LastDayOfMonth	Return the last day of the month of the date x
DiffDays360	Calculate the difference of two dates using the 360-days system
Date	Create a date from numeric representation of year, month, day
Day, Month, Year	Extract part of a date
Hour, Minute, Second	Extract part of time
Week, Weekday	Returns ISO week and weekday of a date
Quarter	Quarter of a date
YearDay, YearMonth	The day in the year of a date
Now, Today	Get current date or date-time
HmsToSec, SecToHms	Convert h:m:s times to seconds and vice versa
Zodiac	The zodiac sign of a date :-)

15.3 Strings

String functions are scattered in base R and the solution for some daily tasks are sometimes hard to find. Experts will solve most of their daily life string manipulation with regular expressions. But beginners and a big part of advanced users are supposed to profit by a set of basic string functions.

StrCap	capitalize the first letter of a string
StrAbbr	abbreviates a string
StrTrunc	truncate string on a given length and add ellipses if it really was truncated
StrTrim	delete white spaces from a string
StrPad	fill a string with defined characters to fit a given length
StrRev	reverse a string
StrChop	split a string by a fixed number of characters.
StrCountW	count the words in a string
StrVal	extract numeric values from a string
StrPos	find position of first occurrence of a string in another one
StrIsNumeric	check whether a string does only contain numeric data
FixToTab	create table out of a running text, by using columns of spaces as delimiter
StrDist	compute Levenshtein or Hamming distance between strings
StrExtract	extracts found matches

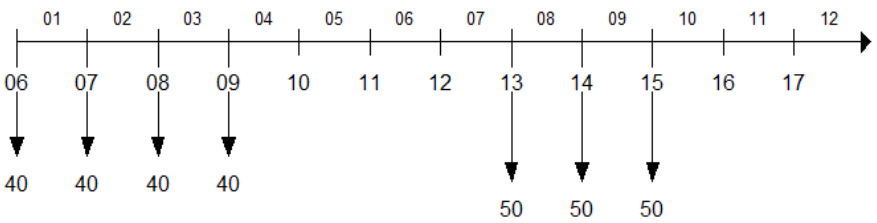
16 Financial functions

DescTools also contains a handful financial functions in order to be able to do the homework for a beginner's course in financial maths.

NPV	Net present value for several cashflows
PMT	Computes the periodic payment of an annuity.
IPMT	Computes the interest payment for an investment for a specified period.
PPMT	Computes the payment on the principal for an investment for a specified period.
IRR	Computes the internal rate of return for a series of cash flows.
YTM	Computes the annual yield of a security that pays interest at maturity.
SLN	Computes the straight-line depreciation of an asset for one period.
DB	Computes the depreciation of an asset for a specified period by using the fixed-declining balance method.
SYD	Computes the sum-of-years digits depreciation of an asset for a specified period.

PlotCashFlow. A cash flow plot is a plot used in finance and allows you to graphically depict the timing of the cash flows as well as their nature as either inflows or outflows. An "up" arrow represents money received and a "down" arrow money paid out.

```
PlotCashFlow(x=c(6:9, 13:15), y=-c(rep(40, 4), rep(50,3)),
             xlim=c(6,17), labels=c(rep(40, 4), rep(50,3)))
```



17 Additional tests

R has a kind of restricted view on what statistical tests are supposed to be useful in practice. The selection of the implemented procedures undoubtedly also corresponds to the application recommendation of the creators of R. Although we are aware that they have probably thought a lot about it, here we put together a series of tests that are often quoted in the statistical literature and should therefore at least be able to be re-enacted and discussed in their advantages and disadvantages.

Name	Purpose
ZTest	One-sample z-test. Tests if a sample comes from a normal distribution with known variance and specified mean, against the alternative that it does not have that mean.
SignTest	One-sample or paired-sample sign test. Tests if a sample comes from an arbitrary continuous distribution with a specified median, against the alternative that it does not have that median.
TTestA	Student's t-test based on sample statistics
YuenTTest	Yuen's robust t-Test with trimmed means and winsorized variances
MosesTest	Moses Test of extreme reactions
JonckheereTerpstraTest	Jonckheere-Terpstra trend test for medians
PageTest	Page test for ordered alternatives
HotellingsT2Test	Hotelling's T2 test for the one and two sample case
VarTest	ChiSquare test for one variance and F test for two variances
SiegelTukeyTest	Non-parametric Siegel-Tukey test for equality in variability.
LeveneTest	Computes Levene's test for homogeneity of variance across groups.
PostHocTest	Wrapper for several post hoc tests (Scheffe, LSD, Tukey) following an ANOVA based on the resulting aov-object
ScheffeTest	Scheffe test following an ANOVA as post-hoc test
DunnTest	Dunn's test of multiple comparisons (following a Kruskal-Wallis test)
ConoverTest	Conover's test of multiple comparisons (following a Kruskal-Wallis test)
NemenyiTest	Nemenyi's test of multiple comparisons (following a Kruskal-Wallis test)
DunnettTest	Dunnett's test of multiple comparisons
PearsonTest	Chi-square goodness-of-fit test. Tests if a sample comes from a specified distribution, against the alternative that it does not come from that distribution.
AndersonDarlingTest	Anderson-Darling test for normality
CramerVonMisesTest	Cramer-von Mises test for normality
LillieTest	Tests if a sample comes from a distribution in the normal family, against the alternative that it does not come from a normal distribution.
ShapiroFranciaTest	Shapiro-Francia test for normality
JarqueBeraTest	Tests if a sample comes from a normal distribution with unknown mean and variance, against the alternative that it does not come from a normal distribution.
CochranQTest	Cochran's Q-test to find differences in matched sets of three or more frequencies or proportions.
RunsTest	Runs test for detecting non-randomness

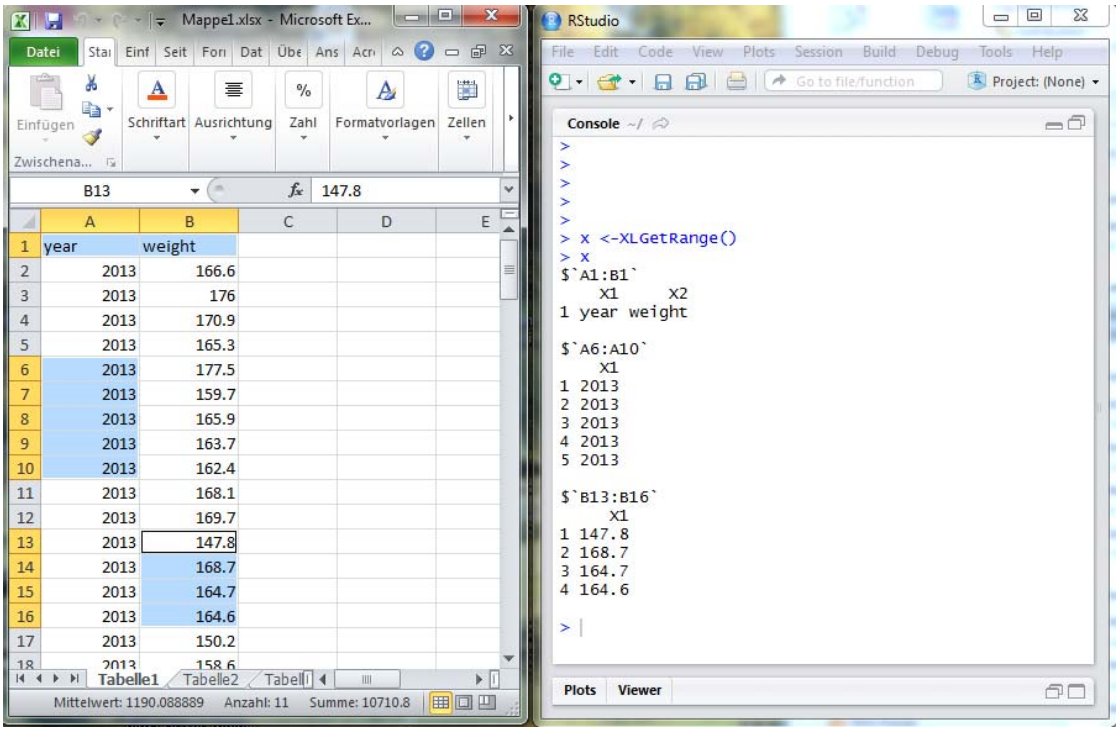
VonNeumannTest	Von Neumann's successive difference test
DurbinWatsonTest	Durbin-Watson test for autocorrelation
BreuschGodfreyTest	Breusch-Godfrey test for higher-order serial correlation.
BartelsRankTest	Bartels rank test for randomness
CochranArmitageTest	Cochran-Armitage test for trend in binomial proportions
MHChisqTest	Mantel-Haenszel Chisquare test
BarnardTest	Barnard's test for 2x2 tables
GTest	Chi-squared contingency table test and goodness-of-fit test
StuartMaxwellTest	Stuart-Maxwell marginal homogeneity test
LehmacherTest	Lehmacher marginal homogeneity test
BreslowDayTest	Test for homogeneity on 2x2xk tables over strata
WoolfTest	Test for homogeneity on 2x2xk tables over strata
HosmerLemeshowTest	Hosmer-Lemeshow goodness of fit tests

Tab. 1) Additional Tests in DescTools

18 Import – Export

18.1 Import data via Excel

The function XLGetRange allows a quick import of data from an Excel-Sheet. The user can either specify a number of cell-references (including a path- and filename) or just select the regions which are to be imported.
The following command will return a list with the contents of the selected cell ranges.



XLView(d.frm) can be used to view a data.frame d.frm in Excel.

18.2 Import SAS datalines

The function ParseSASDatalines can be used to import the SAS data like the following:

```
sas <- "
  data FatComp;
  input Exposure Response Count;
  label Response='Heart Disease';
  datalines;
    0 0  6
    0 1  2
    1 0  4
    1 1 11
 ;"

(FatComp <- ParseSASDatalines(sas))

  Exposure Response Count
1        0        0     6
2        0        1     2
3        1        0     4
4        1        1    11
```

19 DescToolsOptions

There are a few options for the graphical or textual output that can be set. `DescToolsOptions()` displays the currently defined options.

```
$col
    hblue      hred    hgreen
"#8296C4" "#9A0941" "#B3BA12"

$digits
[1] 3

$fixedfont
$name
[1] "Consolas"

$size
[1] 7

attr(,"class")
[1] "font"

$fmt
$fmt$abs
Format name:    abs
 Description:   Number format for counts
 Definition:    digits=0, big.mark="'"
 Example:       314'159

$fmt$num
Format name:    num
 Description:   Number format for floats
 Definition:    digits=3, big.mark="'"
 Example:       314'159.265


$footnote
[1] "¹" "²" "³"

$lang
[1] "engl"

$plotit
[1] TRUE
```

```
$stamp
expression(gettextf("%s/%s", Sys.getenv("USERNAME"), Format(Today(),
    fmt = "yyyy-mm-dd")))

$lastWrd
NULL

$lastXL
NULL

$lastPP
NULL
```

Invoking `DescToolsOptions()` with no arguments returns a list with the current values of the options. Note that not all options listed below are set initially. To access the value of a single option, one can simply use `DescToolsOptions("plotit")`.

To set a new value use the same rationale as with the R options:
`DescToolsOptions(plotit=FALSE)`

`col:`
: a vector of colours, defined as names or as RGB-longs ("#RRGGBB"). By now three colors are used in several plots as defaults. By default they're set to hred, hblue and horange. Change the values by defining DescToolsOptions(col=c("pink", "blue", "yellow")). Any color definition can be used here.

`digits:`
: the number of FIXED digits, used throughout the print functions.

`fixedfont:`
: this font will be used by default, when `Desc` writes to a Word document. Must be defined as a font object, say enumerating name, face and size of the font and setting the class font, e.g. `structure(list(name="Courier New", size=7), class="font")`.

`fmt:`
: Three number format definitions are currently used in the `Desc` routines. The format used for integer values is named `"abs"`, for percentages `"perc"` and for floating point numeric values `"num"`. The format definitions must be of class `"fmt"` and may contain any argument used in the function Format.
Use `Fmt` to access and update formats (as they are organised in a nested list). See the current definitions with:

```
Format(pi*1000, fmt=Fmt("abs"))
# [1] "3'142"
Format(pi*.1, fmt=Fmt("per"))
# [1] "31.4%"
Format(pi*1000, fmt=Fmt("num"))
# [1] "3'141.593"
```

`footnote:`
: a character vector, containing characters to be used as footnote signs. Any character can be defined here. This is currently used by `TOne`.
The author's favorites: `DescToolsOptions("footnote"=c("¹","²","³"))`

`lang:`
: either `"engl"` or `"local"`, defining the language to be used for the names of weekdays and months when using Format.

`plotit:`

logical, defining whether the Desc-procedures should produce plots by default. This is usually a good thing, but it may clutter up your desktop, if you're not using RStudio. Therefore it can be turned off.

`stamp:`
: text or expression to be placed in the right bottom corner of the DescTools plots. This can be useful, if some author or date information should be inserted by default. The default would use an expression as <username>/<date>. See defaults below.

Calling `DescToolsOptions(reset=TRUE)` will reset the options to these defaults:

```
options(DescTools = list(
  col       = c(hblue="#8296C4", hred="#9A0941", hgreen="#B3BA12"),
  digits    = 3,
  fixedfont = structure(list(name = "Consolas", size = 7), class = "font"),
  fmt       = list(abs = structure(list(digits = 0, big.mark = "'"),
                      name = "abs", label = "Number format for counts", default = TRUE,
                      class = "fmt"),
                   per = structure(list(digits = 1, big.mark = "%"),
                      name = "per", label = "Percentage number format", default = TRUE,
                      class = "fmt"),
                   num = structure(list(digits = 3, big.mark = "'"),
                      name = "num", label = "Number format for floats", default = TRUE,
                      class = "fmt")
                ),
  footnote  = c("'", "\"", "\"\""),
  lang      = "engl",
  plotit    = TRUE,
  stamp     = expression(gettextf("%s/%s", Sys.getenv("USERNAME"),
                                  Format(Today(), fmt = "yyyy-mm-dd")))
))
```

This code can as well be copied and pasted to the users' RProfile file, in order to have the options permanently available.

20 References

http://cran.r-project.org/web/packages/vcdExtra/vignettes/vcd-tutorial.pdf

http://www.stattutorials.com/SAS/TUTORIAL-PROC-FREQ-1.htm

Vittinghoff, E., Glidden, D. V., Shiboski, S. C., and Mcculloch, C. E. (2005) Regression Methods in Biostatistics: Linear, Logistic, Survival, and Repeated Measures Models. Springer, New York

http://support.sas.com/documentation/cdl/en/statugfreq/63124/PDF/default/statugfreq.pdf

http://www.stat.ufl.edu/~presnell/Courses/sta4504-2011sp/Notes/icda-notes-3x2.pdf

Agresti, A. (2002) Categorical Data Analysis. John Wiley & Sons.

Dalgaard, P. (2008) Introductory Statistics with R (2. Aufl.), London, UK: Springer.

Wollschläger, D. (2010, 2012) Grundlagen der Datenanalyse mit R, Springer, Berlin.

Tufte, Edward R (2001) [1983], The Visual Display of Quantitative Information (2nd ed.), Cheshire, CT: Graphics Press