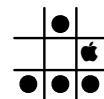
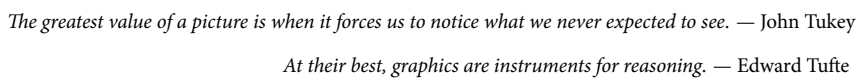


A Visual Guide to R Graphics and Data Munging

With 65 illustrations



7d26af9
2015/04/10



At their best, graphics are instruments for reasoning. — Edward Tufte

Contents

1	Getting started with R graphics	3
1.1	Why R?	3
1.2	The R graphical model	4
1.3	Base vs. lattice graphics	4
1.4	The grammar of graphics	4
2	Data management	5
2.1	Structuring data	5
2.2	Managing data	5
2.2.1	Variable recoding and annotation	6
2.2.2	Variable transformation	8
2.3	Indexing, subsetting, conditioning	9
2.4	Summarizing data	9
2.4.1	Base R functions	10
2.4.2	The Hmisc package	10
2.4.3	The plyr package	14
3	Univariate distributions	15
3.1	Stripchart	15
3.2	Histograms	17
3.3	Density plots	19
3.4	Quantile and related probability plots	21
3.5	Boxplots	24
3.6	Time series	25

4	Two-way graphics	31
4.1	Lineplots	32
4.2	Scatterplots	32
4.3	Barcharts	37
4.4	Dotcharts	37
4.5	Line fits	39
4.6	Time series	40
4.7	Level plot	42
5	Multi-way graphics	45
5.1	Parallel displays	45
5.2	Scatterplot matrix	45
5.3	Three-way tabular data	45
5.4	N-way data	45
6	Customizing theme and panels	47
7	The Hmisc plotting functions	49
7.1	Two-way graphics	49
8	The ggplot2 package	51
8.1	Why ggplot2	51
8.2	Core components of a ggplot graphic	51
9	Interactive and dynamic displays	53
9.1	Exploratory data analysis	53
9.2	Brushing and linking	53
9.3	The ggobi toolbox	53

CHAPTER 1

Getting started with R graphics

1.1 Why R?

In the “GNU world”, most of the plotting program expect data from text file (tab-delimited or csv) arranged by columns, with extra grouping levels denoted by string or integer codes. This is the case with `gnuplot`¹ (<http://www.gnuplot.info/>) or `plotutils` (<http://www.gnu.org/software/plotutils/>), for example.

Here is how we could create an histogram in `gnuplot`, from a series of 500 gaussian variates generate using R as follows:

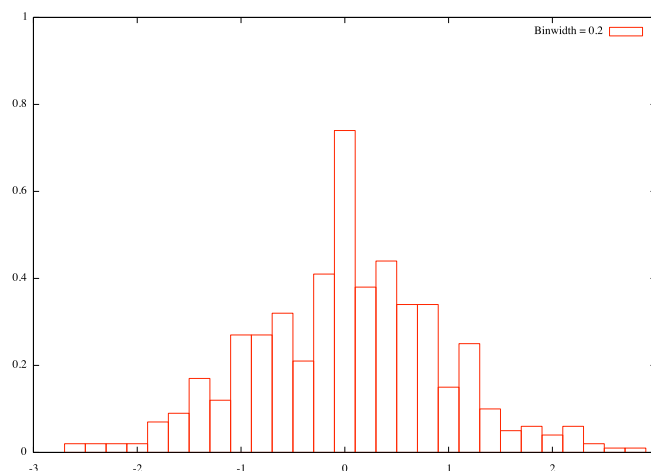
```
$ Rscript -e 'cat(rnorm(500), sep="\n")' > rnd.dat
```

Then, in `gnuplot`, we can run

```
bw=0.2
n=500
bin(x,width)=width*int(x/width)
tstr(n)=sprintf("Binwidth = %1.1f\n", n)
set xrange [-3:3]
set yrange [0:1]
set boxwidth bw
plot 'rnd.dat' using (bin($1,bw)):(1./(bw*n)) smooth frequency \
    with boxes title tstr(bw)
```

to get the picture shown below. (Note that we didn't try to customize anything, except the title.)

¹PK Janert. *Gnuplot in Action. Understanding Data with Graphs*. Manning Publications, 2009. ISBN: 978-1933988399.



The above example shows one important aspect of using a dedicated statistical package: Gnuplot has no function to draw an histogram, and we have to write some code to perform additional tasks, like binning in this case. The same applies for plotutils. We could make use of external programs to do that, like the GSL library (see example 22.11 from the manual, <http://www.gnu.org/software/gsl/manual/>), but this two-stage approach is rather likely to be cumbersome for repetitive tasks.

The author found that Stata is one of the only great alternative to R, but it has its cost. In fact, this textbook is largely inspired from one of Stata Press book on Stata graphical capabilities.²

1.2 The R graphical model

1.3 Base vs. lattice graphics

R comes with several graphical system, including so-called “Base” graphics, which rely on `grDevices` and `??`. While these plotting functions are quite useful, they lack several interesting aspects that are available through `grid`-based packages, including `lattice` and `ggplot2`, and in particular the `update` and `print` function.

1.4 The grammar of graphics

²MN Mitchell. *A Visual Guide to Stata Graphics*. Stata Press, 2008. ISBN: 978-1597180399.

CHAPTER 2

Data management

2.1 Structuring data

In R, the most common structure used to store a data set with mixed-type variables is a `data.frame`. Such an R object presents several characteristics that makes it most appropriate for managing statistical data structure, with few exceptions (e.g., when one only has to work with aggregated data or two-way tables). It should be noted that other data structures might be more appropriate, for example when one is interested in time series analysis, but see the `zoo` package.¹

Many R functions accept `data.frame` as input, and further allow to subset or index it for computation or visualization purpose. In addition to receiving a `data.frame`, some R commands allow to use a *formula* notation, where the right and left-hand side are separated by the `~` (tilde) operator. The use of together with a `data.frame` simplify the accession of variable in a given environment. This is especially true when using the `lattice` package which is entirely based on formula, even if this is not apparent at first sight.

2.2 Managing data

Consider, for example, the *low birth study* which is discussed at length in Hosmer and Lemeshow’s textbook on logistic regression.² A quick look at the variables should make it clear that they won’t be treated the way we like them to be considered: mother’s ethnicity status (`race`) takes three integer values, without any explicit meaning.

```
data(birthwt, package="MASS")
str(birthwt)
```

¹A Zeileis and G Grothendieck. “zoo: S3 Infrastructure for Regular and Irregular Time Series”. In: *Journal of Statistical Software* 14.6 (2005). URL: <http://www.jstatsoft.org/v14/i06>.

²D Hosmer and S Lemeshow. *Applied Logistic Regression*. New York: Wiley, 1989. ISBN: 0-471-35632-8.

2.2.1 Variable recoding and annotation

The `Hmisc` package includes numerous R functions that will facilitate the task of data checking (`describe` provides “codebook” facilities), summarizing (`summary.formula`) or aggregating data, (`summary.formula`).

Here are some examples of use with the `birthwt` data. For illustration purpose, we set some observations as missing on two variables (`age` and `ftv`).

```
set.seed(101)
birthwt$age[5] <- NA
birthwt$ftv[sample(1:nrow(birthwt), 5)] <- NA
yesno <- c("No", "Yes")
birthwt <- within(birthwt, {
  smoke <- factor(smoke, labels = yesno)
  low <- factor(low, labels = yesno)
  ht <- factor(ht, labels = yesno)
  ui <- factor(ui, labels = yesno)
  race <- factor(race, levels = 1:3, labels = c("White", "Black", "Other"))
  lwt <- lwt/2.2 ## weight in kg
})
```

It often helps to keep variable names short and informative, and to have separated labels and/or units in case of continuous measurements. This is available via the `label` and `units` functions. Note that `label` allows to annotate the data frame as well.

```
library(Hmisc)
label(birthwt$age) <- "Mother age"
units(birthwt$age) <- "years"
label(birthwt$bwt) <- "Baby weight"
units(birthwt$bwt) <- "grams"
label(birthwt, self = TRUE) <- "Hosmer & Lemeshow's low birth weight study."
```

These labels can then be used in almost every `Hmisc` functions, even when using `list.tree` in place of `str`. However, we will see that there mostly useful when generating Tables or Figures.

The `contents` command offers a quick summary of data format and missing values, and it provides a list of labels associated to variables treated as factor by R.

Once we have a working data frame, the functions `contents` and `describe` provide two quick summary of the data.

```
> contents(birthwt)
```

```
Data frame:birthwt 189 observations and 10 variables    Maximum # NAs:5
```

	Labels	Units	Levels	Class	Storage	NAs
low			2	integer		0
age	Mother age	years		integer	integer	1

lwt		double	0
race	3	integer	0
smoke	2	integer	0
ptl		integer	0
ht	2	integer	0
ui	2	integer	0
ftv		integer	5
bwt	Baby weight grams	integer	integer
			0

+-----+-----+-----+		
Variable	Levels	
+-----+-----+-----+		
low	No, Yes	
smoke		
ht		
ui		
+-----+-----+-----+		
race	White, Black, Other	
+-----+-----+-----+		

As can be seen, `contents` displays storage mode and class of R variables that are present in the data frame. The number of missing values is also reported for each variable. The levels of each `factor`-type variable is also printed at the end of the output.

```
> describe(subset(birthwt, select = c(low, age, race, ftv)), digits = 3)
```

4 Variables				189 Observations						

low										
	n	missing	unique							
	189	0	2							
No (130, 69%), Yes (59, 31%)										

age : Mother age [years]										
	n	missing	unique	Info	Mean	.05	.10	.25	.50	.75
	188	1	24	1	23.3	16	17	19	23	26
	.90	.95								
	31	32								
lowest : 14 15 16 17 18, highest: 33 34 35 36 45										

race										
	n	missing	unique							
	189	0	3							
White (96, 51%), Black (26, 14%), Other (67, 35%)										

ftv										
	n	missing	unique	Info	Mean					

```

184      5      6    0.83    0.783

      0  1  2  3  4  6
Frequency 98 45 30 7 3 1
%         53 24 16 4 2 1
-----

```

The `describe` function gives more details, and, in particular, provides useful summary statistics for each variable. Here, we only considered four variables: a binary variable (`low`), a continuous measure (`age`), a three-level factor (`race`), and a count variable (`ftv`). The output will be different depending on the type of variable, and for all but continuous measures (defined as variable taking at least 10 distinct values) `describe` will display a table of counts and frequencies.

2.2.2 Variable transformation

In case we would like to consider one of the above factors as a numerical variable, we can now use `as.numeric` and R will take care of attributing the lowest integer score to the baseline category. Of course, there might be occasion where we would like to change that reference level; or, we might want to collapse two discrete categories. Again, there are simple commands to do that, for example:

```

birthwt$low <- relevel(birthwt$low, "Yes")
levels(birthwt$race)[2:3] <- "Black+Other"

```

Another common task consists in transforming some predictors, either for visualization purpose or when building an explanatory or predictive model. As a simple example, we can imagine centering some of the predictors of interest, like `age`, in the above example. The `within` or `transform` command can be used to append the centered variable to the list of variables present in the `data.frame`:

```

birthwt <- transform(birthwt, age.c=scale(age, scale=FALSE))

```

Likewise, we may want to recode previous premature labours (`ptl`) as yes/no and number of physician visits during the first trimester (`ftv`) as one/more than one, like shown below (we show two different syntax that basically perform the same task by relying on R indexing):

```

birthwt <- transform(birthwt, ptl.yn=factor(ptl > 0, labels=c("No", "Yes")),
  ftv.c=factor(ifelse(ftv < 2, "1", "2+")))

```

`Hmisc` provides a replacement for R's `cut` function with better default options (especially the infamous `include.lowest = FALSE`) to discretize a continuous variable. The `cut2` function has many useful options, including `g=` and `levels.mean=` to return `g` classes and report center of each class instead of class intervals:

```

table(cut2(birthwt$age, g = 3, levels.mean = TRUE, digits = 3))

```

If there is some reason to treat `ftv` as an ordered factor, a command like

```

as.ordered(cut2(birthwt$ftv, c(0, 1, 2, 6)))

```

might do the job.

2.3 Indexing, subsetting, conditioning

A lot of statistical operations that practitioners use to apply on a given dataset are mostly variations around the idea of indexing or subsetting. By comparison, SQL-like operations would be selection and projection.

The `subset` command offers a simple and elegant way to combine both operations: for a given data frame, the `subset` = option is used to filter rows according to logical expression or simple row indexes while the `select` = option is used to return only selected variables based on their index (e.g., column 1 and 3) or an unquoted name (e.g., `c(low, lwt)`).

The following instruction will print the age of hypertensive mothers whose baby was underweight:

```
subset(birthwt, low == "Yes" & ht == "Yes", age)
```

It should be noted that `subset` returns a data frame.

Some people might prefer to use their favorite SQL-like language, so something like this would perfectly fit their needs:

```
library(sqldf)
sqldf("SELECT age FROM birthwt WHERE low = 0 AND ht = 1 LIMIT 3", row.names = TRUE)
```

Unfortunately, this doesn't work with "labelled" objects, as typically returned by `Hmisc`, although a solution is readily available at <http://stackoverflow.com/q/2394902/420055>.

There are also a bunch of command dedicated to variables clustering, analysis of missing patterns, or simple (`impute`) or multiple (`aregImpute`, `transcan`) imputation methods. Here is how we would impute missing values with the median in the case of a continuous variable:

```
lwt <- birthwt$lwt
lwt[sample(length(lwt), 10)] <- NA
lwt.i <- impute(lwt)
summary(lwt.i)
```

Missing observations will be marked with an asterisk when we print the whole object in R. To use the mean instead of the median, we just have to add the `fun = mean` option.

2.4 Summarizing data

Statisticians generally spend a great part of their time in data cleansing, data transformation or re-expression,³ and data visualization.

³DC Hoaglin, F Mosteller, and JW Tukey. *Understanding Robust and Exploratory Data Analysis*. Wiley-Interscience, 1983. ISBN: 0-471-09777-2.

2.4.1 Base R functions

Both the `by` and `tapply` function allow to apply a builtin or custom function to help summarizing a numeric variable by a categorical variable. Most of the time, these two functions are less handy than `aggregate` since the latter offers a formula interface and returns a data frame.

```
aggregate(bwt ~ ui + I(ftv > 1), data = birthwt, mean)
```

There is, however, one caveat when using `aggregate`: even if you can pass a custom function that returns multiple values that can be printed on screen, the resulting data frame will still have only one column for its output. For exemple, the dimensions of the data frame created by the following call to `aggregate` is 4 by 3, even if it looks like there two separate columns for bwt mean and SD.

```
f <- function(x) c(mean = mean(x), sd = sd(x))
aggregate(bwt ~ ui + I(ftv > 1), data = birthwt, f)
```

2.4.2 The `Hmisc` package

There are three useful commands that provide summary statistics for a list of variables. They implement the “split-apply-combine” strategy⁴ in the spirit of R’s built-in functions (unlike `plyr`).

The first one, `summarize`, can be seen as an equivalent to R’s `aggregate` command. Given a response variable and one or more classification factors, it applies a specific function to all data chunk, where each chunk is defined based on factor levels. The results are stored in a matrix, which can easily be coerced to a data frame using, e.g., `as.data.frame` or `Hmisc::matrix2dataFrame`.

Note that enhanced results can be printed on the console using the `prn` command.

Here is a first example, using a dedicated function to print the mean and standard deviation (SD) of a numerical variable:

```
f <- function(x, na.opts = TRUE) c(mean = mean(x, na.rm = na.opts), sd = sd(x,
  na.rm = na.opts))
out <- with(birthwt, summarize(bwt, race, f))
```

Here is the output produced by R:

```
Average baby weight by ethnicity    out

      race  bwt    sd
3 White 3103 727.9
1 Black 2720 638.7
2 Other 2805 722.2
```

⁴H Wickham. “The Split-Apply-Combine Strategy for Data Analysis”. In: *Journal of Statistical Software* 40.1 (2011). URL: <http://www.jstatsoft.org/v40/i01>.

Contrary to `aggregate`, this command provides multiway data structure in case we ask to compute more than one quantity. In this case, the dimension of out is a 3 by 3 data frame. Summarizing multivariate responses or predictors is also possible, with either `summarize` or `mApply`. Of course, any built-in functions, such as `colMeans` could be used in place of our custom summary command.

```
with(birthwt, summarize(bwt, llist(race, smoke), f))
```

The second command, `bystats`, (or `bystats2` for two-way tabular output) allows to describe with any custom or built-in function one or multiple outcome by two explanatory variables, or even more. Sample size and the number of missing values are also printed.

With the following instruction,

```
with(birthwt, bystats(cbind(bwt, lwt), smoke, race))
```

we would get

Mean of cbind(bwt, lwt) by smoke

	N	bwt	lwt
No White	44	3429	63.11
Yes White	52	2827	57.41
No Black	16	2854	67.93
Yes Black	10	2504	64.82
No Other	55	2816	54.16
Yes Other	12	2757	56.36
ALL	189	2945	59.01

whereas

```
with(birthwt, bystats2(lwt, smoke, race))
```

gives

Mean of lwt by smoke

	N	Mean
No	115	60.02
Yes	74	58.24
ALL	189	59.01

The third and last command is `summary.formula`, which can be abbreviated as `summary` as long as `formula` is used to describe variables relationships. There are three main configurations (`method=`): "response", where a numerical variable is summarized for each level of one or more variables (numerical variables will be discretized in 4 classes), as `summarize` does; "cross", to compute conditional and marginal means of several response variables described by at most 3 explanatory variables (again, continuous predictors are represented as quartiles); "reverse", to summarize univariate distribution of a set of variables for each level of a classification variable (which appears on the left-hand side of the formula). Variables are viewed as continuous as long as they have more than 10 distinct values, but this can be changed by setting, e.g., `continuous = 5`. When `method = "reverse"`, it is possible to add `overall = TRUE`, `test = TRUE` to add overall statistics and corresponding statistical tests of null effect between the groups.

Here are some examples of use, with their outputs.

```
> summary(bwt ~ race + ht + lwt, data = birthwt)
```

Baby weight N=189

```
+-----+-----+-----+
|      |      |N |bwt |
+-----+-----+-----+
|race  |White  | 96|3103|
|      |Black  | 26|2720|
|      |Other  | 67|2805|
+-----+-----+-----+
|ht     |No     |177|2972|
|      |Yes    | 12|2537|
+-----+-----+-----+
|lwt    |[36.4, 50.9]| 53|2656|
|      |[50.9, 55.5]| 43|3059|
|      |[55.5, 64.1]| 46|3075|
|      |[64.1,113.6]| 47|3038|
+-----+-----+-----+
|Overall|      |189|2945|
+-----+-----+-----+
```

```
> summary(cbind(lwt, age) ~ race + bwt, data = birthwt, method = "cross")
```

mean by race, bwt

```
+-----+
|N      |
|Missing|
|lwt    |
|age     |
+-----+
+-----+-----+-----+-----+-----+
| race|[ 709,2424)|[2424,3005)|[3005,3544)|[3544,4990]| ALL |
+-----+-----+-----+-----+-----+
```


White	19	23	20	33	95
	0	1	0	0	1
	55.55	57.67	62.23	63.25	60.14
	22.74	24.78	24.50	24.91	24.36
+-----+-----+-----+-----+-----+					
Black	9	9	6	2	26
	0	0	0	0	0
	65.10	59.70	70.83	93.41	66.73
	23.44	20.89	20.00	20.50	21.54
+-----+-----+-----+-----+-----+					
Other	20	16	19	12	67
	0	0	0	0	0
	51.23	52.95	58.90	55.34	54.55
	22.20	22.69	22.26	22.50	22.39
+-----+-----+-----+-----+-----+					
ALL	48	48	45	47	188
	0	1	0	0	1
	55.54	56.48	61.97	62.51	59.06
	22.65	23.35	22.96	24.11	23.27
+-----+-----+-----+-----+-----+					

```
> summary(low ~ race + ht, data = birthwt, fun = table)
```

```
low      N=189
```

+-----+-----+-----+-----+					
		N	No	Yes	
+-----+-----+-----+-----+					
race	White	96	73	23	
	Black	26	15	11	
	Other	67	42	25	
+-----+-----+-----+-----+					
ht	No	177	125	52	
	Yes	12	5	7	
+-----+-----+-----+-----+					
Overall		189	130	59	
+-----+-----+-----+-----+					

Finally, here is a more complex Table produced by `summary.formula` with `method = "reverse"`.

```
out <- summary(low ~ race + age + ui, data = birthwt, method = "reverse", overall = TRUE,
               test = TRUE)
print(out, prmsd = TRUE, digits = 2)
```

Although we display the output on the console directly, it is also possible to build a *LaTeX* Table using the `latex` function.

```
latex(out, file = "tab-summary.tex", ctable = TRUE, digits = 1, exclude1 = FALSE, caption = NULL)
```

Here is the final output

	N	No	Yes	Combined	Test Statistic
		<i>N</i> = 130	<i>N</i> = 59	<i>N</i> = 189	
race : White	189	56% (73)	39% (23)	51% (96)	$\chi^2_2 = 5, P = 0.08^1$
Black		12% (15)	19% (11)	14% (26)	
Other		32% (42)	42% (25)	35% (67)	
Mother age _{years}	188	19 23 28	20 22 25	19 23 26	$F_{1,186} = 1, P = 0.2^2$
ui : No	189	89% (116)	76% (45)	85% (161)	$\chi^2_1 = 5, P = 0.02^1$
Yes		11% (14)	24% (14)	15% (28)	

a b c represent the lower quartile *a*, the median *b*, and the upper quartile *c* for continuous variables.

N is the number of non-missing values.

Numbers after percents are frequencies.

Tests used:

1Pearson test; 2Wilcoxon test

2.4.3 The `plyr` package

The `plyr` package⁵ offers a general solution to those kind of tasks.

⁵Ibid.

Univariate distributions

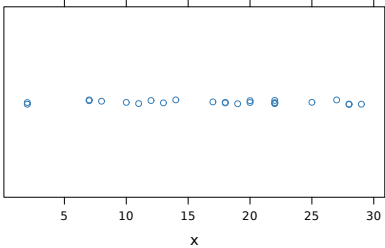
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

3.1 Stripchart

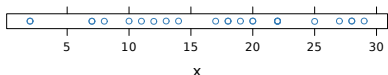
Stripchart aims at showing the distribution of a series of continuous measurements (much like scatterplot for 2D data discussed in § 4.2). They are useful for small to moderate dataset. With large N it is probably better to switch to alternative displays, see next sections.

```
x <- sample(1:30, 25, replace=TRUE)
stripplot(~ x, jitter.data=TRUE, factor=.8, aspect=.5)
```

With this synthetic dataset where several observations can take the same value, jittering point locations on the horizontal and vertical axes ensures a better representation.

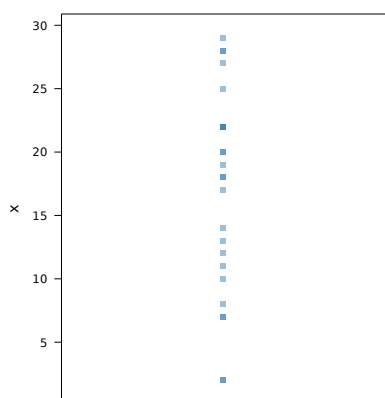


```
stripplot(~ x, jitter.data=TRUE, factor=.8, aspect="xy")
```



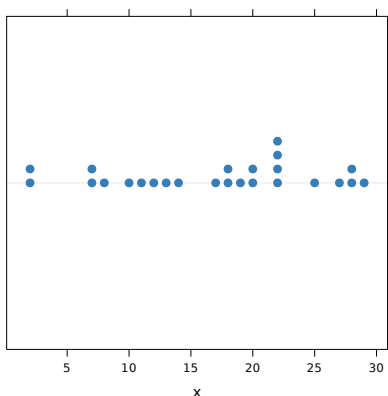
A better way of flattening the display is to use an “xy” aspect.

```
stripplot(x ~ 1, horizontal=FALSE, jitter.data=TRUE, aspect=1.2,
scales=list(x=list(draw=F)), xlab="", pch=15, alpha=.5)
```



This is basically the same picture but the x and y axis have been transposed. We used a different symbol and transparency to highlight where replication occurs in the data. Obviously, that won't work so nicely with larger sample size or a higher density of replication.

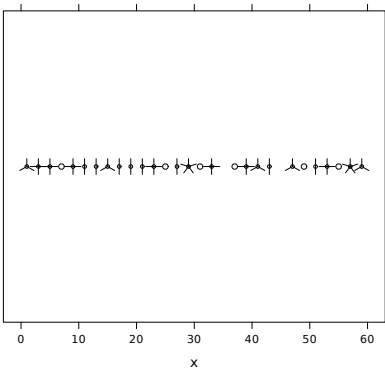
```
stripplot(~ x, panel=HH::panel.dotplot.tb, cex=1.2, factor=.2)
```



In contrast to the base `stripchart` function, there is no way of imposing a stacked display in lattice. However, there is some convenient panel function in the HH package.

```
x <- sample(seq(1, 60, by=2), 75, replace=TRUE)
stripplot(1 ~ x, panel=panel.sunflowerplot, col="black",
          seg.col="black", seg.lwd=1, size=.08)
```

With possible replicates, it is also interesting to use “sunflowers” where multiple leaves are used for each duplicate. The custom panel function mimics the base `sunflowerplot` function. For an alternative way of embedding “sunflowers” into a lattice display, see the following thread on R-help: <http://bit.ly/Ig4RTq>. Note that we remove *y*-axis annotation using commands presented before (i.e., using `scales=`).

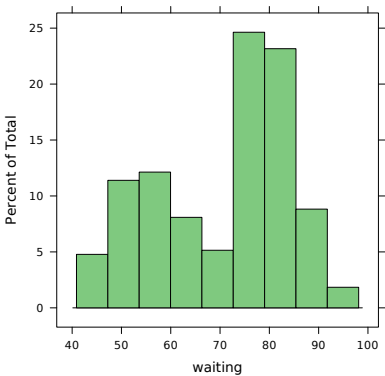


3.2 Histograms

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

```
histogram(~ waiting, data=faithful)
```

A simple histogram of waiting time expressed as density. Note that forgetting the `~` operator will raise an error message.



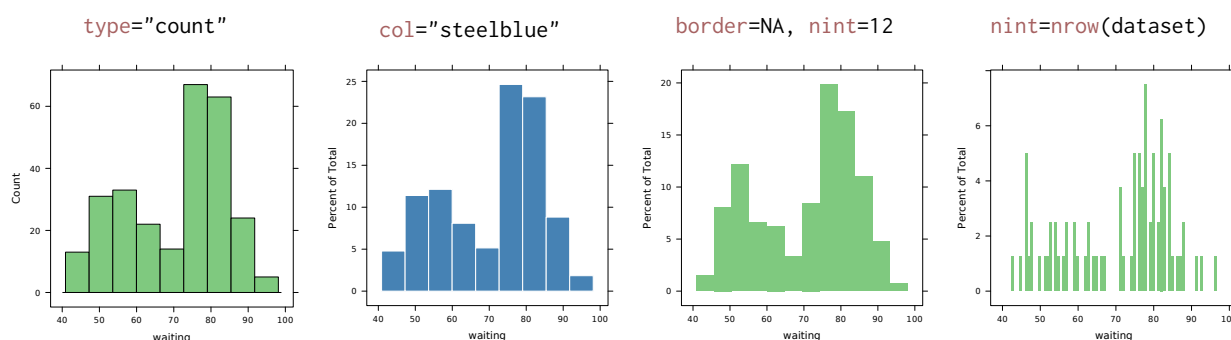
faithful. Waiting time between eruptions and the duration of the eruption for the Old Faithful geyser in Yellowstone National Park, Wyoming, USA.

Box 3.1 shows some custom settings with the `faithful` dataset. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, con-

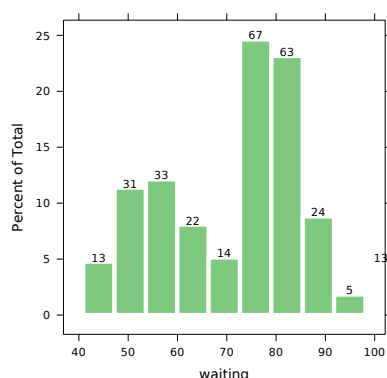
sectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Box 3.1

Common options for `histogram` include displaying counts instead of percents, or varying bar color. It is also possible to change default bin size. When `nint=nrow(dataset)`, we have a so-called high-density vertical lines, much like when using `plot(..., type="h")`.



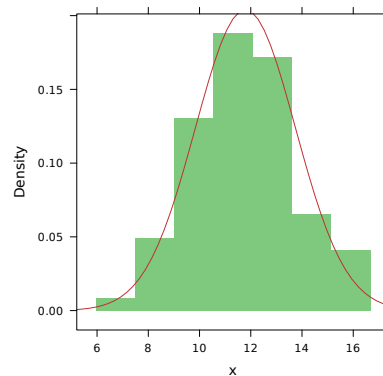
```
p <- histogram(~ waiting, data=faithful, lwd=5, type="percent",
               border="white")
x.breaks <- p$panel.args.common$breaks
y.values <- table(cut(faithful$waiting, breaks=x.breaks))
update(p, panel=function(...) {
  panel.histogram(...)
  panel.text(x.breaks+diff(x.breaks)[1]/2,
             y.values/nrow(faithful)*100,
             as.character(y.values), cex=.8, adj=c(.5,0))
})
```



The following example demonstrates how a default histogram displaying percents data can be annotated with counts data. This is intended to show how we can steal away default setting to display the distribution of discrete values.)

```
x <- rnorm(80, mean=12, sd=2)
histogram(~ x, type="density", border=NA,
  panel=function(x, ...) {
    panel.histogram(x, ...)
    panel.mathdensity(dmath=dnorm, col="#BF3030",
      args=list(mean=mean(x), sd=sd(x)))
  })
```

An example where we superimposed a normal density with parameters estimated from the sample.)

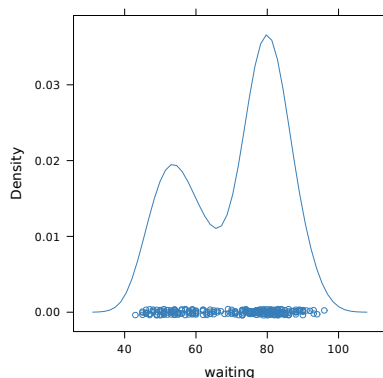


3.3 Density plots

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

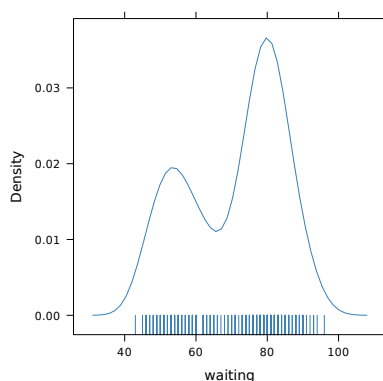
¹BW Silverman. *Density Estimation*. London: Chapman and Hall, 1986. ISBN: 978-0412246203.

```
densityplot(~ waiting, data=faithful)
```



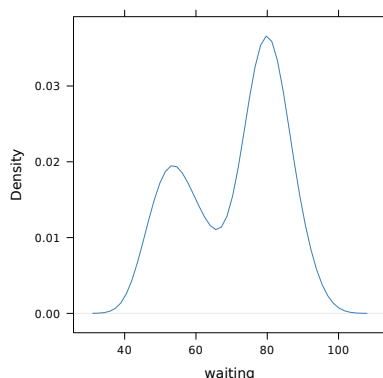
The default panel relies on R's `density` function. As such, the default kernel is gaussian with $n = 512$ equally spaced at which the density is estimated. The choice of the bandwidth follows Silverman's rule of thumb, namely $\min(0.9SD, IQR/1.34n)$. An alternative bandwidth can be selected using `bw="SJ"`.

```
densityplot(~ waiting, data=faithful, plot.points="rug")
```



Instead of a mini stripchart displayed at $y = 0$, a "rugplot" can be preferred. It might help spotting possible local concentration of data points, compared to simple jittered points.

```
densityplot(~ waiting, data=faithful, plot.points=FALSE, ref=TRUE)
```



Sometimes, adding a reference line crossing at $y = 0$ may be informative, especially for multi-modal distributions. It is advised to avoid plotting individual observations like was done in the preceding graphics.

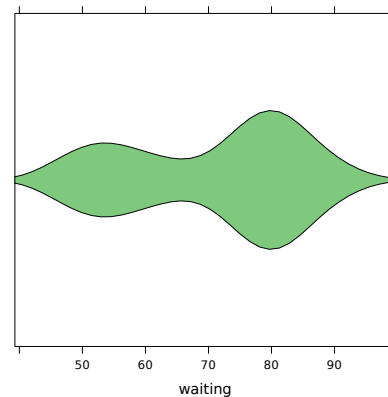
An alternative way of presenting density plots are so-called "violin plots"² which feature

²JL Hintze and RD Nelson. "Violin Plots: A Box Plot-Density Trace Synergism". In: *The American Statistician* 52.2 (1998), pp. 181–184.

the main components of a boxplot (§ 3.5) and a kernel density estimation, and “bean plots”³ where density trace are mirrored to form a polygon shape. The latter presents the advantage of allowing asymmetrical plotting depending on a grouping factor.

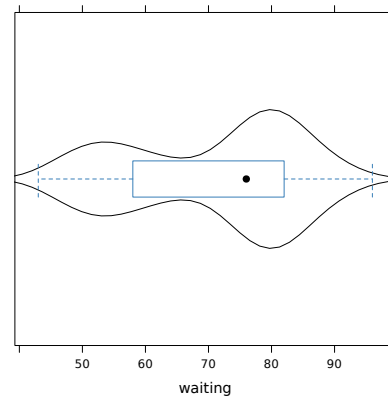
```
bwplot(~ waiting, data=faithful, panel=panel.violin)
```

A simple “violin” panel.



```
bwplot(~ waiting, data=faithful,
  panel=function(..., box.ratio) {
    panel.violin(..., col="transparent", varwidth=FALSE,
      box.ratio=box.ratio)
    panel.bwplot(..., fill=NULL, box.ratio=.15)})
```

A simple “violin” panel.



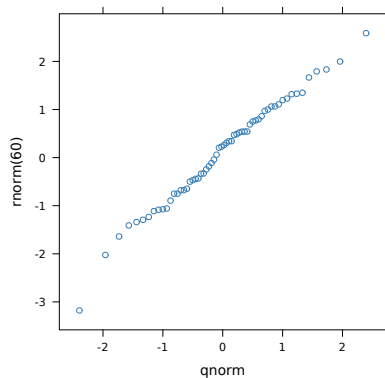
3.4 Quantile and related probability plots

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna

³P Kampstra. “Beanplot: A Boxplot Alternative for Visual Comparison of Distributions”. In: *Journal of Statistical Software* 28 (2008). URL: <http://www.jstatsoft.org/v28/c01>.

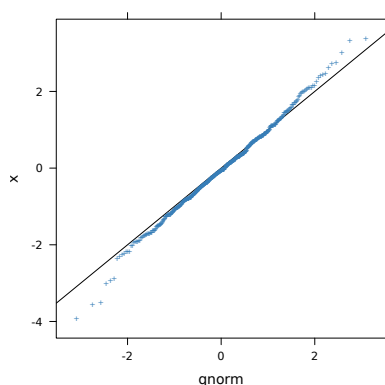
fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

```
qqmath(~ rnorm(60))
```



A simple quantile plot of 60 gaussian variates. The theoretical quantiles of an $\mathcal{N}(0; 1)$ are shown on the x -axis.

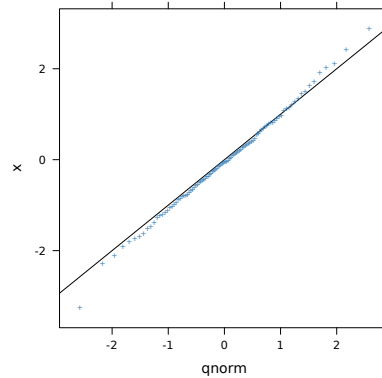
```
x <- rt(500, df=20)
qqmath(~ x, pch="+", abline=c(0,1), dist=qnorm)
```



This is basically asking to draw the same QQ-plot, but with a higher number of data points coming from a different distribution (Student $t(20)$). A reference line is added to facilitate comparison. This basically shows how closely the t -distribution can be approximated by an $\mathcal{N}(0; 1)$ when n gets very large.

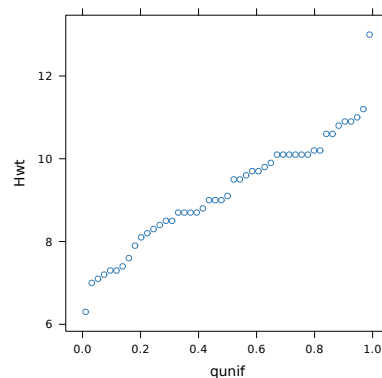
```
qqmath(~ x, pch="+", abline=c(0,1), dist=qnorm,
       f.value=ppoints(100))
```

Same as above but subsampling data points.



```
qqmath(~ Hwt, data=cats, subset=Sex == "F", dist=qunif)
```

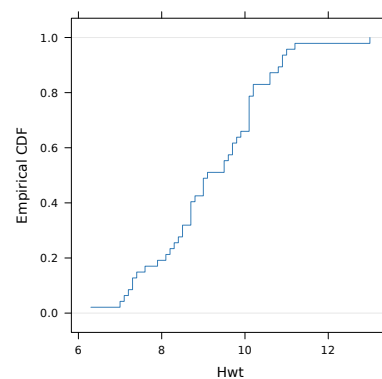
Here is one way to show the cumulative distribution function (CDF) of some sample dataset. Note that we need to explicitly need to ask `qqmath` to use the Uniform distribution as a reference.



`cats`. The heart and body weights of samples of male and female cats used for *digitalis* experiments. The cats were all adult, over 2 kg body weight.

```
ecdfplot(~ Hwt, data=cats, subset=Sex == "F")
```

An alternative way of plotting the empirical CDF is to rely on the `ecdfplot` function from the `latticeExtra` package.

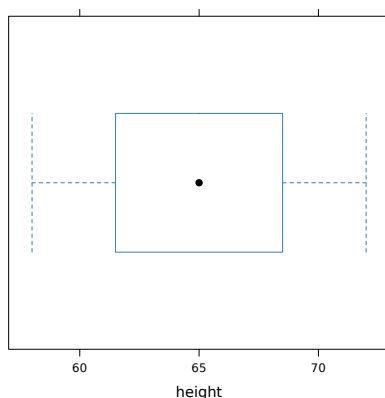


3.5 Boxplots

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

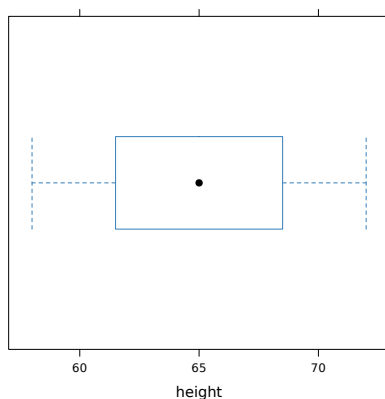
```
bwplot(~ height, data=women)
```

women. This data set gives the average heights and weights for American women aged 30-39.



Boxplots are shown in horizontal mode by default. Internally, the `boxplot.stats` function is used so that hinges corresponds to the first and third quartile while notches extends to $\pm 1.58\text{IQR}/\sqrt{n}$. It provides a visual summary analogous to Tukey's five number (see `fivenum`).

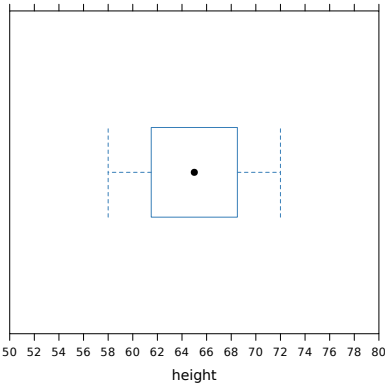
```
bwplot(~ height, data=women, box.ratio=.5)
```



This is the same picture but with a thinner box. When there are several boxplots to draw side by side, this might be useful.

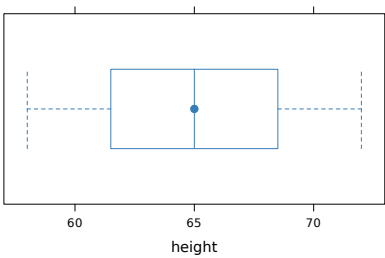
```
bwplot(~ height, data=women, box.ratio=.5,
       scales=list(x=list(limits=c(50, 80), at=seq(50, 80, by=2)))))
```

A more detailed x-scale has been created (without prejudice to its usefulness) by simply updating the `scales=` argument.



```
bwplot(~ height, data=women, aspect=.5,
       panel=function(x, ...) {
         panel.bwplot(x, pch="|", ...)
         panel.points(mean(x), 1, pch=19, cex=1)
       })
```

It is possible to alter the way boxplot are drawn, but also the statistical summaries that are displayed. For instance, in the above code, we computed the mean (drawn as a vertical bar inside the box) in addition to the median. Of note, if there are missing values, we should add `na.rm=TRUE` when calling `mean(x)`.

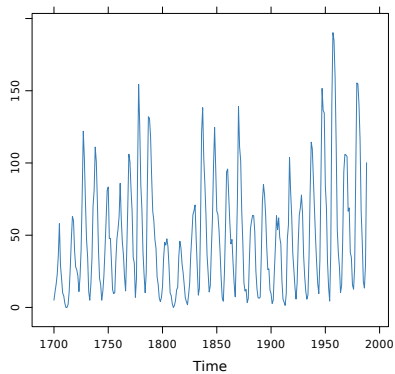


3.6 Time series

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

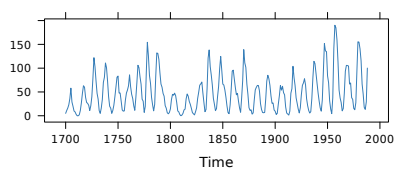
```
xyplot(sunspot.year)
```

sunspot.year. Yearly numbers of sunspots from 1700 to 1988.



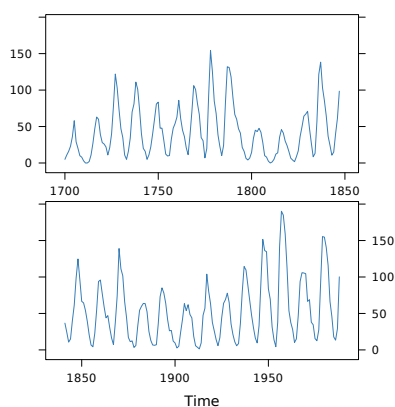
A simple time-series is displayed as a lineplot, but taking care of arranging the x -axis for time measurements.

```
xyplot(sunspot.year, aspect=.3, scales=list(y=list(rot=0)))
```



A more comprehensive picture after aspect ratio has been lowered so as to better highlight the cyclic component.

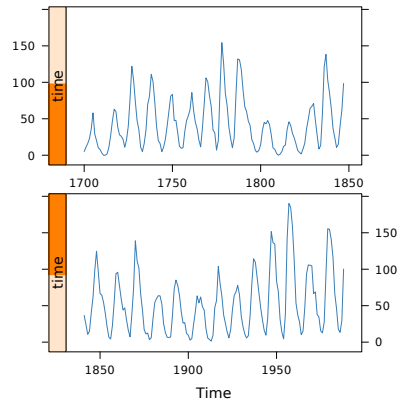
```
xyplot(sunspot.year, strip=FALSE, cut=list(number=2, overlap=.05))
```



The same time series cut into two pieces with 5% of overlap.

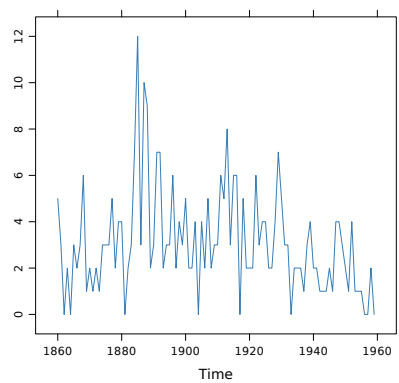
```
xyplot(sunspot.year, strip=FALSE, strip.left=TRUE,
       cut=list(number=2, overlap=.05))
```

Now we highlight explicitly that the two series of measurements are related by adding a colored ribbon denoting time period.



```
xyplot(zoo(discoveries))
```

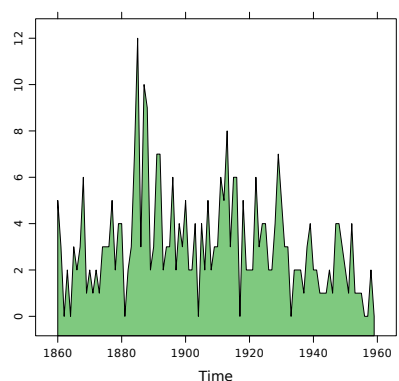
The `zoo` has to be loaded before using the above command. Briefly, it takes care of handling time-series data correctly, and it is interfaced to lattice's `xyplot` as an S3 method (see `xyplot.zoo`).



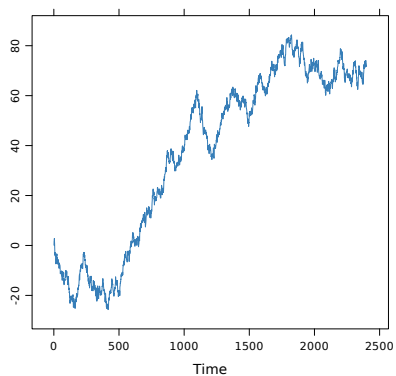
discoveries. The numbers of "great" inventions and scientific discoveries in each year from 1860 to 1959.

```
xyplot(zoo(discoveries), panel=panel.xyarea)
```

It is possible to add a shaded area by using a specific panel function from the `latticeExtra` package.



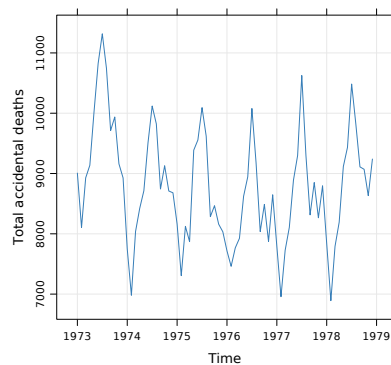
```
xt <- ts(cumsum(rnorm(200 * 12)))
p <- xyplot(xt)
```



The lattice package works with `ts` objects too.

```
xt <- zoo(accdeaths)
xyplot(xt, type=c("l", "g"), ylab="Total accidental deaths")
```

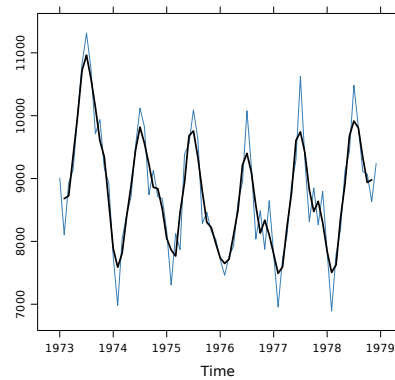
accdeaths. A regular time series giving the monthly totals of accidental deaths in the USA.



Another regular time series is.

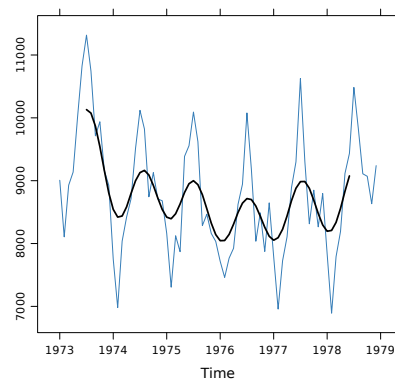

```
xyplot(xt,
  panel=function(x, y, ...) {
    panel.xyplot(x, y, ...)
    panel.lines(rollmean(zoo(y, x), 3), lwd=2, col=1)
  })
```

The same dataset with a rolling mean (of width 3).



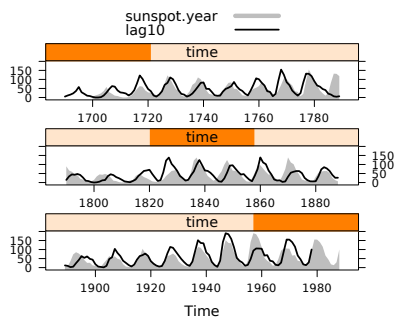
```
xyplot(xt) +
  layer(panel.tskernel(x, y, c=3, col=1, lwd=2))
```

Discrete symmetric smoothing kernels, available in `latticeExtra`, can be used instead of a rolling mean. Here an approximate gaussian filter was used to highlight the seasonal component. Of note, the above R command makes use of a `layer`, see also Chapter 6



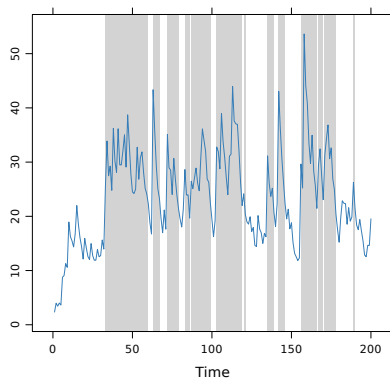
```
xyplot(ts.union(sunspot.year, lag10=lag(sunspot.year, 10)),
       superpose=TRUE, panel=panel.superpose,
       panel.groups=function(..., group.number) {
         if (group.number == 1) panel.xyarea(...)
         else panel.xyplot(...)
       }, border=NA, cut=list(n=3, overlap=0), aspect="xy",
       par.settings=simpleTheme(col=c("grey", "black"), lwd=c(5, 2)))
```

sunspot.year. Yearly numbers of sunspots between 1700 and 1988.



More complex arrangement can be done, again with the `latticeExtra` panel function. Here, yearly numbers of sunspots are shown together with a lagged version (10 years).

```
flow <- ts(filter(rlnorm(200, mean = 1), 0.8, method = "r"))
xyplot(flow,
       panel=function(x, y, ...) {
         panel.xblocks(x, y > mean(y), col="lightgray")
         panel.xyplot(x, y, ...)
       })
```



blabla blabla

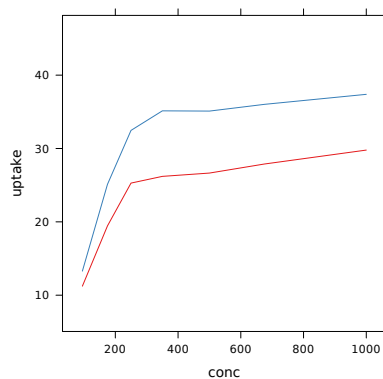
CHAPTER 4

Two-way graphics

This chapter covers graphical displays for two-way relationships, possibly by considering additional variables (numerical or categorical) to highlight ternary relationships. Two-way graphics are not limited to numerical variables as we may be interested in showing the relationships between two ordered categorical variables, two unordered or “nominal” variables. Moreover, as stated above, categorical or discretized variables can be used to provide additional information on top of a line- or scatter-plot by simply varying point size, point or line colors, and so on. Of course, we could extend this idea to the point of displaying sixth dimensions in a single graph (e.g., using symbol with varying length and width, color, and shading pattern). But, such a complex graph would likely be poorly readable and uninformative in the end. So, this chapter basically provides necessary R command to create line-plot, scatter-plot, bar-plot, dot-plot

4.1 Lineplots

```
xyplot(uptake ~ conc, data=C02, groups=Treatment, type="a")
```



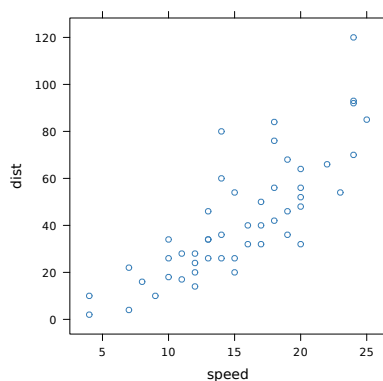
Automatic averaging.

4.2 Scatterplots

The basic R command for displaying a two-way scatterplot is `xyplot`. A command like `xyplot(y ~ x)` will produce a 2D plot almost identical to what would be obtained using base graphics, `plot(x, y)`. However, the default layout is generally better and it looks more pretty.

```
xyplot(dist ~ speed, data=cars)
```

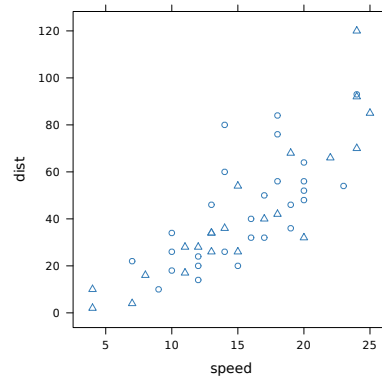
cars. The data give the speed of cars and the distances taken to stop. Note that the data were recorded in the 1920s.



This basic scatterplot shows default options when calling the `xyplot` command. The formula interface is used to plot `dist` (y-axis) as a function of `speed` (x-axis), with automatic determination of axis units.

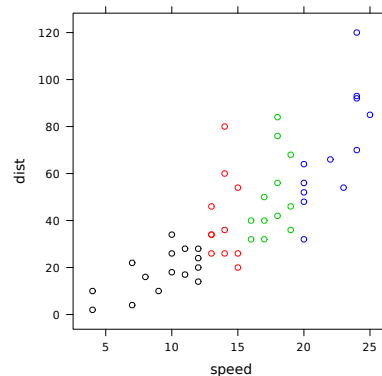
```
xyplot(dist ~ speed, data=cars, pch=rbinom(nrow(cars), 1, .5)+1)
```

We pick a random symbol ($\circ = 1, \triangle = 2$) for each observation, using the `pch=` argument. In fact, this argument is transferred to the `panel.xyplot` function that acts as the default panel function. Note that the vector of symbols should have the same length as the x and y components, otherwise recycling occurs. It is, however, not a good idea to manipulate the `pch=` (or `col=`, see below) argument, and it is better to rely on the `par.settings=` parameter since it allows to use custom themes which will facilitate the display of legend.



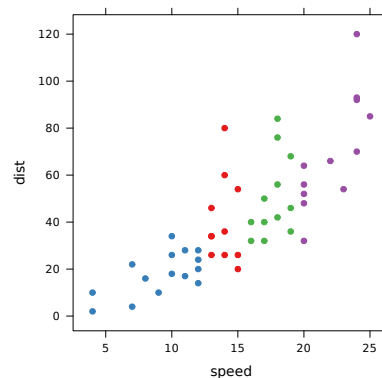
```
xyplot(dist ~ speed, data=cars,
       col=with(cars, cut(speed, breaks=quantile(speed),
                        include.lowest=TRUE)))
```

Color (`col=`) of each observation depends of the quartile they belong to. Note that passing colors this way will override default theming options.

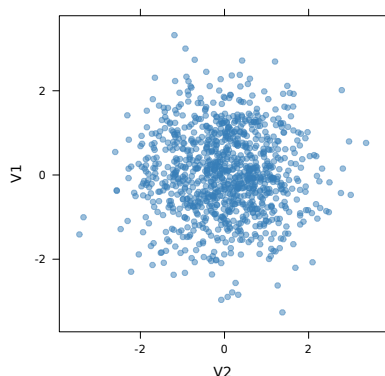


```
xyplot(dist ~ speed, data=cars, pch=19,
       groups=with(cars, cut(speed, breaks=quantile(speed),
                          include.lowest=TRUE)))
```

This is basically the same code as previously shown except that we replaced the `col=` argument by `groups=`. This has the advantage of observing the current theme, and this will further facilitate the insertion of an automatic legend.

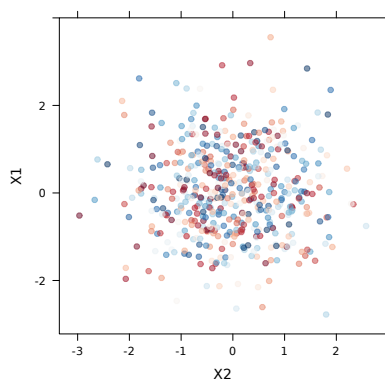


```
xy <- as.data.frame(replicate(2, rnorm(1000)))
xyplot(V1 ~ V2, data=xy, pch=19, alpha=.5)
```



When there are a high proportion of points that overlap, using transparent color may be useful. We replaced the default symbol with its filled counterpart. An equivalent way of specifying transparent color would be to use `rgb(.22, .49, .72, alpha=.5)`.

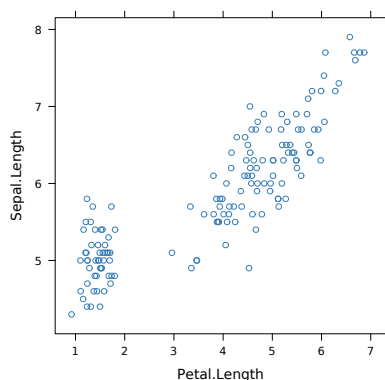
```
dat <- data.frame(replicate(2, rnorm(500)), z=sample(0:40, 500, T))
cols <- colorRampPalette(brewer.pal(11, "RdBu"))(diff(range(dat$z)))
xyplot(X1 ~ X2, data=dat, col=cols[dat$z], pch=19, alpha=.5)
```



Alpha-blending and color palette might be combined as well. Here, we used a pre-defined color scheme (Red to Blue) from the RColorBrewer package. As the selected palette has only 11 different colors, whereas the grouping factor, z, has 40 levels, we use linear interpolation to increase the number of available colors.

```
xyplot(Sepal.Length ~ Petal.Length, data=iris, jitter.x=TRUE,
       amount=.2)
```

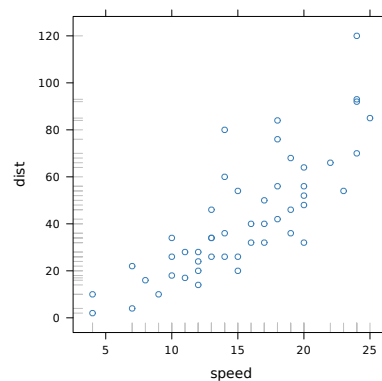
iris. This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are *Iris setosa*, *versicolor*, and *virginica*.



As an alternative to transparent colors, one may resort on “jittering”. This is also useful when not so many points are available but show few variations on one dimension. The `panel.xyplot` function uses `jitter.x=` and `jitter.y=` to vary *x* and *y* coordinates by adding a random shift drawn from a uniform distribution, $\mathcal{U}(-a, a)$, where *a* stands for the `amount=` parameter.

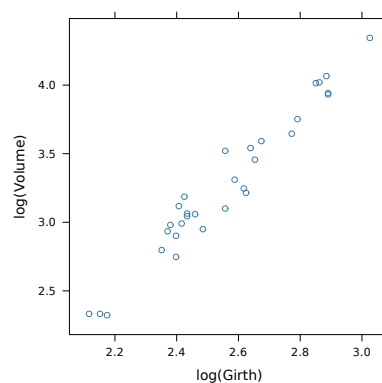
```
xyplot(dist ~ speed, data=cars,
       panel=function(x, y, ...) {
         panel.xyplot(x, y, ...)
         panel.rug(x, y, ...)
       })
```

It is possible to superimpose the univariate distribution of both series of measurement using “rug” plots. Usually, they remain quite discreet (read *non-invasive*) but provide additional information to spot possible asymmetry. We need to ask explicitly for a custom `panel`, though.



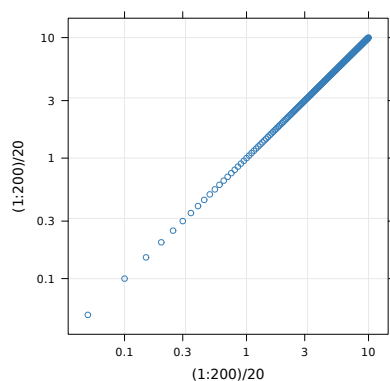
```
xyplot(log(Volume) ~ log(Girth), data=trees)
```

A simple log-log plot. Note that we would have to manually update the `scales=` component to provide more suitable annotations for the x and y -axis.



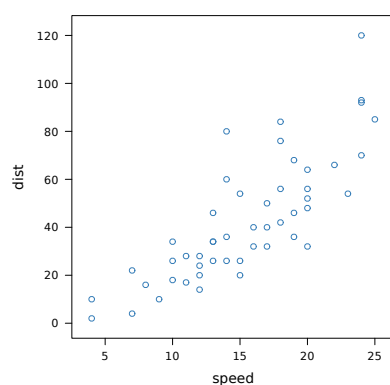
trees. This data set provides measurements of the girth, height and volume of timber in 31 felled black cherry trees. Note that girth is the diameter of the tree (in inches) measured at 4 ft 6 in above the ground.

```
xyplot((1:200)/20 ~ (1:200)/20, type=c("p", "g"),
       scales=list(x=list(log=10), y=list(log=10)),
       xscale.components=xscale.components.log10.3,
       yscale.components=yscale.components.log10.3)
```



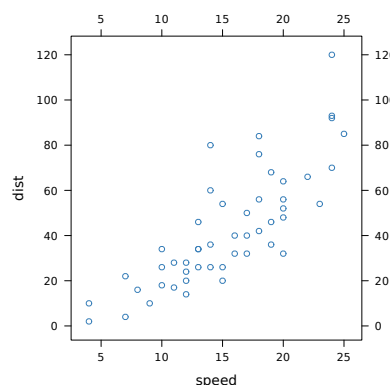
Instead of transforming variables in the formula, it is easier and safer to do this through the `scales=` parameter. The `[x|y]scale.components` are convenient functions that help to annotate axes with correct units and tick marks spacing.

```
xyplot(dist ~ speed, data=cars, scales=list(tck=c(1,0)))
```



To get rid of ticks on opposite axes, we can change default values for `tck=` in the `scales=` component. The `tck=` parameter controls the length of tick marks; however, with a vector of length 2 it can be used to deal with left/bottom and right/top axis separately.

```
xyplot(dist ~ speed, data=cars, scales=list(alternating=3))
```

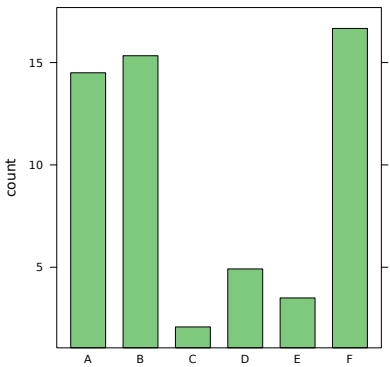


To annotate both axes, we can alter the `alternating=` parameter. In most case, however, adding grid lines in the background should provide enough information. Using `alternating=2` would reverse the annotation of axis (right/top instead of left/bottom).

4.3 Barcharts

```
spray.df <- aggregate(count ~ spray, data=InsectSprays, FUN=mean)
barchart(count ~ spray, data=spray.df)
```

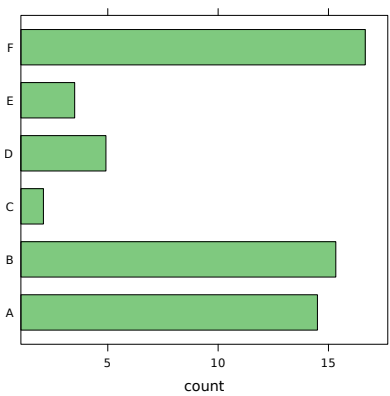
Before using barchart with one continuous and one categorical variable, we need to consider how to aggregate data, in other words what summary measure to consider.



InsectSprays. The counts of insects in agricultural experimental units treated with different insecticides.

```
barchart(spray ~ count, data=spray.df)
```

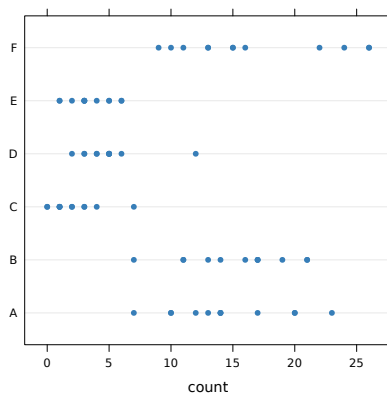
To reverse x and y axis, we just need to use the exchange the right-hand and left-hand side of the preceding formula.



4.4 Dotcharts

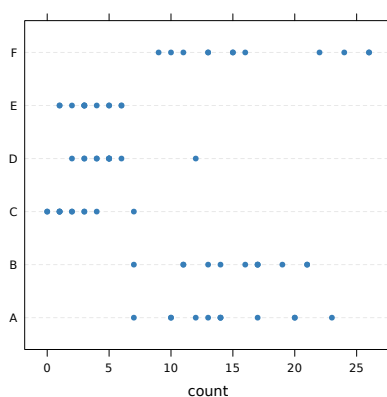
low ink-ratio

```
dotplot(spray ~ count, data=InsectSprays)
```



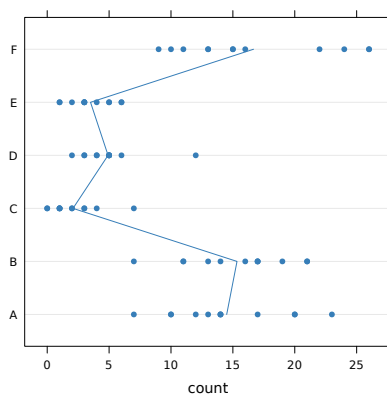
Instead of barcharts, it is usually more easy to use Cleveland's dotcharts as they allow to show individual (within level) variations.

```
dotplot(spray ~ count, data=InsectSprays, lty=2)
```



The main panel can be customized easily. For example, we can change the way horizontal lines are drawn.

```
dotplot(spray ~ count, data=InsectSprays, type=c("p", "a"))
```



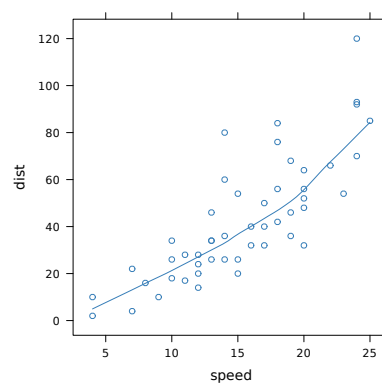
It is also possible to show aggregated data, like average count per level, using `panel.average`. The default `fun=` argument is `mean`, but we could use `median` instead.

4.5 Line fits

In this section, we discuss the addition of model fit to existing two-way graphics. For example, it may be interesting to show a regression or lowess¹ line when using `xyplot`.

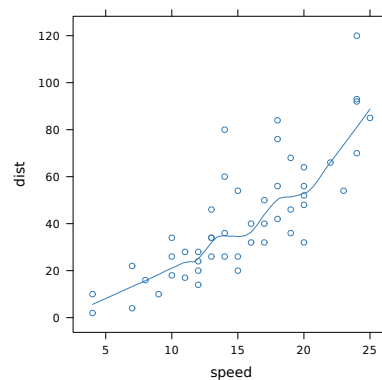
```
xyplot(dist ~ speed, data=cars, type=c("p", "smooth"))
```

An adaptive loess smoother superimposed on top of a standard scatterplot.



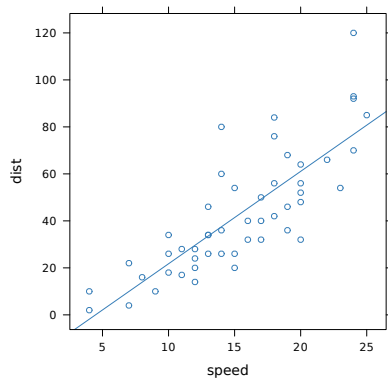
```
xyplot(dist ~ speed, data=cars, type=c("p", "smooth"), span=1/3)
```

Window span can be controlled using the `span=` argument, where lower value means more sensitivity to local variations.



¹WS Cleveland. "Robust locally weighted regression and smoothing scatterplots". In: *Journal of the American Statistical Association* 74 (1979), pp. 829–836.

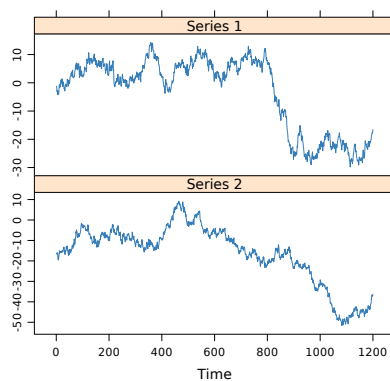
```
xyplot(dist ~ speed, data=cars, type=c("p", "r"))
```



Regression fit can be shown using the same idea, through the `type=` argument.

4.6 Time series

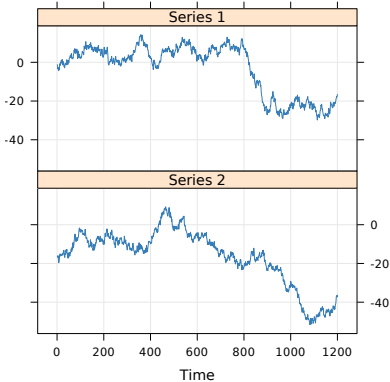
```
xt <- ts(matrix(cumsum(rnorm(200 * 12))), ncol=2))
xyplot(xt)
```



blabla blabla.

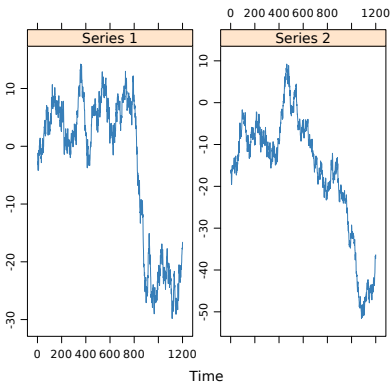
```
xyplot(xt, scales=list(y="same"), type=c("l","g"))
```

blabla blabla.



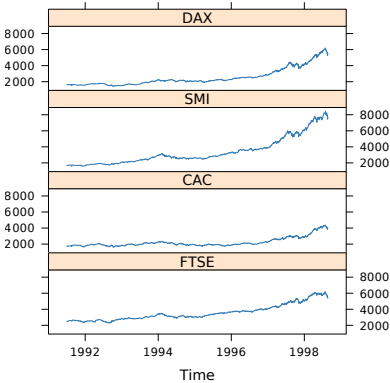
```
xyplot(xt, layout=c(2,1))
```

blabla blabla.



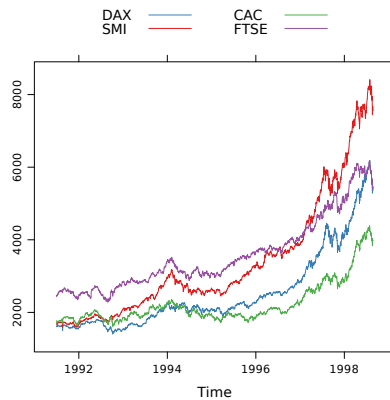
```
xyplot(EuStockMarkets, scales=list(y="same"))
```

blabla blabla.



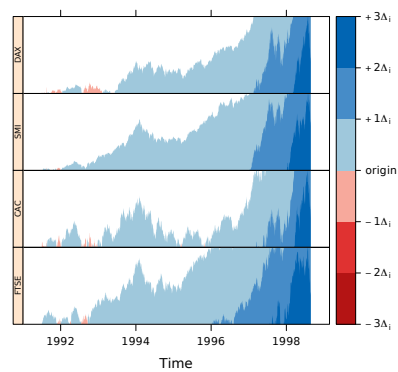
EuStockMarkets. Contains the daily closing prices of major European stock indices: Germany DAX (Ibis), Switzerland SMI, France CAC, and UK FTSE. The data are sampled in business time, i.e., weekends and holidays are omitted.

```
xyplot(EuStockMarkets, superpose=TRUE, auto.key=list(columns=2))
```



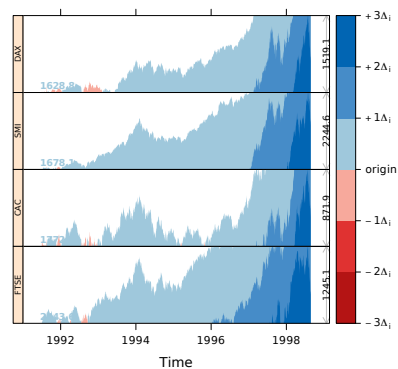
blabla blabla.

```
horizonplot(EuStockMarkets, colorkey=TRUE)
```



blabla blabla.

```
horizonplot(EuStockMarkets, colorkey=TRUE) + infolayers
```



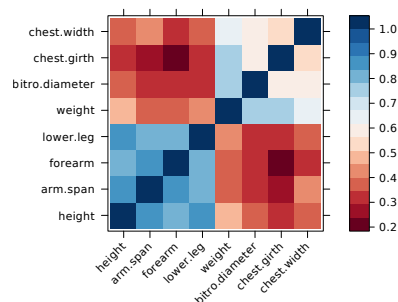
blabla blabla.

4.7 Level plot

Level plot can be used to display data summaries as “heatmap”. Here are two examples:

```
data(Harman23.cor)
levelplot(Harman23.cor$cov, scales=list(x=list(rot=45)),
          xlab="", ylab="")
```

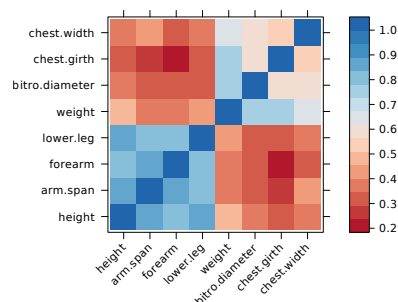
Level plots can be used to display correlation matrix in a concise way (not unlike `symnum`).



Harman23.cor. A correlation matrix of eight physical measurements on 305 girls between ages seven and seventeen. HH Harman. *Modern Factor Analysis*. Third ed. Table 2.3. University of Chicago Press, 1976

```
cols <- colorRampPalette(brewer.pal(8, "RdBu"))
levelplot(Harman23.cor$cov, scales=list(x=list(rot=45)),
          xlab="", ylab="", col.regions=cols)
```

The default color scheme can be changed easily. Here, we are using a Red-Blue color palette, from the `RColorBrewer` package. (Note that it introduces little changes compared to the preceding plot because a custom theme is currently in use for the whole textbook.)



Sometimes, it is helpful to reorder the entries in a correlation matrix or a two-way cross-classification. This can be done using `order` or custom packages, like

It is also possible to combine a dendrogram with a distance matrix, and add this clustering information to an heatmap. Here is an illustration using the `mtcars` dataset and the `dendrogramGrob` function from `latticeExtra`.

CHAPTER 5

Multi-way graphics

This chapter focus on multi-variable displays, where usually two-way graphics are conditioned on values taken by one or more variables, or a combination thereof. These so-called “displays” are very good at conveying information about trend or variation between two numerical variables across the levels of a third factor.

5.1 Parallel displays**5.2 Scatterplot matrix****5.3 Three-way tabular data****5.4 N-way data**

CHAPTER 6

Customizing theme and panels

The Hmisc plotting functions

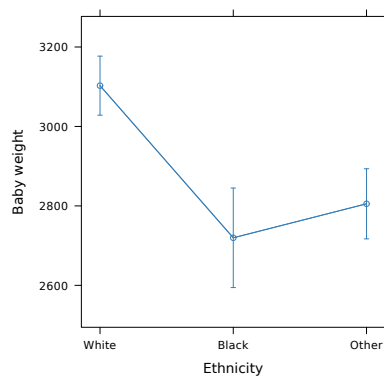
7.1 Two-way graphics

Hmisc provides automatic labelling of curves or levels of grouping factor, which are used as in standard lattice graphics (`groups=`), without the need to rely on the `directlabels` package.

Let us assume that we defined the following helper functions to compute standard error of a mean and the corresponding 95% confidence interval:

```
se <- function(x) sd(x)/sqrt(length(x))
f <- function(x) c(mean = mean(x),
  lwr = mean(x) - 1.96 * se(x),
  upr = mean(x) + 1.96 * se(x))
```

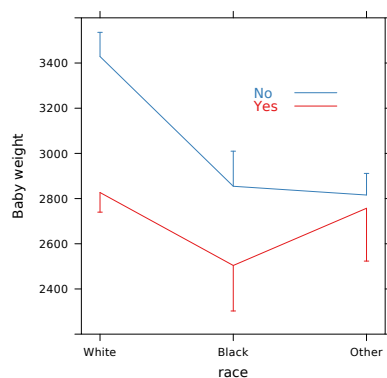
```
d <- with(birthwt, summarize(bwt, race, f))
xYplot(Cbind(bwt, lwr, upr) ~ numericScale(race, label = "Ethnicity"),
  data = d, type = "b", keys = "lines",
  ylim = range(apply(d[, 3:4], 2, range)) + c(-1,1) * 100,
  scales = list(x = list(at = 1:3, labels = levels(d$race))))
```



The race variable needs to be converted to a numerical variable for proper scaling on the x-axis. Note that labeling of the x-axis can be done at the same time. The `keys=` argument controls the type of drawing, while `type=` behaves similarly to the lattice plotting parameter.

Here is another example using a grouping factor.

```
d <- with(birthwt, summarize(bwt, llist(race, smoke), f))
xYplot(Cbind(bwt, lwr, upr) ~ numericScale(race), groups = smoke,
  data = d, type = "l", keys = "lines", method = "alt bars",
  ylim = c(2200, 3600),
  scales = list(x = list(at = 1:3, labels = levels(d$race))))
```



bla bla.

The ggplot2 package

8.1 Why ggplot2

How does ggplot2 differ from lattice? Both packages rely on the `grid` system. However, unlike lattice, ggplot2 makes heavy use of the so-called “Grammar of Graphics” approach.¹

8.2 Core components of a ggplot graphic

¹L Wilkinson. *The Grammar of Graphics*. Springer, 2005.

Interactive and dynamic displays

9.1 Exploratory data analysis

9.2 Brushing and linking

9.3 The ggobi toolbox

Index

abline, 18, 19	group.number, 26	quantile, 29
adj, 14	groups, 28, 29, 45, 46	qunif, 19
aggregate, 9, 33		
alpha, 12, 30	histogram, 13–15	range, 30
alternating, 32	horizonplot, 38	rbinom, 29
amount, 30	horizontal, 12	ref, 16
args, 15		replace, 11, 13
as.character, 14	if, 26	replicate, 30
as.numeric, 8	include.lowest, 8, 29	rgb, 30
aspect, 11, 12, 21, 22, 26		rlnorm, 26
at, 21, 45, 46	jitter.data, 11, 12	rnorm, 15, 18, 24, 30, 36
auto.key, 38	jitter.x, 30	rollmean, 25
	jitter.y, 30	rot, 22, 39
barchart, 33		rt, 18
border, 14, 15, 26	labels, 45, 46	
box.ratio, 17, 20, 21	lag, 26	sample, 11, 13, 30
boxplot.stats, 20	layer, 25	scales, 12, 13, 21, 22, 31, 32,
breaks, 14, 29	layout, 37	37, 39, 45, 46
brewer.pal, 30, 39	levels, 45, 46	
bw, 16	limits, 21	sd, 15
by, 9, 13, 21	list, 12, 15, 21–23, 26, 32,	seg.col, 13
	37–39, 45, 46	seg.lwd, 13
c, 9, 14, 18, 19, 21, 24–26, 32,	log, 31, 32	seq, 13, 21
34–37, 45, 46	lty, 34	simpleTheme, 26
cex, 12, 14, 21	lwd, 14, 25, 26	span, 35
col, 13–15, 17, 25, 26, 29, 30		strip, 22, 23
col.regions, 39	matrix, 36	strip.left, 23
colorkey, 38	mean, 15, 21, 26, 33, 34	stripchart, 12
colorRampPalette, 30, 39	median, 34	stripplot, 11–13
columns, 38	method, 26, 45, 46	subset, 9, 19
contents, 6, 7		summary.formula, 6
cumsum, 24, 36	na.rm, 21	sunflowerplot, 13
cut, 8, 14, 22, 23, 26, 29	ncol, 36	superpose, 26, 38
cut2, 8	nint, 14	symnum, 39
	nrow, 14, 29	
data, 13, 14, 16, 17, 19–21,	number, 22, 23	table, 14
28–36, 39, 45, 46		tapply, 9
data.frame, 5, 8, 30	overlap, 22, 23, 26	tck, 32
density, 16		transform, 8
densityplot, 16	panel, 12–15, 17, 21, 23, 25,	treillis, 41
describe, 6, 8	26, 31	ts, 24, 26, 36
df, 18	panel.average, 34	ts.union, 26
diff, 14, 30	panel.bwplot, 17, 21	type, 14, 15, 24, 28, 32, 34–
dist, 18, 19, 28, 29, 31, 32, 35,	panel.groups, 26	37, 45, 46
36	panel.histogram, 14, 15	
dmath, 15	panel.lines, 25	update, 14
dotplot, 34	panel.mathdensity, 15	
draw, 12	panel.points, 21	varwidth, 17
	panel.rug, 31	
ecdfplot, 19	panel.text, 14	with, 29, 45, 46
else, 26	panel.tskernel, 25	within, 8
	panel.xblocks, 26	
f.value, 19	panel.xyarea, 23, 26	xlab, 12, 39
factor, 7, 11, 12	panel.xyplot, 25, 26, 29–31	xscale.components, 32
fill, 17	par.settings, 26, 29	xy, 12, 30
filter, 26	pch, 12, 18, 19, 21, 29, 30	xyplot, 22–26, 28–32, 35–38
fivenum, 20	plot, 14, 28	
formula, 5	plot.points, 16	ylab, 24, 39
FUN, 33	ppoints, 19	yscale.components, 32
fun, 34		yscale.components.log10.3,
function, 14, 15, 17, 21, 25,	qnorm, 18	32
26, 31	qqmath, 18, 19	
		zoo, 5, 23–25