Trellis Graphics Continued

Statistics 135

Autumn 2005

Trellis Function Structure

As seen in the examples last time, there is a basic structure to calling a trellis graphics function. It involves 3 main components.

1. Model

   This involves the response variables, predictor variables, and the conditioning variables.

   The basic form is

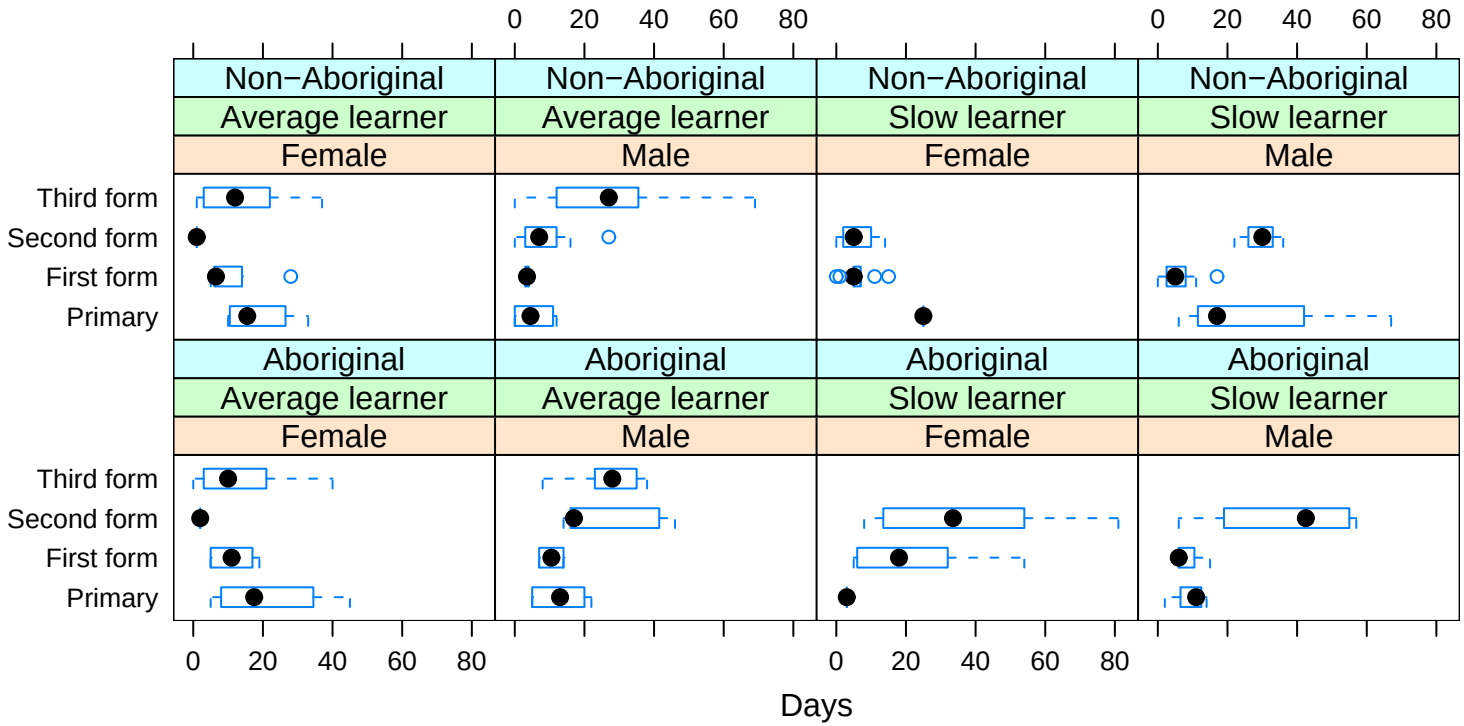
   $$\texttt{y} \sim \texttt{x} \mid \texttt{v1} * \texttt{v2} * \ldots * \texttt{vn}$$

   Where `y` is the variable for the y-axis, `x` is the variable for the x-axis, and `v1`, `v2`, ..., `vn` are the conditioning variables.

   For example

   ```
   bwplot(Age ~ Days | Sex*Lrn*Eth, data=Quine, layout=c(4,2))
   ```

which gives the plot

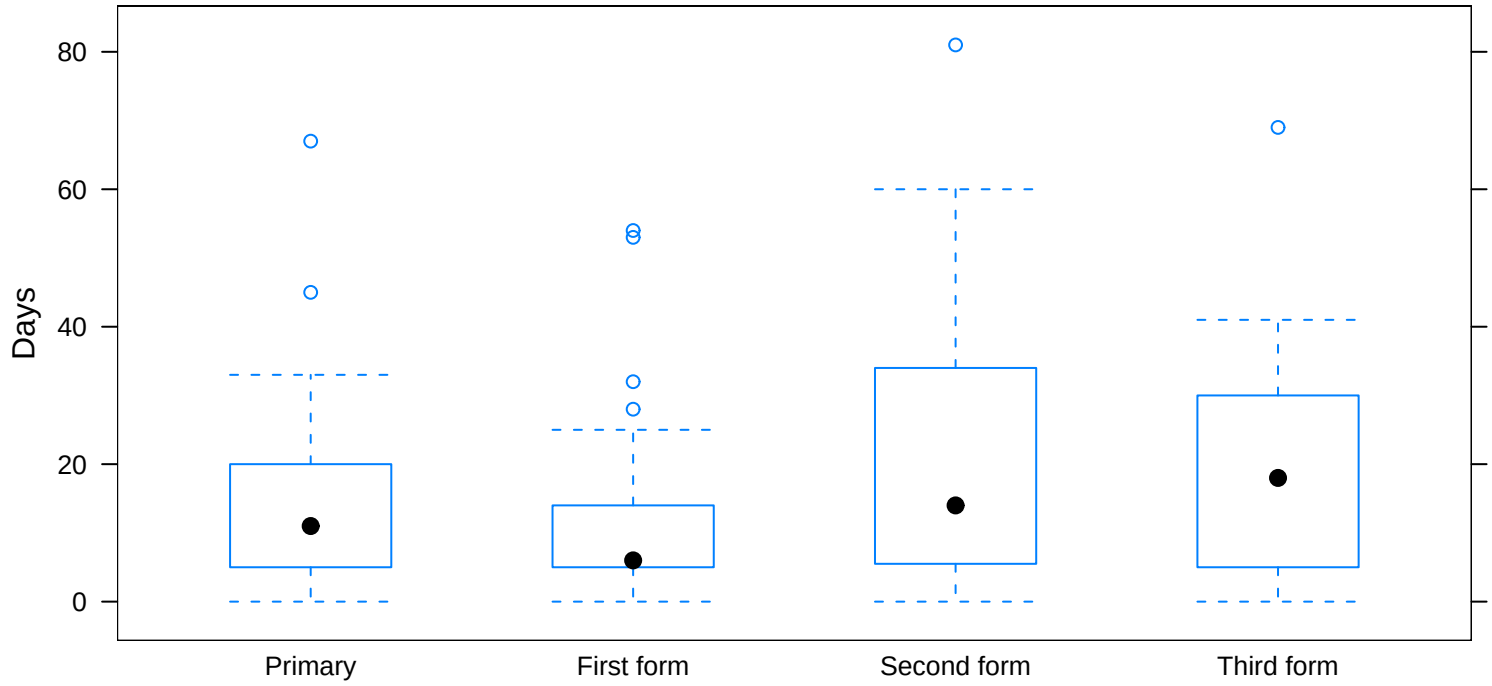


While `y` is often the response variable, it doesn't have to be, as can be seen in the above example, where `Days` is the response variable.

Note that all parts of the formula are not needed.

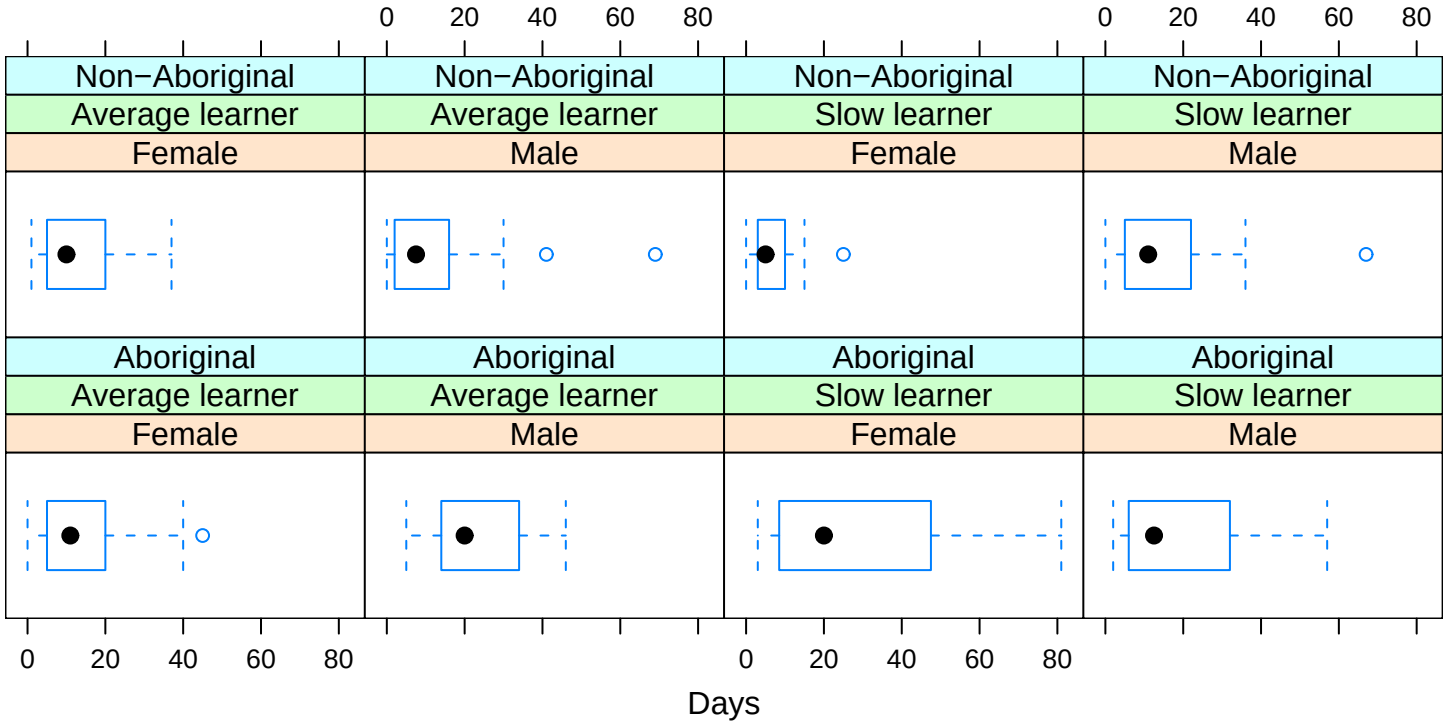
No conditioning variables

```
bwplot(Age ~ Days, data=Quine)
```



No y variable

```
bwplot(~ Days | Sex*Lrn*Eth, data=Quine, layout=c(4,2))
```



If only one variable is to be plotted, it must be preceded by a $\sim$.

For trivariate plots, the model formula is of the form

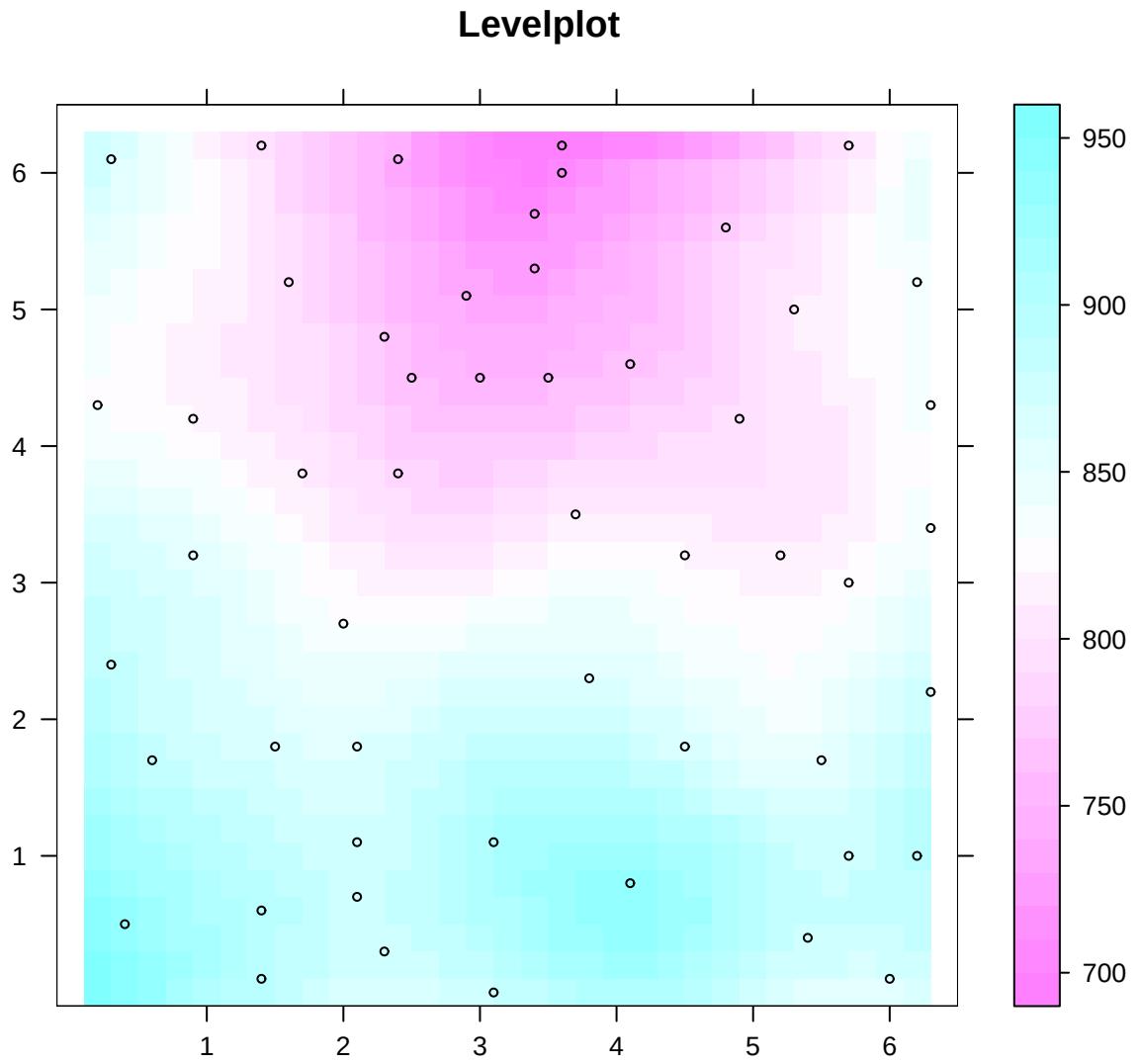
```
z ~ x * y | v1 * v2 * ... * vn
```

In the following example, a surface is estimated by `loess`, given heights (`z`) at different locations (`x, y`).

```
topo.loess <- loess(z ~ x * y, topo, degree=2, span=0.25)
topo.mar <- list(x=seq(0, 6.5, 0.2), y=seq(0, 6.5, 0.2))
topo.plt <- expand.grid(topo.mar)
topo.plt$z <- as.vector(predict(topo.loess, topo.plt))
```

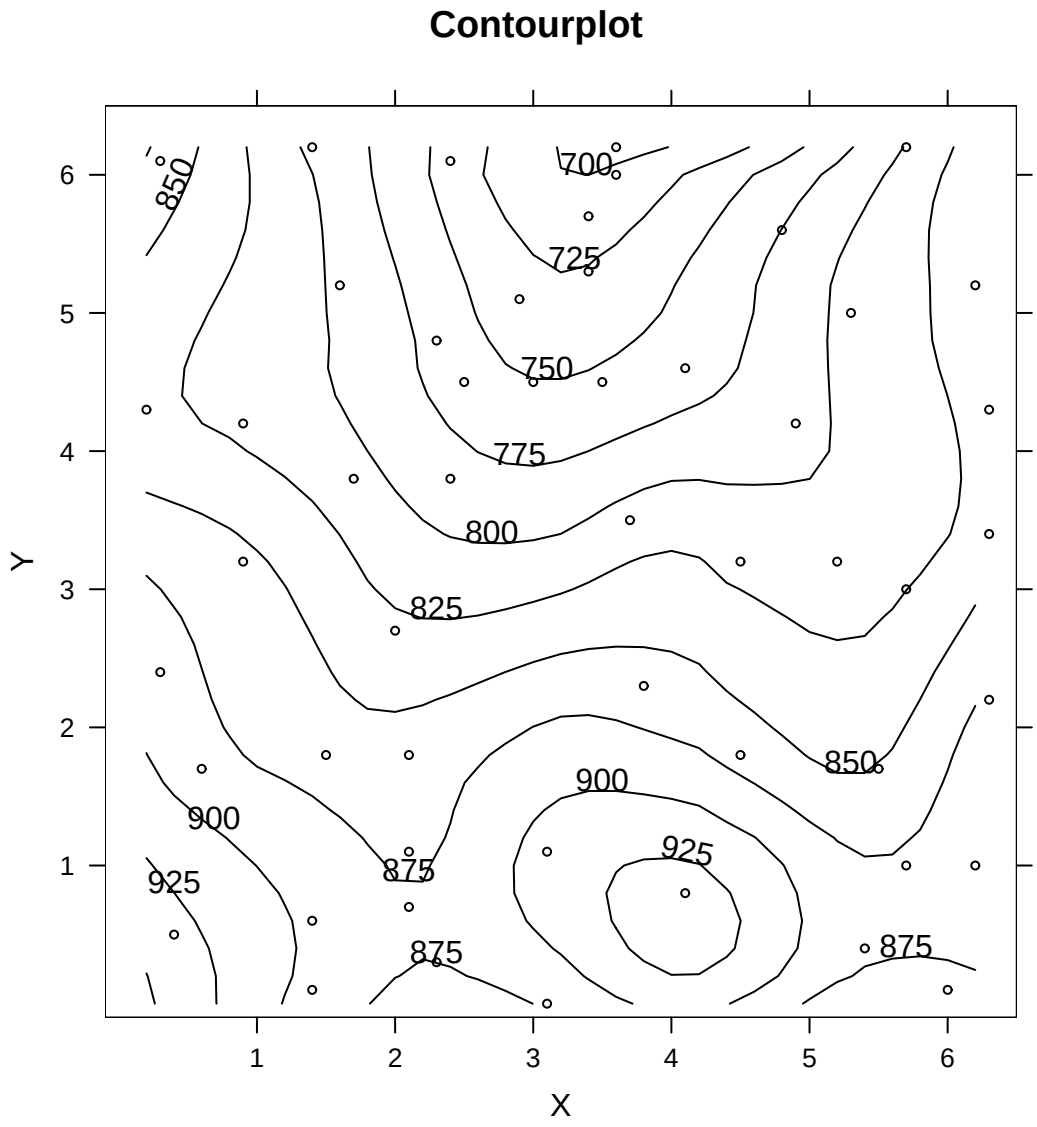
One way of displaying this surface is by a `levelplot`, which can be thought of a colour version of a contour plot.

```
levelplot(z ~ x*y, topo.plt, aspect=1,
  at=seq(690, 960, 10), xlab="", ylab="",
  main="Levelplot",
  panel = function(x, y, subscripts, ...) {
    panel.levelplot(x, y, subscripts, ...)
    panel.xyplot(topo$x, topo$y, cex=0.5, col=1)
  }
)
```


Levelplot
6
5
4
3
2
1
1
2
3
4
5
6
950
900
850
800
750
700

For a good old fashioned contourplot, try the commands

```
contourplot(z ~ x*y, topo.plt, aspect=1,
  at=seq(700, 1000, 25), xlab="X", ylab="Y",
  main="Contourplot",
  panel = function(x, y, subscripts, ...) {
    panel.contourplot(x, y, subscripts, ...)
    panel.xyplot(topo$x, topo$y, cex=0.5, col=1)
  }
)
```


Contourplot
700
725
750
775
800
825
850
875
900
925
X
Y

2. Panel components

   This describes what should be plotted for each combination of the conditioning variables.

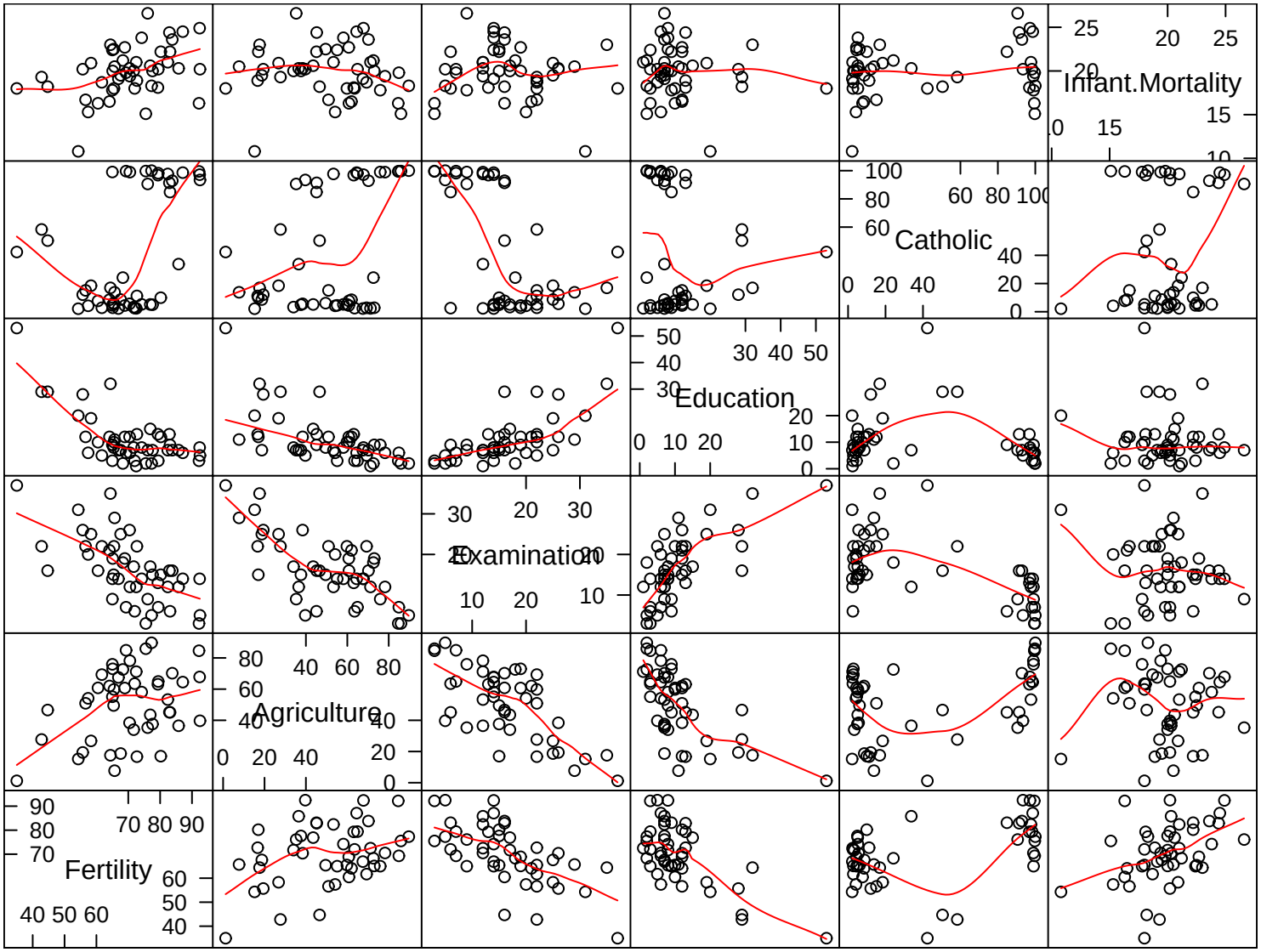
   We've seen examples of this already. See the `levelplot` and `contourplot` examples. One more is

```
splom(~ swiss, aspect = "fill", main="Trellis Approach",
  panel = function(x, y, ...) {
    panel.xyplot(x, y, ...)
    panel.loess(x, y, col=2, ...)
  }
)
```

   `splom` creates a scatter plot matrix for all of the variable in the input data frame (swiss data frame from `MASS` in the case).

   In each panel of the matrix, the data is to be plotted with a loess smooth added.

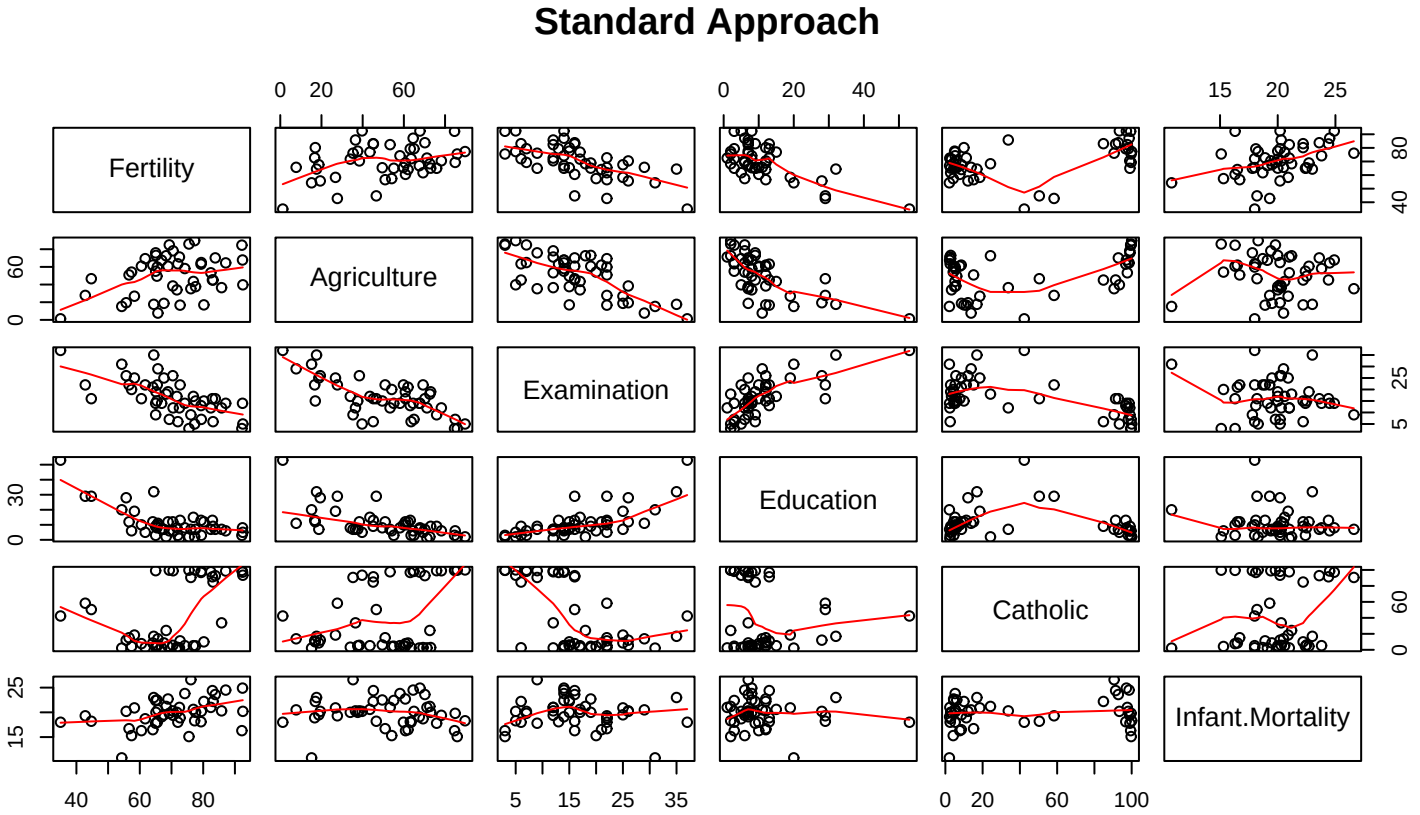
Trellis Approach



Scatter Plot Matrix

For completeness, note that a trellis graph isn't needed for this plot. It could also be done by

```
pairs(swiss, panel=panel.smooth, main="Standard Approach")
```



3. Additional options

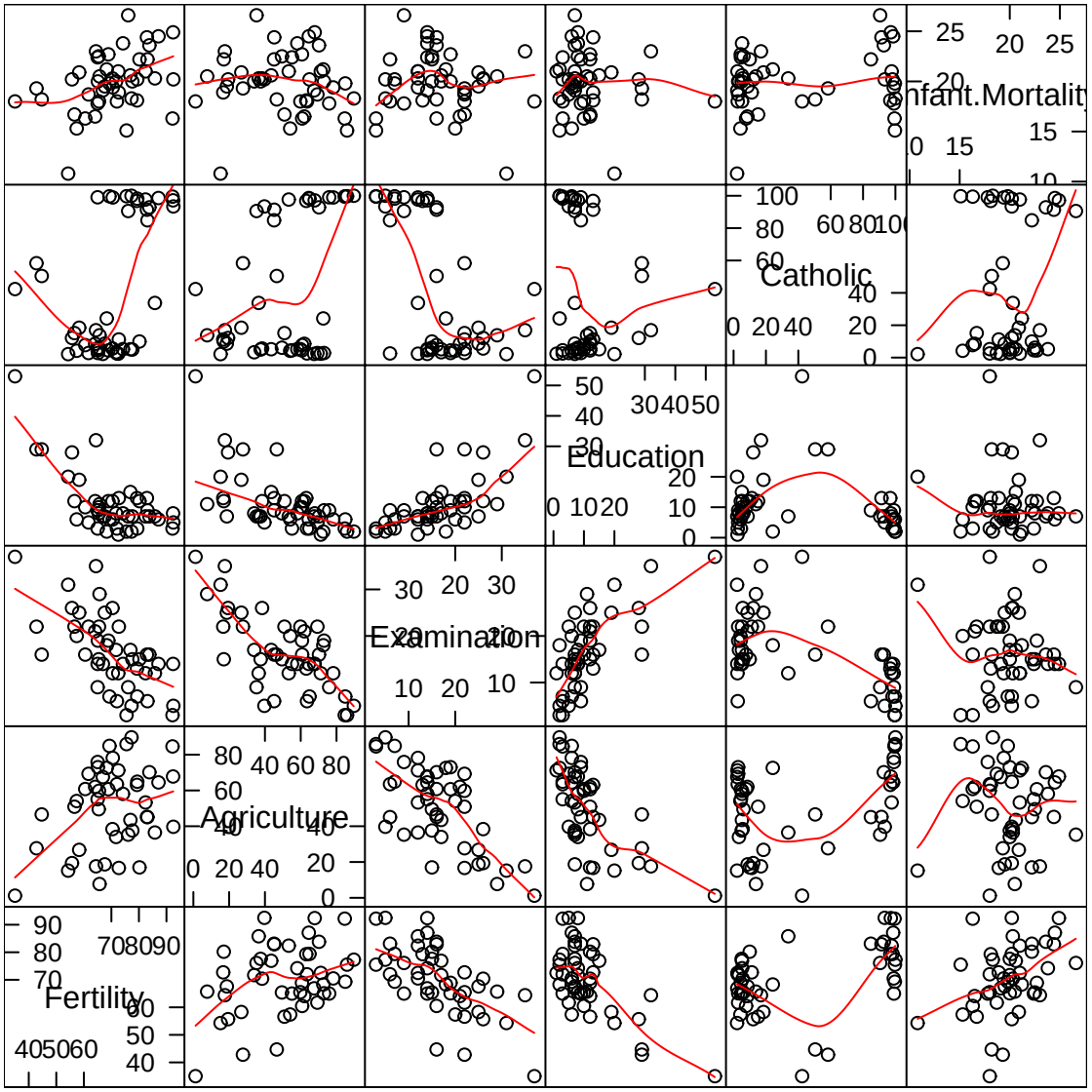
   These include axis labels, plot title, legends (`key`), group info, etc.

   These options can be general, such as those above, or they may be specific to the particular plotting function.

   For example, lets look at the option `aspect="fill"` in the `splom` function. In the previous version, the plot filled the plotting region. However if we leave this option out, we get each panel being square

```
splom(~ swiss, main='No aspect="Fill"',
  panel = function(x, y, ...) {
    panel.xyplot(x, y, ...)
    panel.loess(x, y, col=2, ...)
  }
)
```

No aspect="Fill"



Scatter Plot Matrix

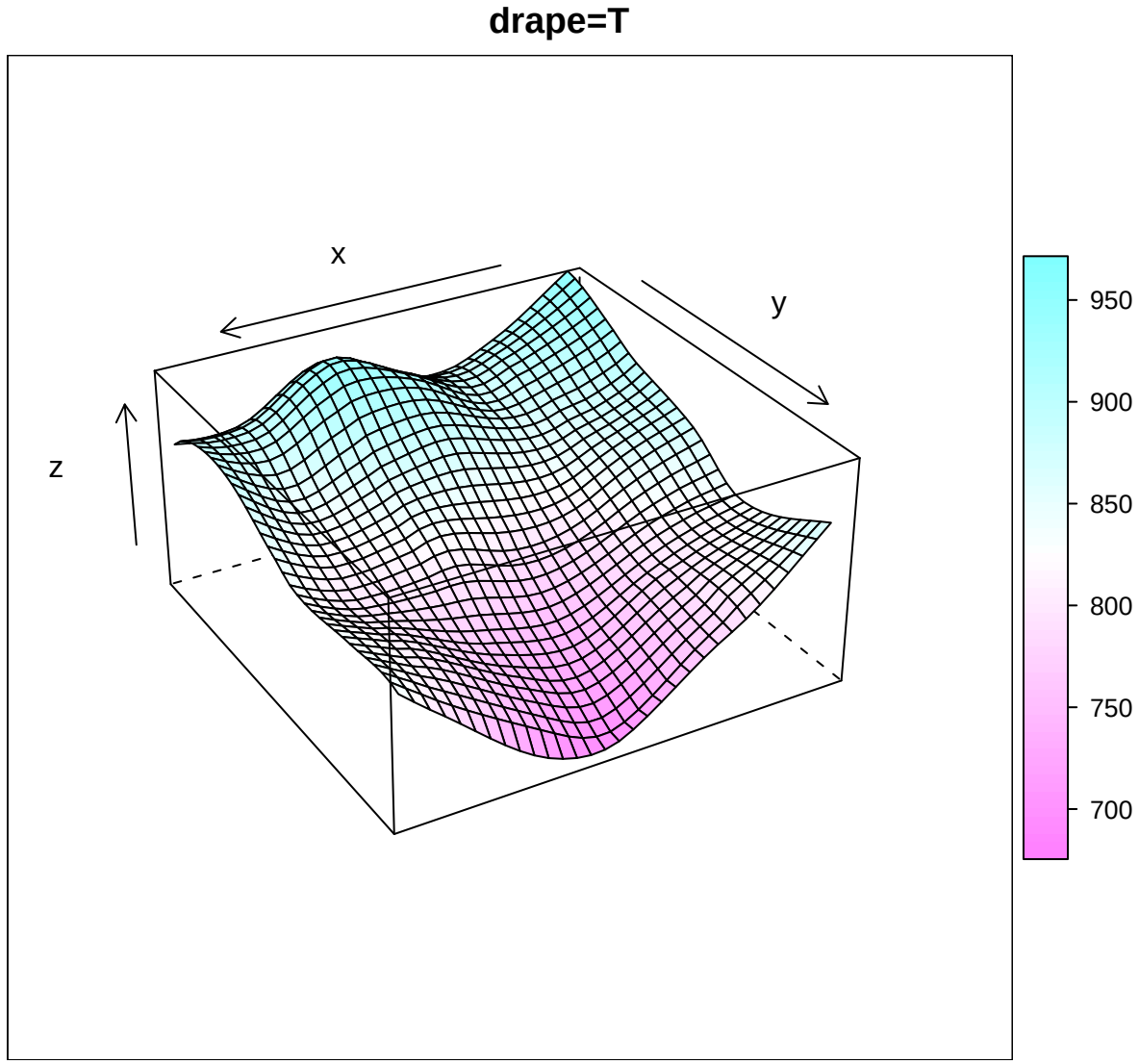
Another example of a plot specific option is the drape option for the `wireframe` plot.

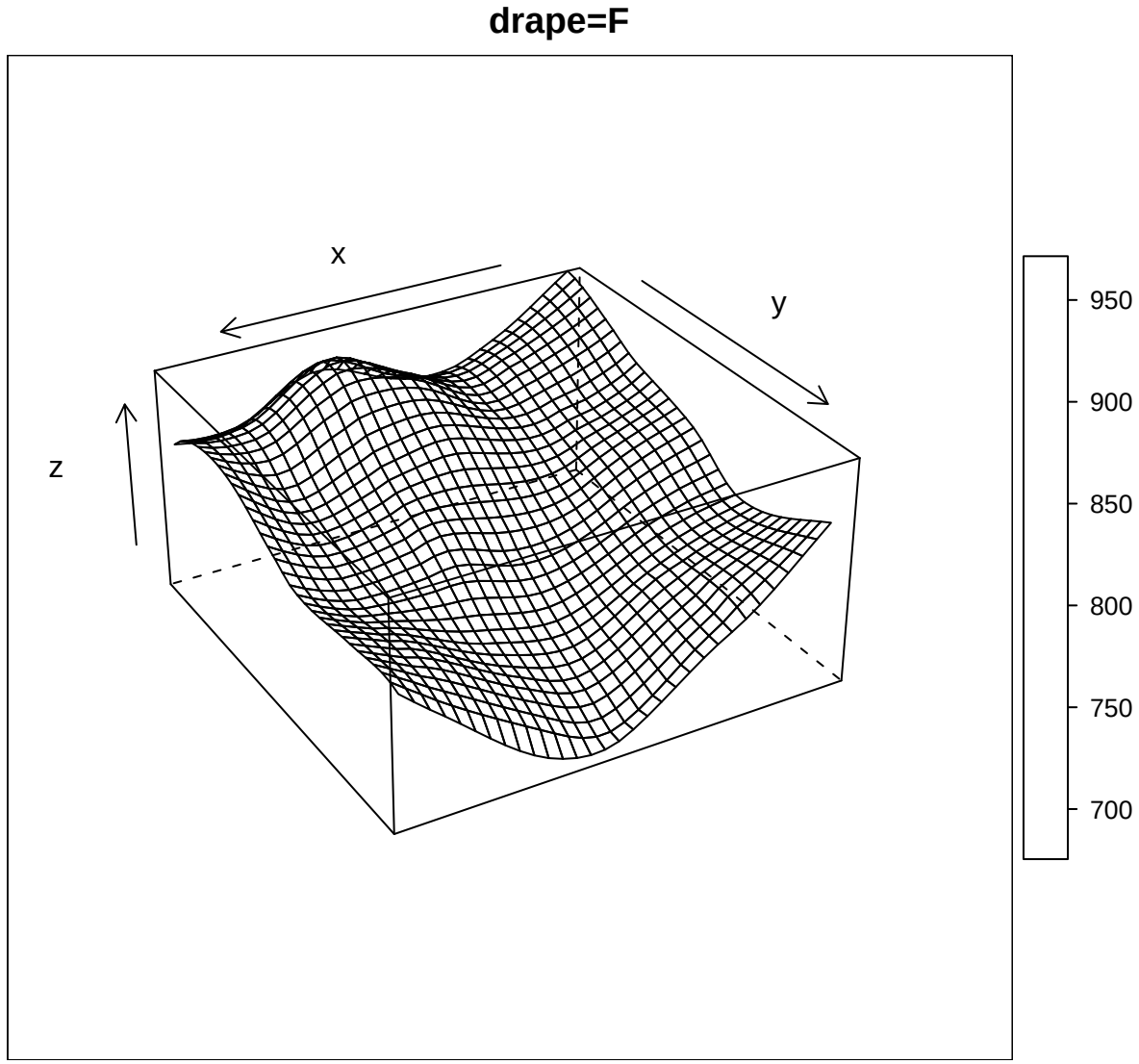
Version 1: `drape=T`

```
wireframe(z ~ x*y, topo.plt, aspect=c(1, 0.5), drape=T,
  main="drape=T",
  screen=list(z=-150, x=-60),
  colorkey = list(space="right", height=0.6)
)
```

Version 2: `drape=F`

```
wireframe(z ~ x*y, topo.plt, aspect=c(1, 0.5), drape=F,
  main="drape=F",
  screen=list(z=-150, x=-60),
  colorkey = list(space="right", height=0.6)
)
```


drape=T
x
y
z
950
900
850
800
750
700


drape=F
x
y
z
950
900
850
800
750
700

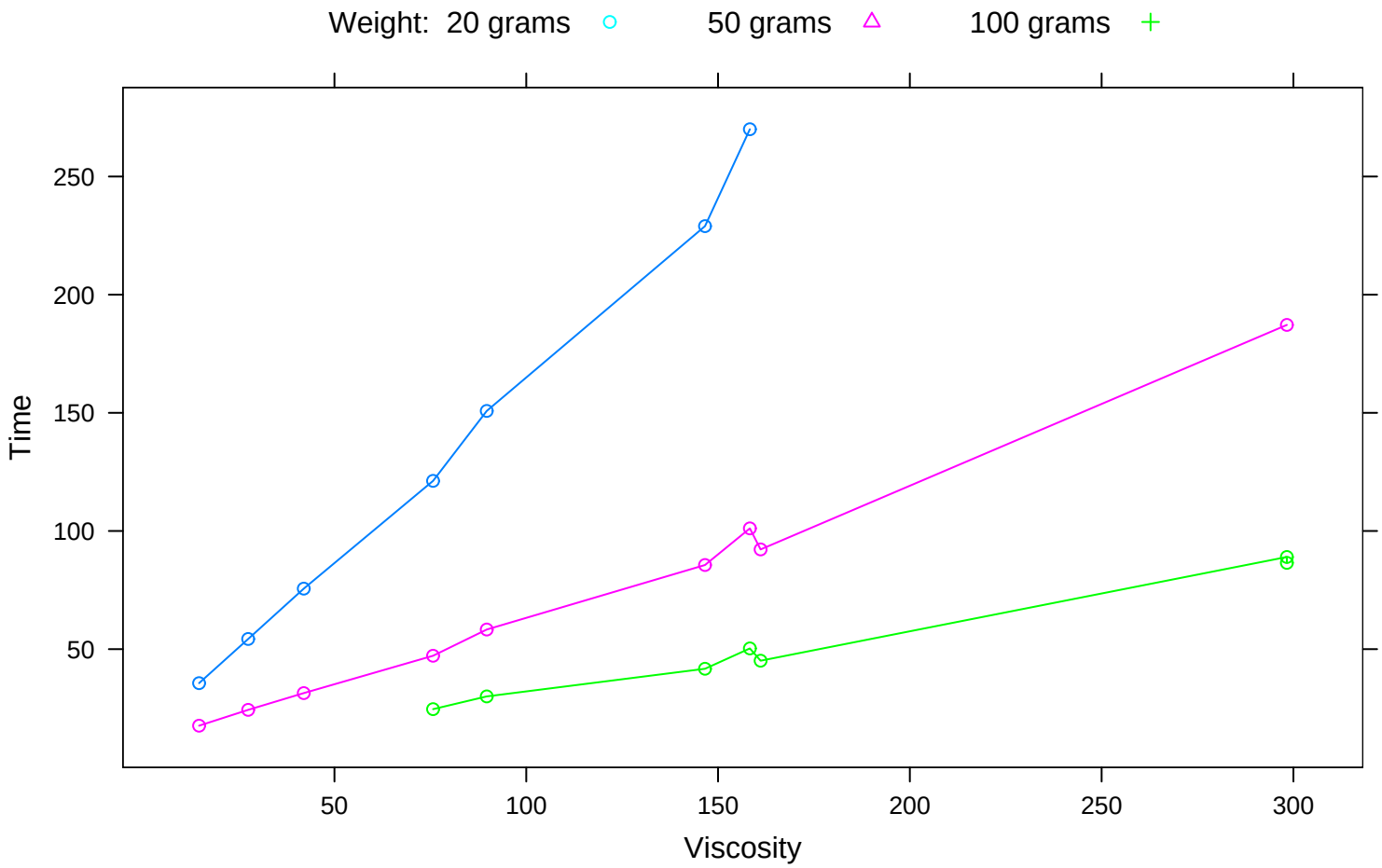
Another useful option is `key`, the trellis equivalent to `legend`. Lets look at with a `xyplot` example

```
# recode plotting symbols

sps <- trellis.par.get("superpose.symbol")
sps$pch <- 1:7
trellis.par.set("superpose.symbol",sps)

# generate plot

xyplot(Time ~ Viscosity, data=stormer, groups=Wt,
  panel = panel.superpose, type="b",
  key = list(columns=3,
      text=list(paste(c("Weight: ", "",""),
                  unique(stormer$Wt), "grams")),
      points=Rows(sps,1:3))
  )
dev.off()
```



(Note that there is a glitch in this plot. The plotting symbols should be as in the key. I think it has to do with using `postscript` as the trellis device.)

This example also exhibits the trellis equivalent to `par`, the functions `trellis.par.get`, which gets the current trellis parameters, and `trellis.par.set`, which sets new values for the trellis parameters.

In this case, we are resetting the plotting symbols.

```
> sps <- trellis.par.get("superpose.symbol")
> sps
$alpha [1] 1 1 1 1 1 1 1

$cex [1] 0.8 0.8 0.8 0.8 0.8 0.8 0.8

$col [1] "#00ffff" "#ff00ff" "#00ff00" "#ff7f00" "#007eff"
     [6] "#ffff00" "#ff0000"

$font [1] 1 1 1 1 1 1 1

$pch [1] 1 2 3 4 5 6 7
```

See `help(trellis.par.get)` for the possibilities available to change here.

This example exhibits one other approach for dealing with a conditioning variable. This is the `groups` option. It is appropriate in plots like `xyplot` or `stripplot`, which generates the appropriate plot for each level of the variable.