

Trellis graphics

Extremely useful approach for graphical exploratory data analysis (EDA)

Allows to examine for complicated, multiple variable relationships.

Types of plots

`xyplot`: scatterplot

`bwplot`: boxplots

`stripplot`: display univariate data against a numerical variable

`dotplot`: similar to stripplot

histogram

`densityplot`: kernel density estimates

barchart

`piechart`: (Not available in R)

`spiom`: scatterplot matrices

`contourplot`: contour plot of a surface on a regular grid

`levelplot`: pseudo-colour plot of a surface on a rectangular grid

wireframe: perspective plot of a surface
evaluated on a regular grid

cloud: perspective plot of a cloud of points (3D
scatterplot)

Note in S-Plus, the trellis graphics functions are contained in the library, `trellis`. This library should be loaded automatically when starting S-Plus. However if it isn't, give the command `library(trellis)`. In R, the trellis graphics features are usually not automatically loaded. To load them you need to give the command `library(lattice)`.

<digression>

`.First()` and `.Last()` functions

It is possible to configure S-Plus and R to load certain files, override defaults, etc when they start up with the use of the `.First()` function. This is similar to the use of `.cshrc/.bashrc` and `.login` files in Unix.

Any commands that you wish to run everytime you start up S-Plus and R should go in the `.First()` function. An example of a `.First()` function is

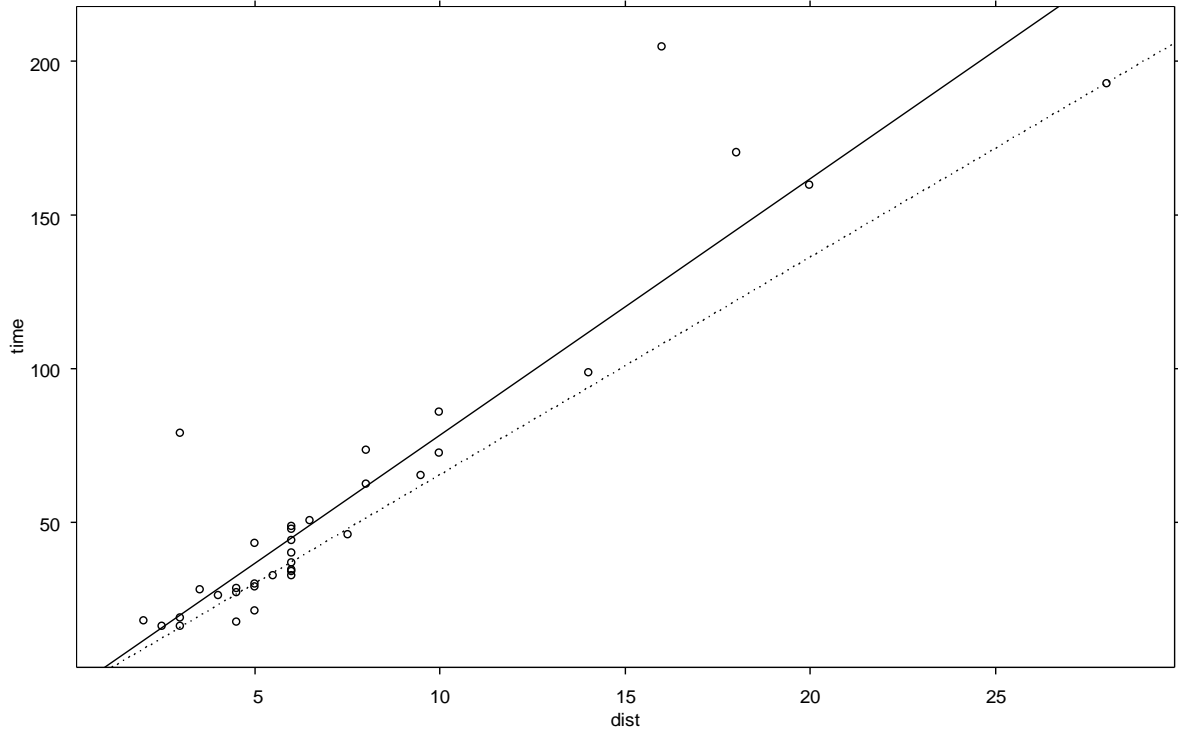
```
.First<- function() {  
  library(lattice)  
  options(contrasts=c("contr.treatment", "contr.poly"))  
}
```

So everytime that S-Plus/R is run from the directory containing this `.First()` function, these two commands will be run. The `options()` command sets the default contrasts when dealing with factors in linear models.

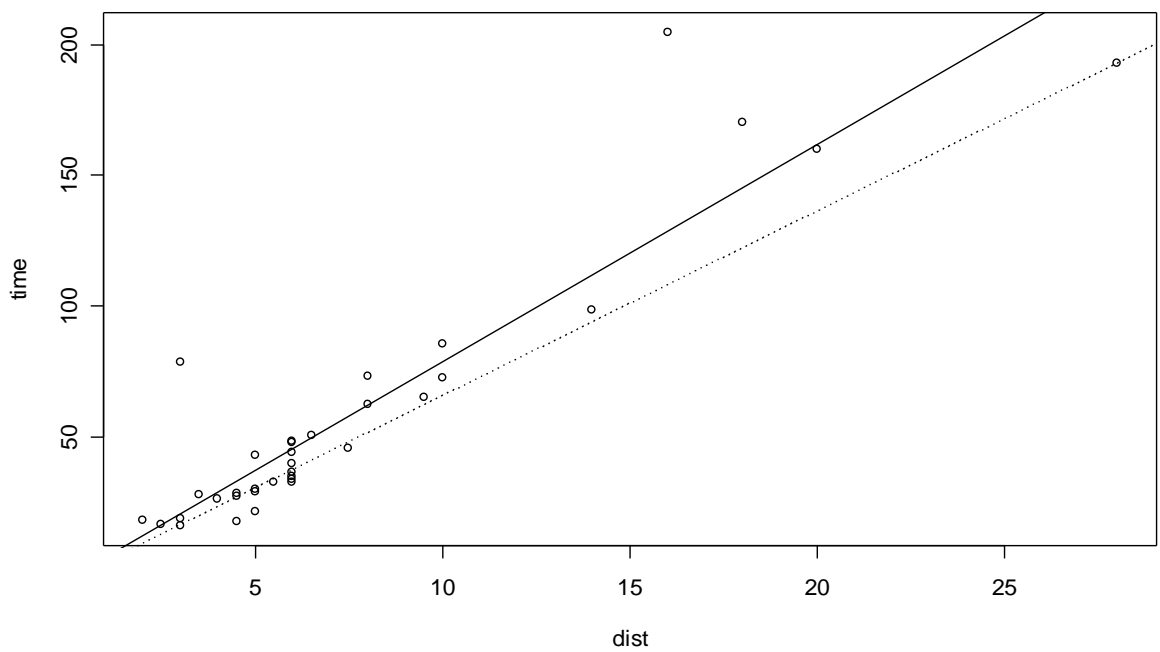
The `.Last()` function is similar to Unix's `.logout` file in that it contains the commands you want run when quitting S-Plus/R.

</digression>

Trellis graph



Usual approach



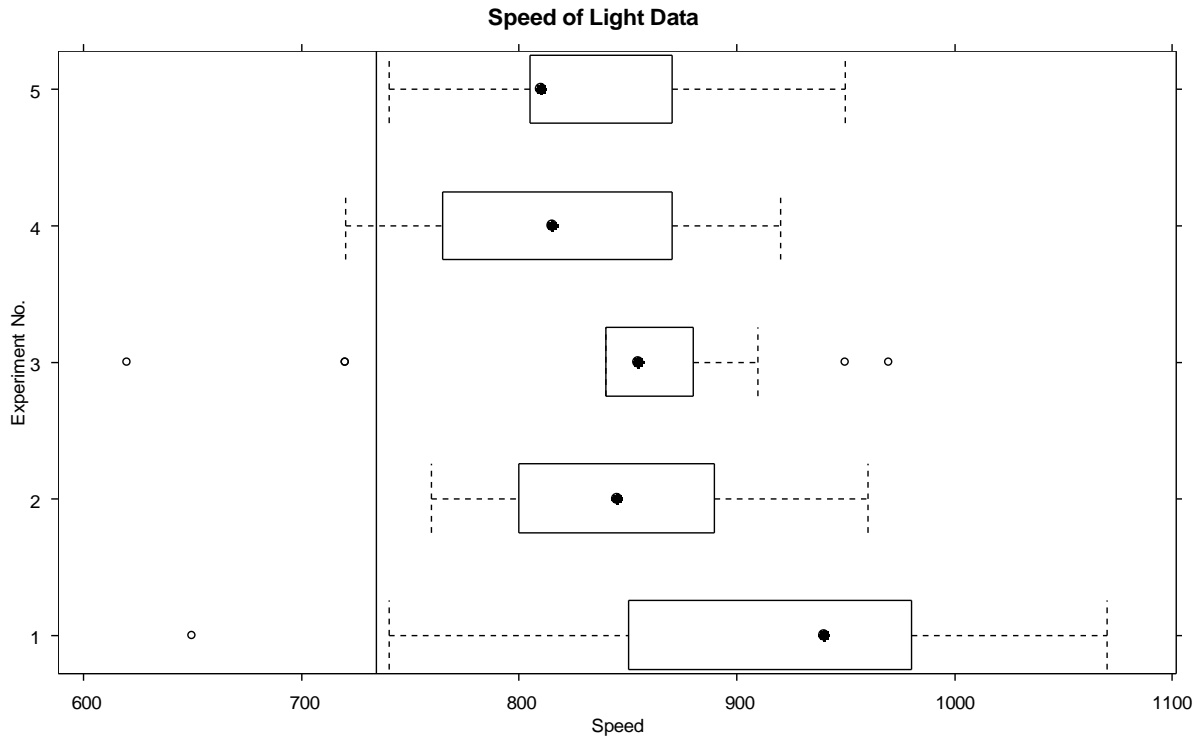
Trellis approach

```
xyplot(time ~ dist, data=hills,  
  panel = function(x, y, ...) {  
    panel.xyplot(x, y, ...)  
    panel.lmline(x, y ,type="l")  
    panel.abline(ltsreg(x, y), lty=3)  
  }  
)
```

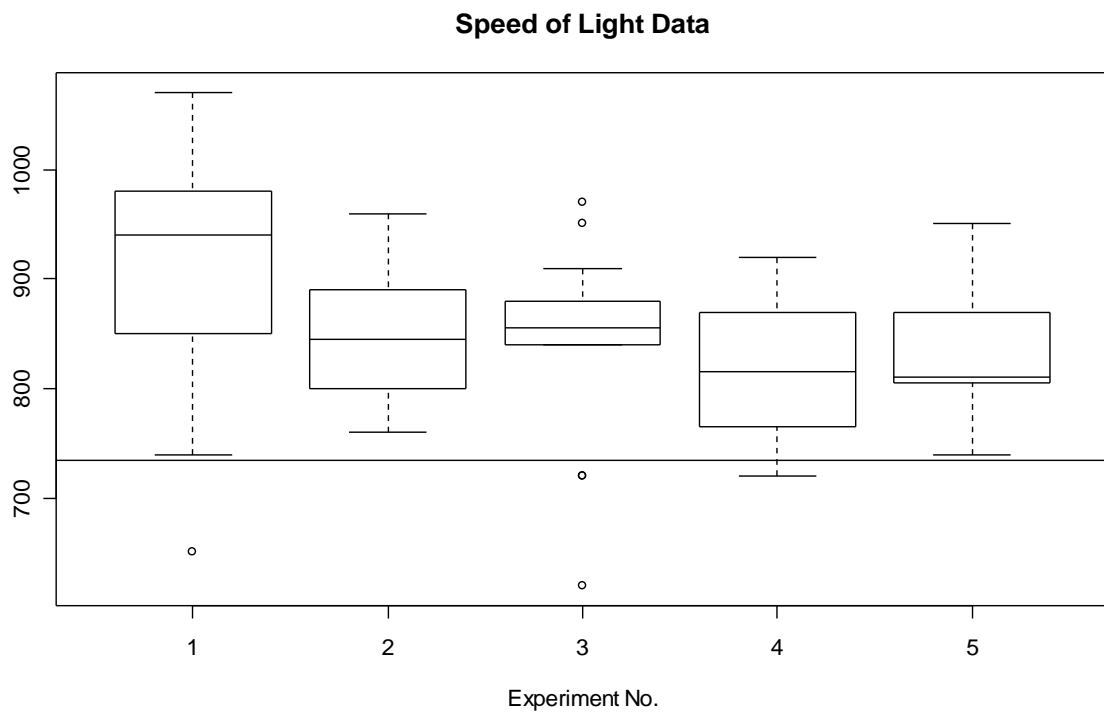
Usual plotting approach

```
attach(hills)  
plot(dist, time)  
abline(lm(time ~ dist))  
abline(ltsreg(time ~ dist), lty=3)  
detach()
```

Trellis approach



Usual approach



Trellis approach

```
bwplot(Expt ~ Speed, main="Speed of Light Data",
       ylab="Experiment No.", data=morley,
       panel = function(x, y, ...) {
         panel.bwplot(x, y, ...)
         panel.abline(v=734.5)
       }
)
```

Usual plotting approach

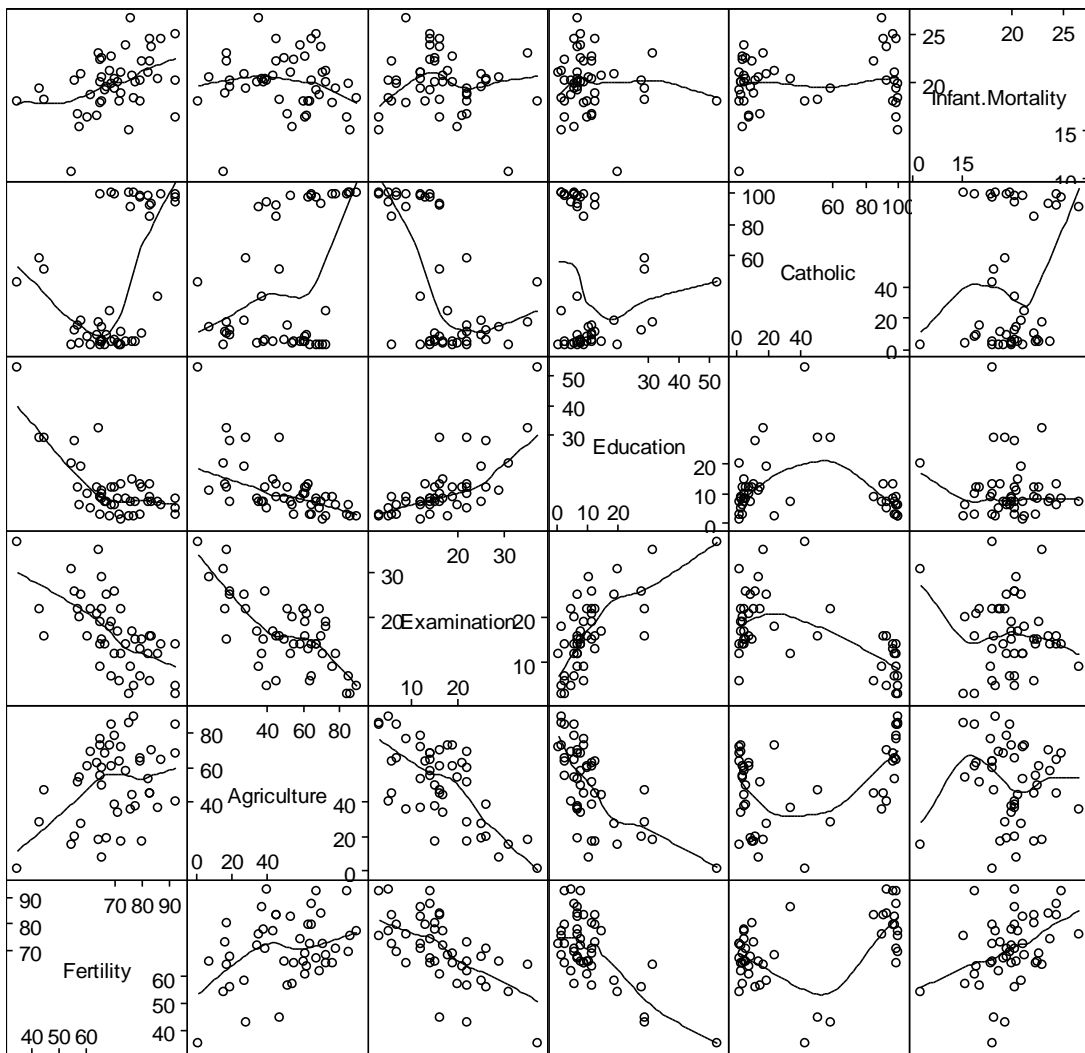
```
attach(morley)
plot.factor(Expt, Speed, main="Speed of Light Data",
            xlab="Experiment No.")
abline(h=734.5)
detach(morley)
```

Instead of plot.factor, the boxplot command

```
boxplot(split(Speed, Expt), main="Speed of Light Data",
         xlab="Experiment No.")
```

could be used as well.

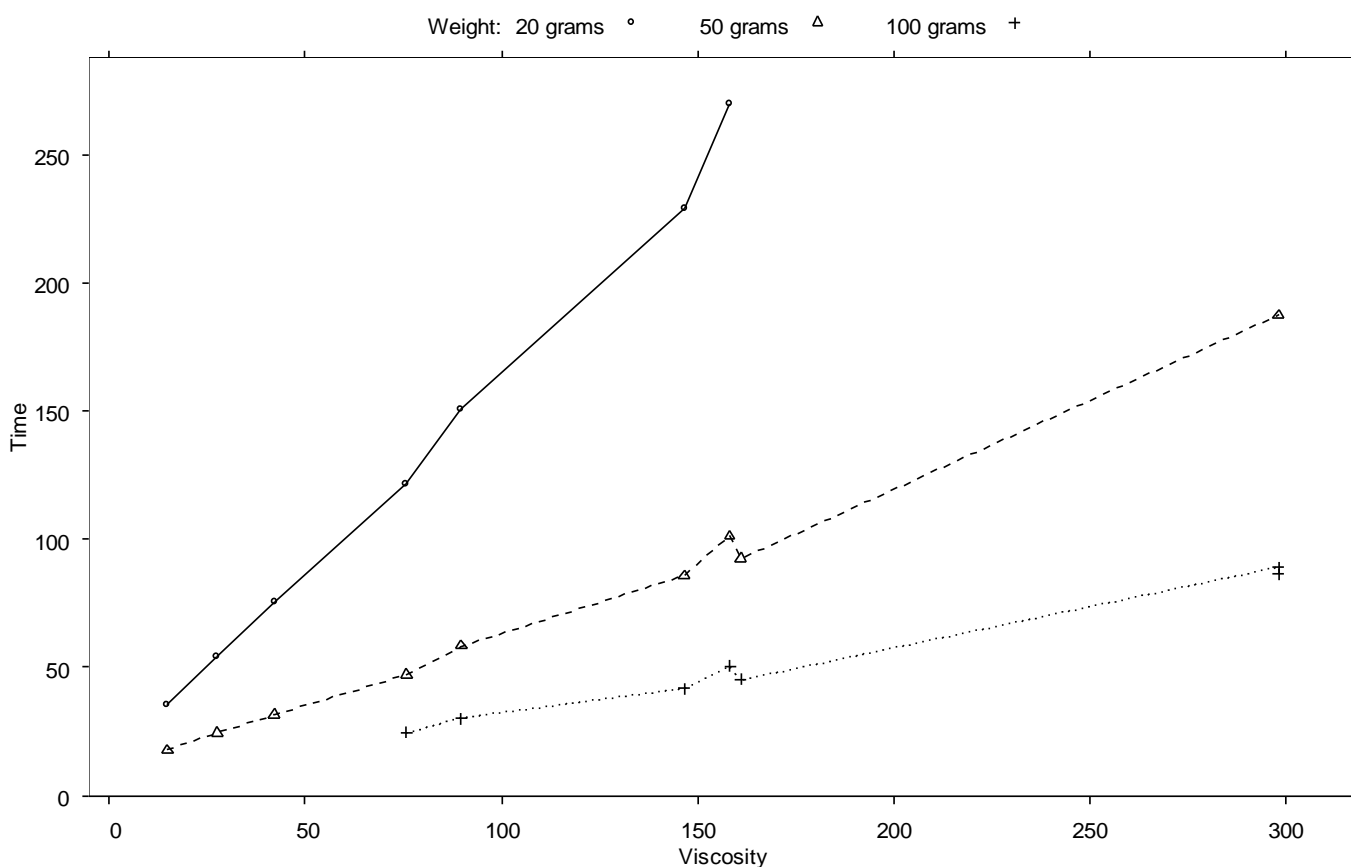
Many trellis type plots can not be done using the more standard graphics commands, or are very complicated. One example is a scatter plot matrix



Scatter Plot Matrix

```
splom(~ swiss.df, aspect="fill",
      panel = function(x, y, ...) {
        panel.xyplot(x, y, ...); panel.loess(x, y, ...)
      }
    )
```

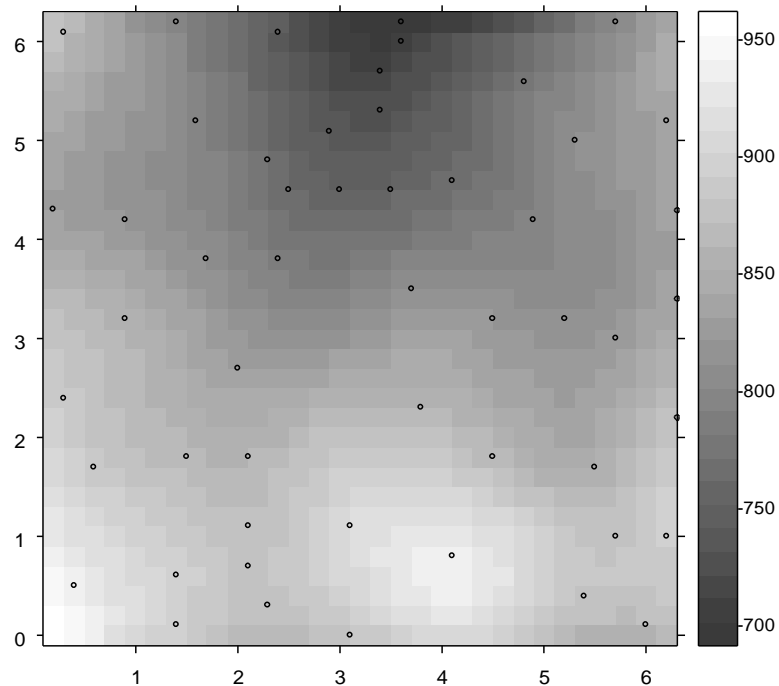
This example shows the power of panel functions, as the basic plot commands can easily be applied to multi-panel displays. In this example the scatterplot smoother loess is applied to each panel of the plot



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

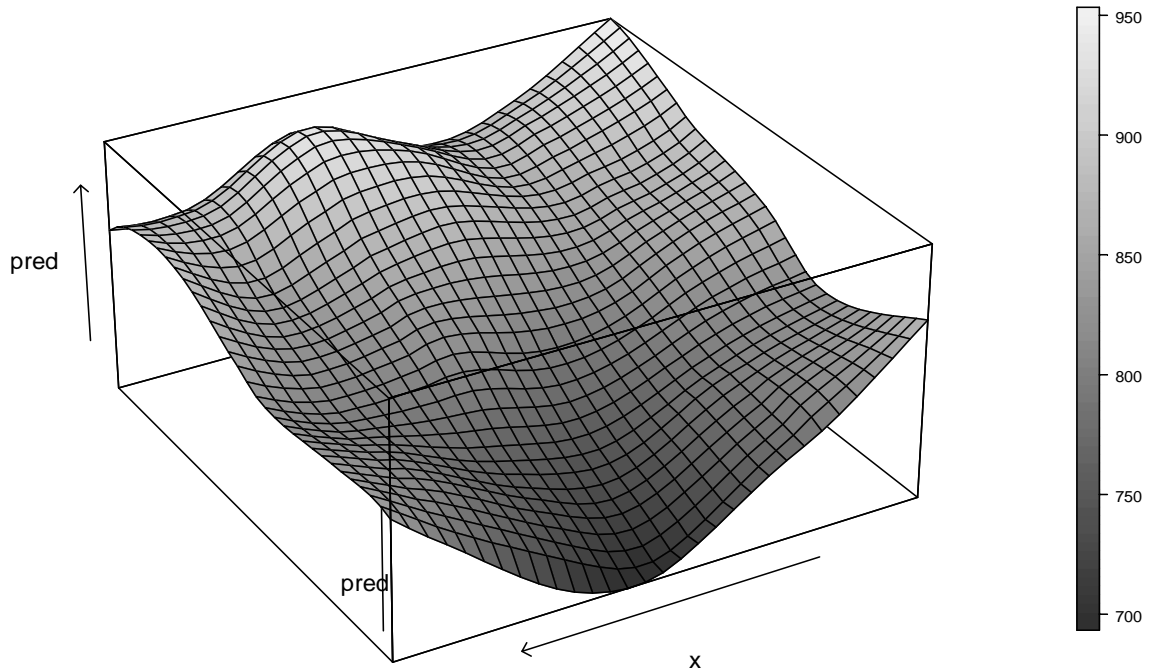
xyplot(Time ~ Viscosity, data=stormal, groups = Wt,
  panel = panel.superpose, type="b",
  key = list(columns=3,
    text= list(paste(c("Weight: ", "", "")),
      unique(stormal$Wt), "grams")),
  points=Rows(sps,1:3))
```

There are an number approaches with Trellis graphics for displaying 3 dimensional information

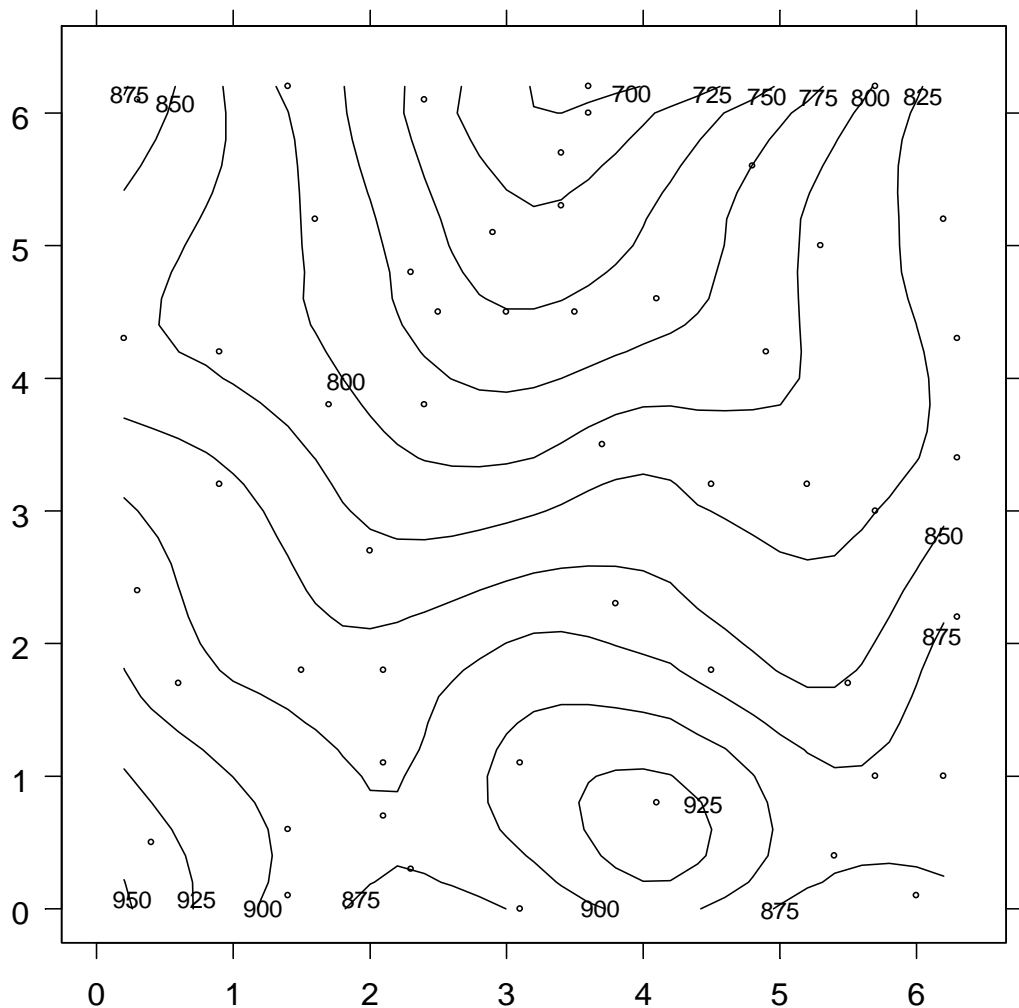


```
topo.plt <- expand.grid(topo.mar)
topo.plt$pred <- as.vector(predict(topo.loess,topo.plt))

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



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



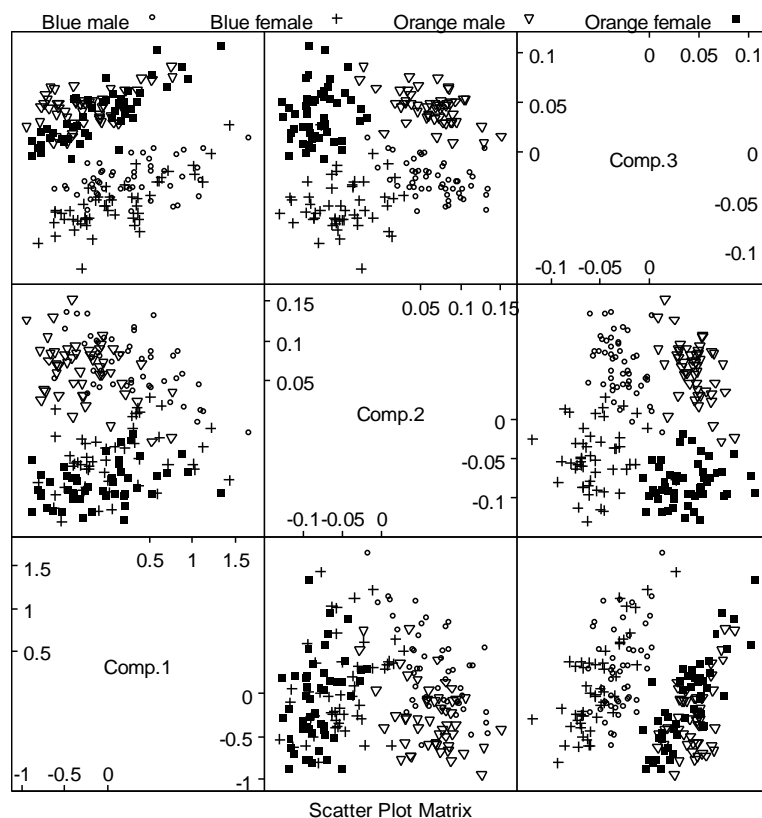
```

contourplot(z ~ x * y, mat2tr(topo.lo), aspect=1,
  at = seq(700, 1000, 25), xlab="", ylab="",
  panel = function(x,y,subscripts, ...) {
    panel.contourplot(x,y,subscripts, ...)
    panel.xyplot(topo$x, topo$y, cex=0.5) } )

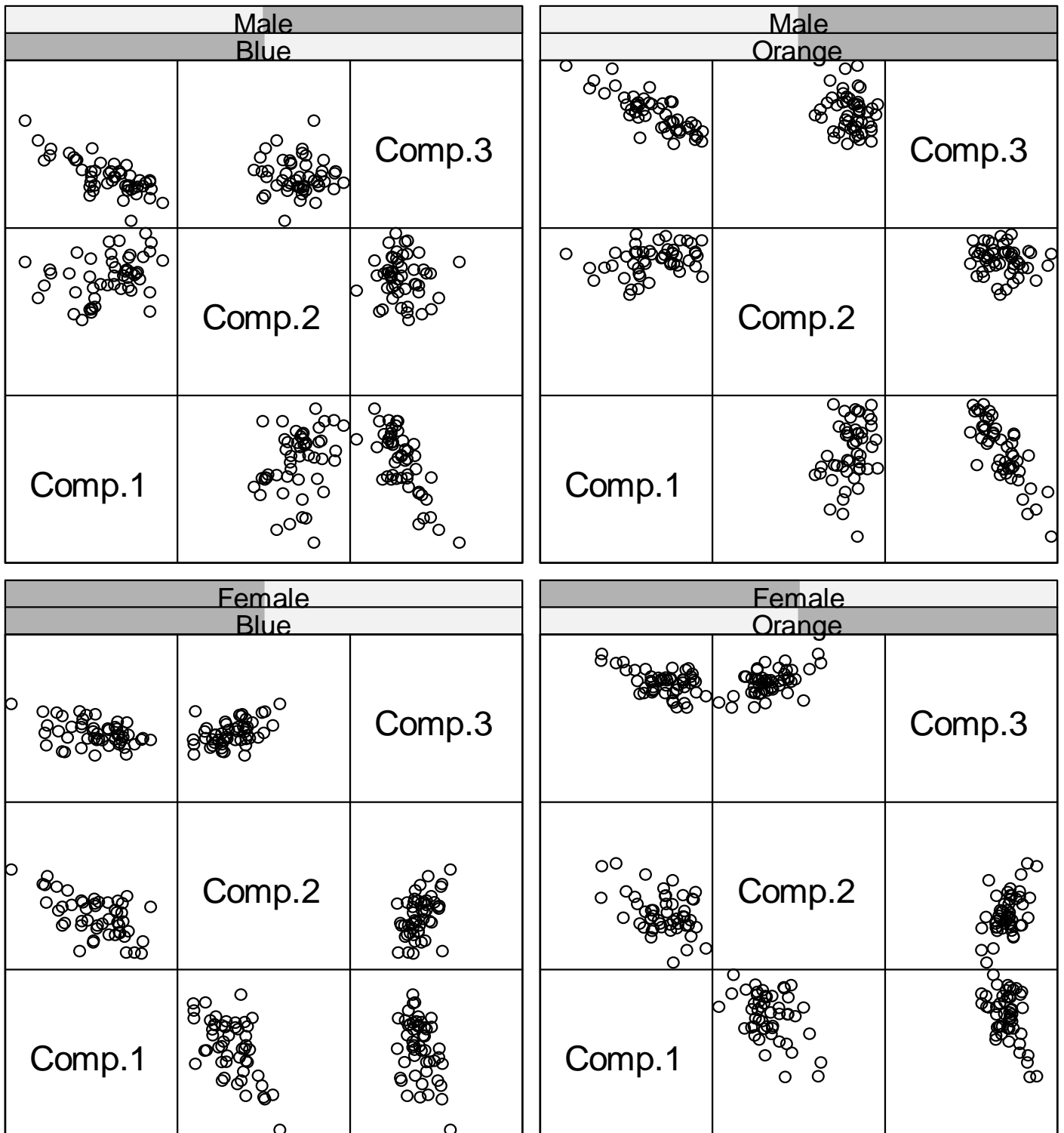
```

Note that these last two plots are from S-Plus not R (where the rest of today's plots are from. I couldn't get the code to work properly in R for these two plots.

Where Trellis plots get particularly useful is when we condition.

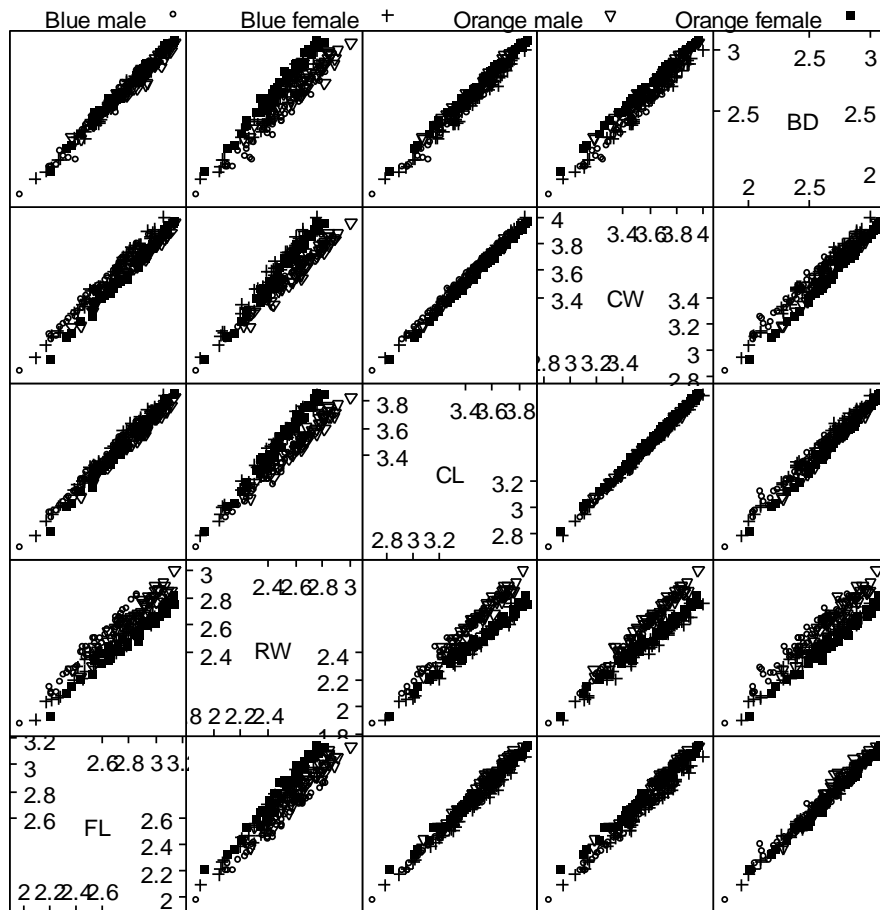


```
lcrabs <- log(crabs[,4:8])
lcrabs.pc <- predict(princomp(lcrabs))
crabs.grp <- c("B", "b", "O", "o")[rep(1:4, rep(50, 4))]
splom(~lcrabs.pc[,1:3], groups=crabs.grp,
      panel = panel.superpose,
      key = list(text=list(c("Blue male", "Blue female",
                            "Orange male", "Orange female")),
                points=Rows(trellis.par.get("superpose.symbol"), 1:4),
                columns=4))
```



```
sex <- crabs$sex; levels(sex) <- c("Female", "Male")
sp <- crabs$sp; levels(sp) <- c("Blue", "Orange")
splom(~lcrabs.pc[,1:3] | sp*sex, pscales=0)
```

The `princomp` command calculates the principle components of the data. Principle components finds linear combinations of the data, such that each is orthogonal to each other, and they maximize the variance of each. The crab data is highly correlated and the principle component analysis rotates the data to 5 uncorrelated descriptors



Scatter Plot Matrix

```
> summary(princomp(lcrabs))
```

Importance of components:

	Comp.1	Comp.2	Comp.3	Comp.4
Std deviation	0.5166405	0.07465358	0.047914392	0.024804021
Prop of Variance	0.9689051	0.02023046	0.008333672	0.002233308
Cum Proportion	0.9689051	0.98913557	0.997469244	0.999702552

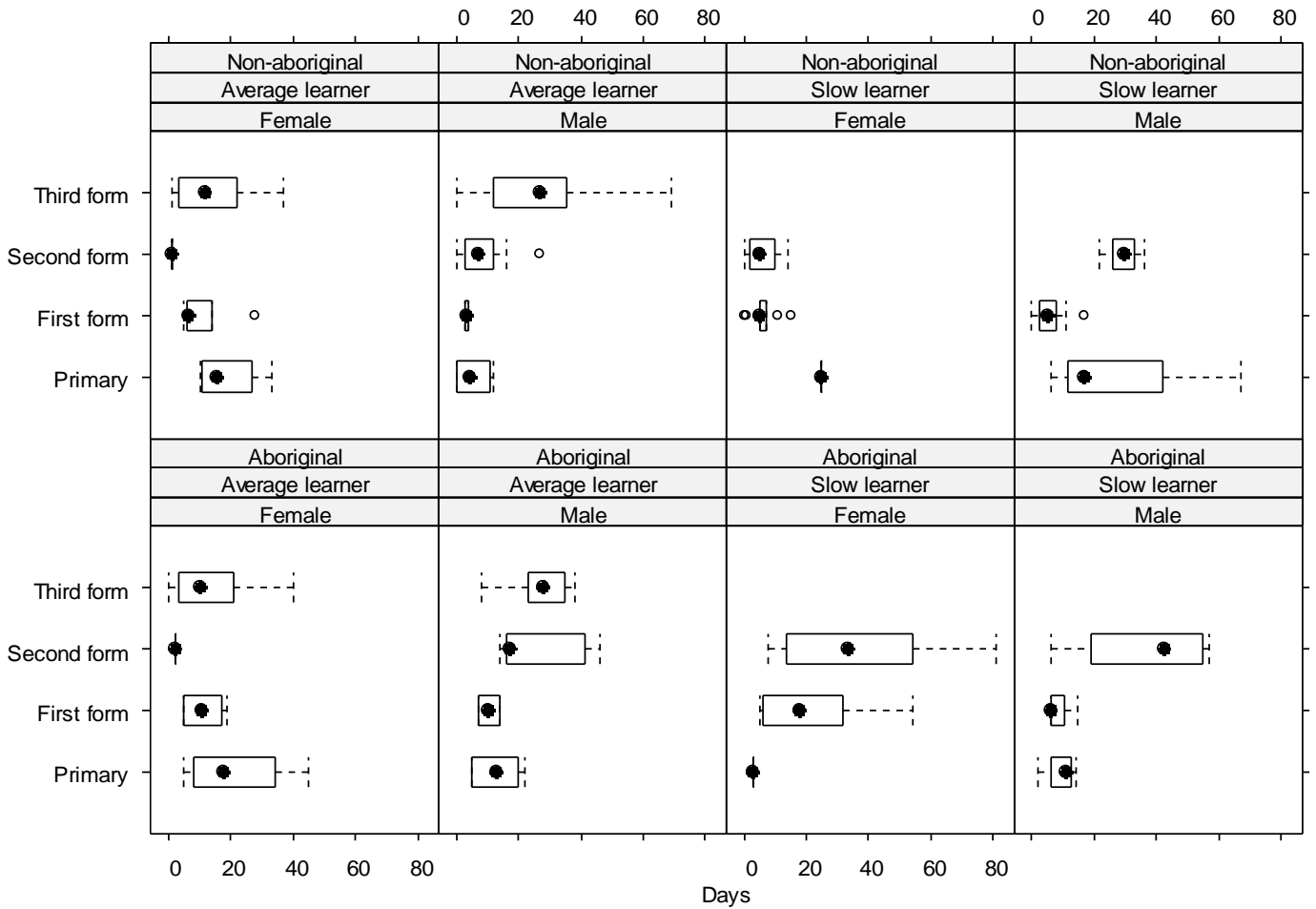
Comp.5

Standard deviation	0.0090521887
Proportion of Variance	0.0002974484
Cumulative Proportion	1.0000000000

```
> loadings(princomp(lcrabs))
```

Loadings:

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5
FL	-0.452	-0.157	0.438	0.752	0.114
RW	-0.387	0.911			
CL	-0.453	-0.204	-0.371		-0.784
CW	-0.440		-0.672		0.591
BD	-0.497	-0.315	0.458	-0.652	0.136

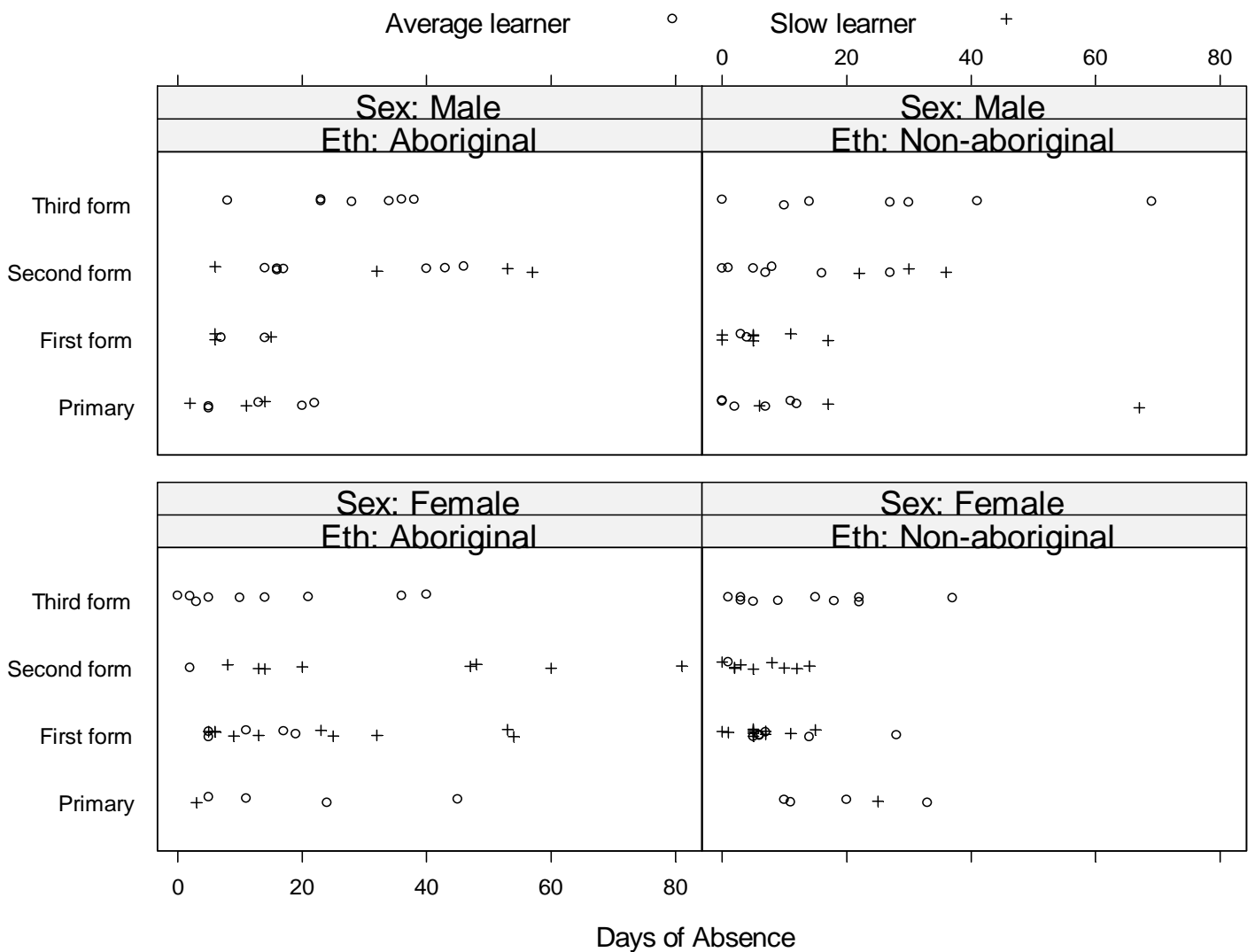


```

Quine <- quine
levels(Quine$Eth) <- c("Aboriginal", "Non-aboriginal")
levels(Quine$Sex) <- c("Female", "Male")
levels(Quine$Age) <- c("Primary", "First form", "Second form",
  "Third form")
levels(Quine$Lrn) <- c("Average learner", "Slow learner")

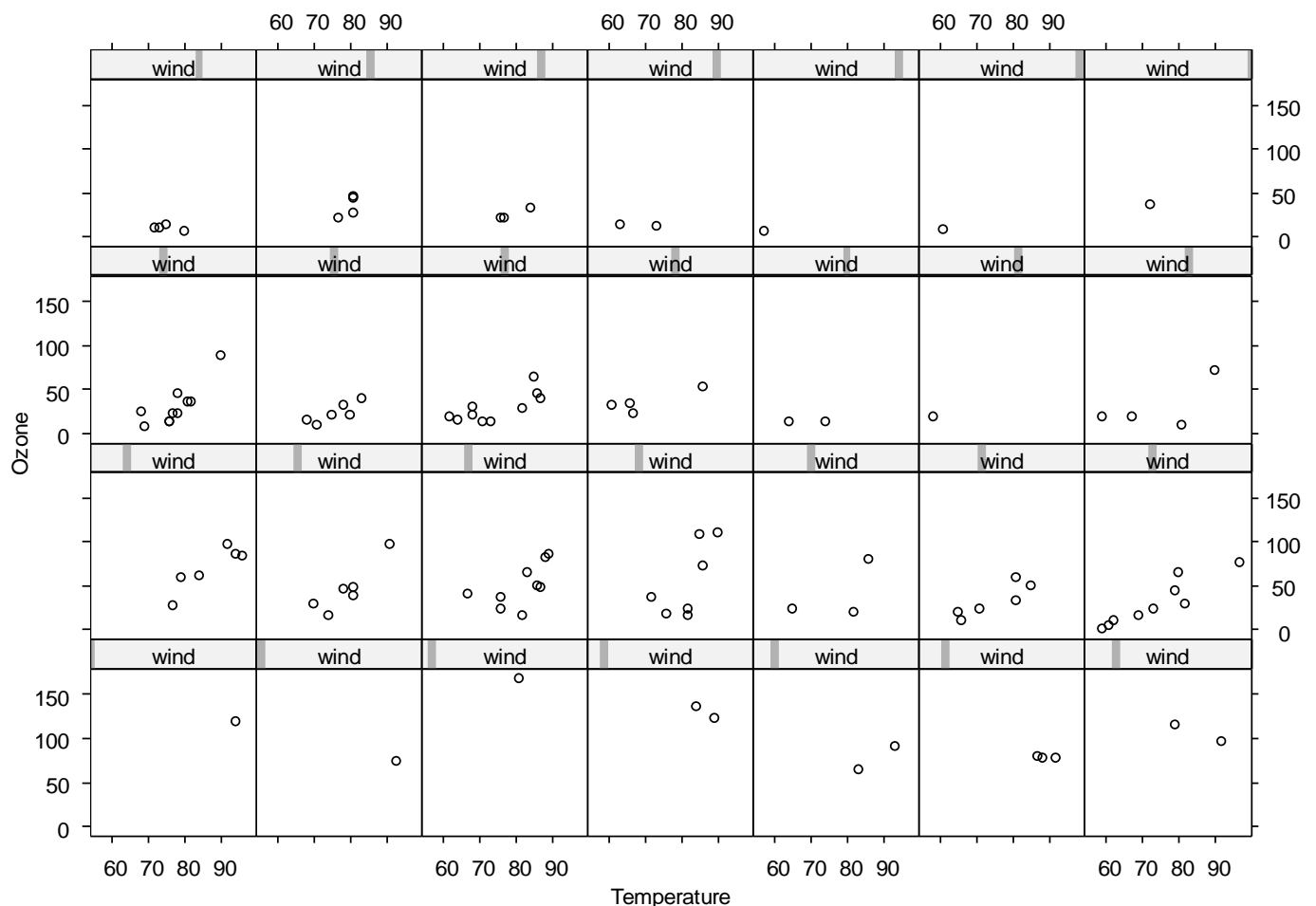
bwplot(Age ~ Days | Sex*Lrn*Eth, data=Quine, layout=c(4, 2))

```



```
stripplot(Age ~ Days | Eth*Sex, data=Quine,
  groups=Lrn, jitter=T,
  panel = function(x,y, subscripts, jitter.data=F, ...) {
    if(jitter.data) y <- jitter(y)
    panel.superpose(x,y,subscripts,...) },
  xlab = "Days of Absence",
  between = list(y=1), par.strip.text = list(cex=1.2),
  key = list(columns=2, text=list(levels(Quine$Lrn)),
    points=Rows(trellis.par.get("superpose.symbol"), 1:2)),
  strip = function(...)
    strip.default(..., strip.names=c(T,T), style=1))
```

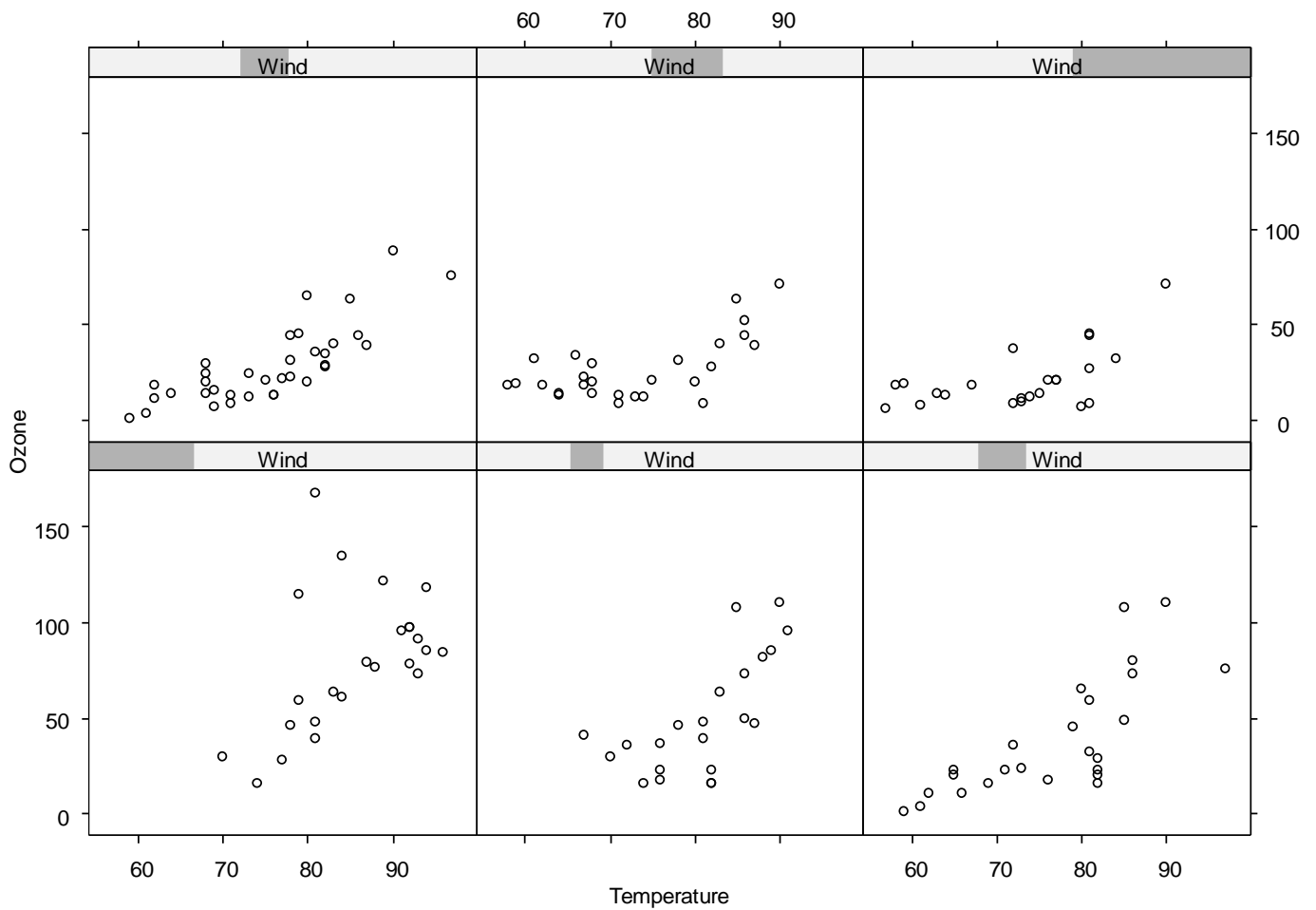
The conditioning variables in trellis plot are treated as categorical. For continuous variables, you can have lots of categories



```
xyplot(ozone ~ temperature | wind, data=environmental,
       ylab="Ozone", xlab="Temperature")
> length(unique(environmental$wind))
[1] 28
```

Lets split wind into different levels (shingles) and condition on that instead.

```
Wind <- equal.count(environmental$wind, number=6, overlap=0.2)
xyplot(ozone ~ temperature | Wind, data=environmental,
       ylab="Ozone", xlab="Temperature")
```



```
> Wind
```

```
Data:
```

```
 [1]  7.4  8.0 12.6 11.5  8.6 13.8 20.1  9.7  9.2 10.9 13.2
11.5 12.0 18.4 11.5
 [16]  9.7  9.7 16.6  9.7 12.0 12.0 14.9  5.7  7.4  9.7 13.8
11.5  8.0 14.9 20.7
```

```
and so on
```

```
 [106] 10.3 16.6  6.9 14.3  8.0 11.5
```

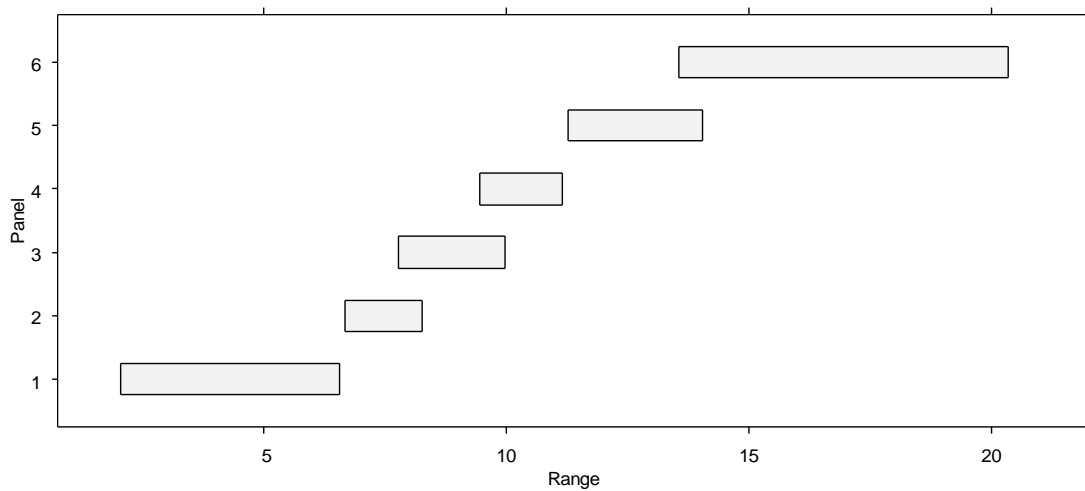
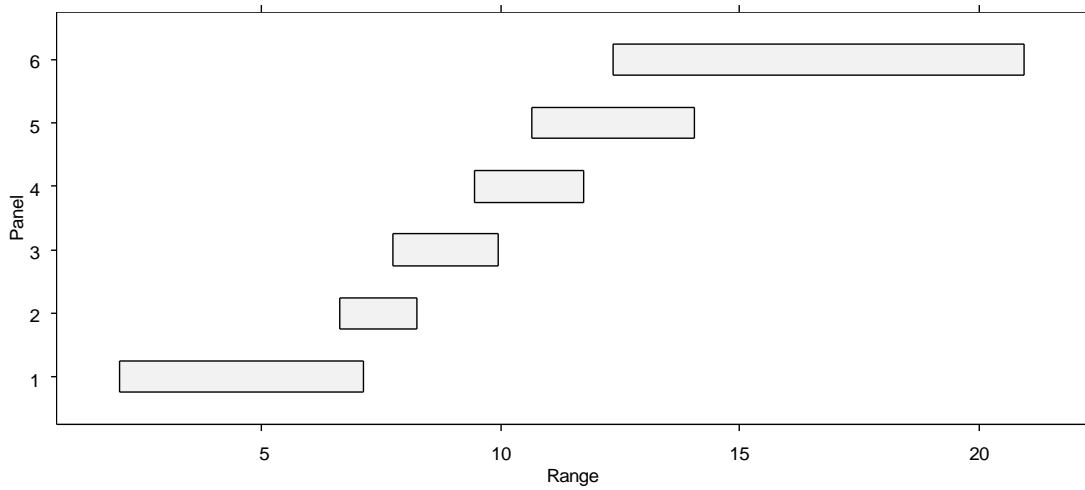
```
Intervals:
```

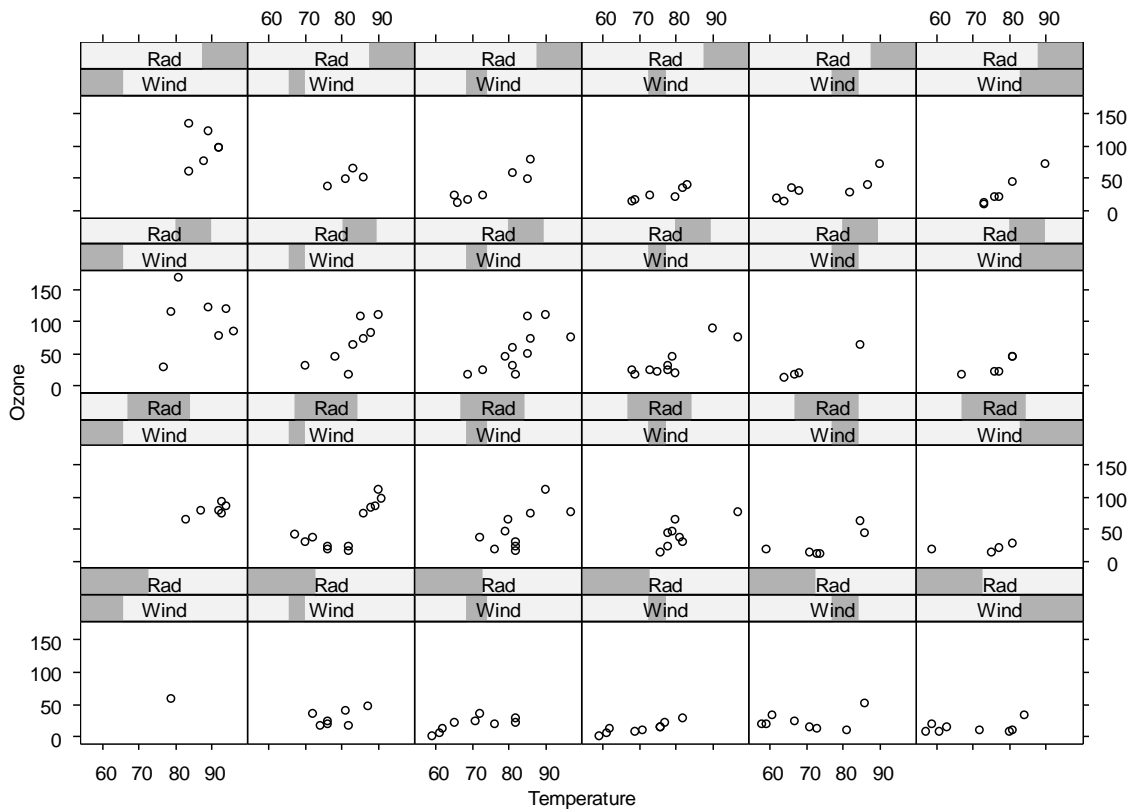
	min	max	count
1	2.05	7.15	24
2	6.65	8.25	22
3	7.75	9.95	25
4	9.45	11.75	35
5	10.65	14.05	27
6	12.35	20.95	23

```
Overlap between adjacent intervals:
```

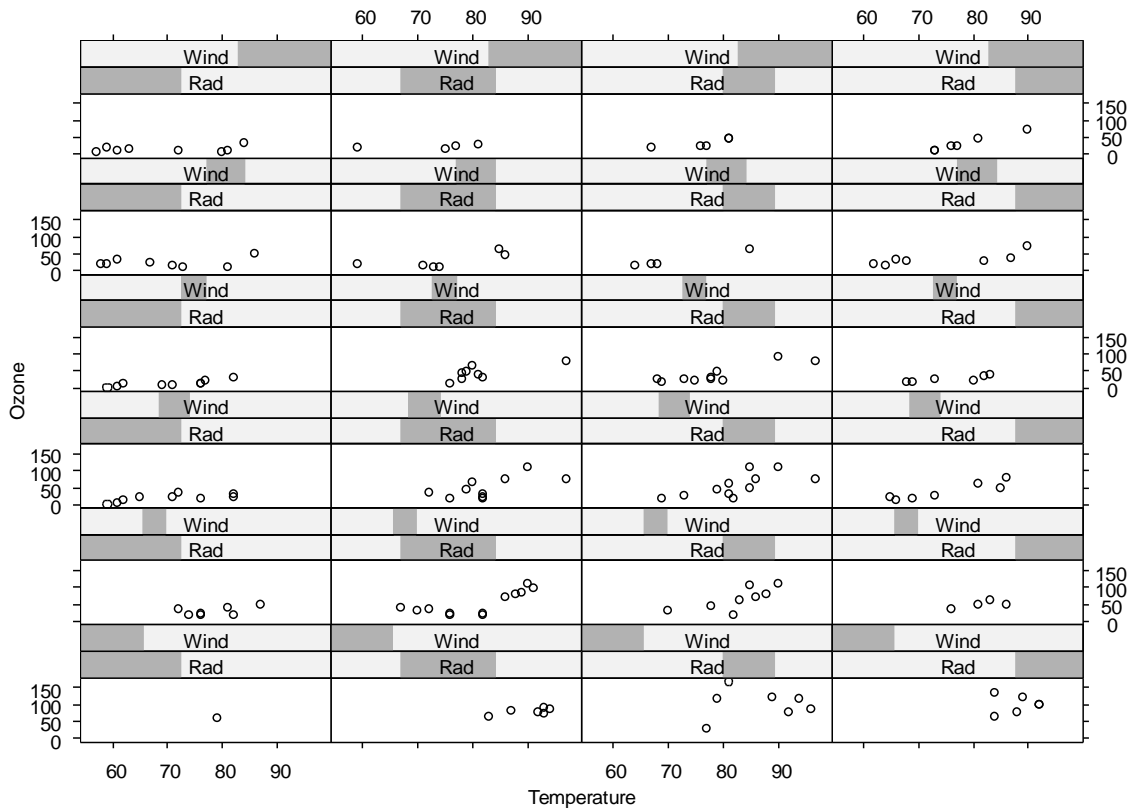
```
[1]  6  7  9 16  7
```

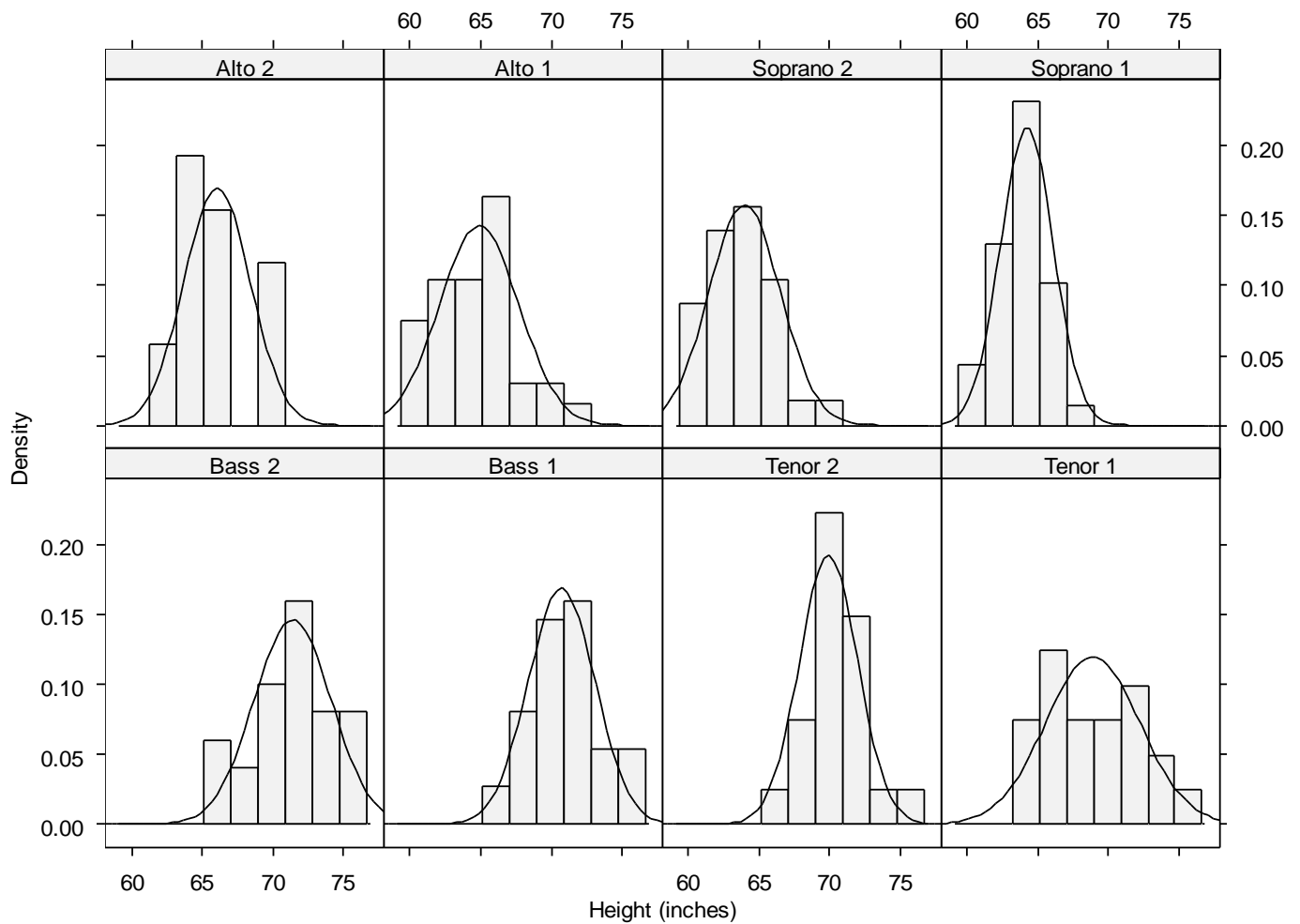
The shingles can also be displayed graphically with `plot(Wind, aspect=0.4)`





```
xyplot(ozone ~ temperature | Wind * Rad, data=environmental)
```

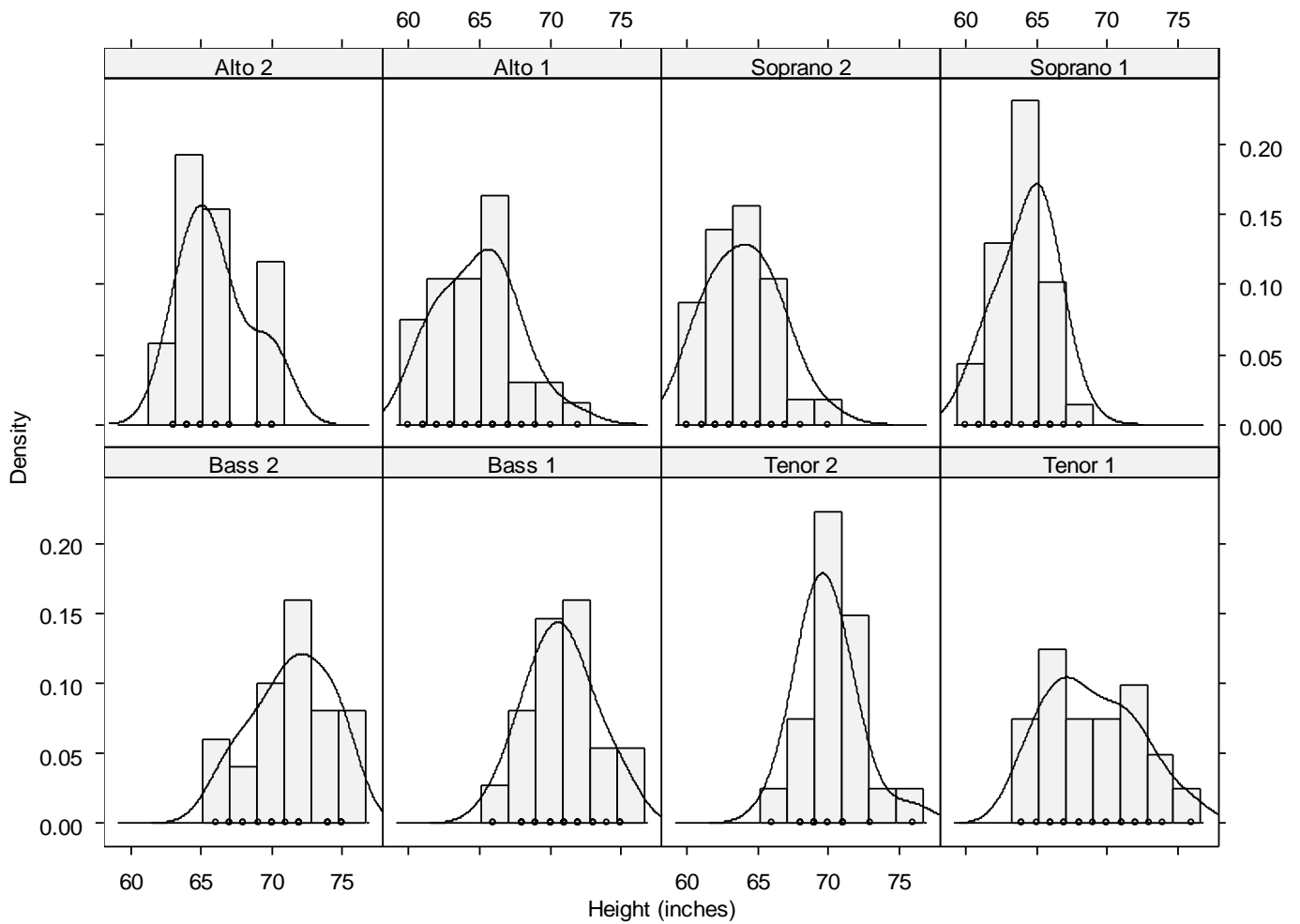




```

histogram( ~ height | voice.part, data = singer,
  xlab = "Height (inches)", type = "density",
  panel = function(x, ...) {
    panel.histogram(x, ...)
    panel.mathdensity(dmath = dnorm,
      args = list(mean=mean(x),sd=sd(x)),
      col=1)
  }
)

```



```

histogram( ~ height | voice.part, data = singer,
  xlab = "Height (inches)", type = "density",
  panel = function(x, ...) {
    panel.histogram(x, ...)
    panel.densityplot(x, darg = list(bw=1.5),
      col=1)
  }
)

```

Note that it is not possible to update Trellis graphics after they have been plotted. A Trellis plotting command actually creates a Trellis object which can be saved for example,

```
speed.tr <-bwplot(Expt ~ Speed, main="Speed of Light Data",  
  ylab="Experiment No.", data=morley)
```

This command creates the object but doesn't display it. To see what it looks then give the command

```
speed.tr
```

This will display the graph in the current Trellis device, which is probably to the screen.

A different Trellis device can be set with a command like

```
trellis.device("win.metafile", file="tdev.emf", color=F)
```

`win.metafile` (R) and `win.graph` (S-Plus version 6 or later) create Windows metafile which are great for plugging into Microsoft Word. To see what other devices are available, see `help(Devices)`. Then to display the graph in the new device, run the commands

```
speed.tr
```

```
dev.off()
```

This is similar to using `postscript` or `pdf` with regular plotting commands.