Lattice Tricks for the Power UseR

Deepayan Sarkar

Fred Hutchinson Cancer Research Center

10 August 2007

The lattice package

- Provides common statistical graphics with conditioning
- Traditional user interface:
    - collection of high level functions: xyplot(), dotplot(), etc.
    - interface based on formula and data source

Origins of lattice

- Reimplementation of the Trellis suite in S/S-PLUS
- Original goal: API compatibility with Trellis
- Trellis documentation applicable to Lattice

Subsequent Extensions

- Motivated by
  - Feature requests on the R mailing lists
  - Personal work (e.g., simplifying nlme plots)
  - Trying to enable less verbose code
- Overall, there are many non-trivial bits and pieces
- Some useful features of Trellis are not emphasized enough

Today's topics

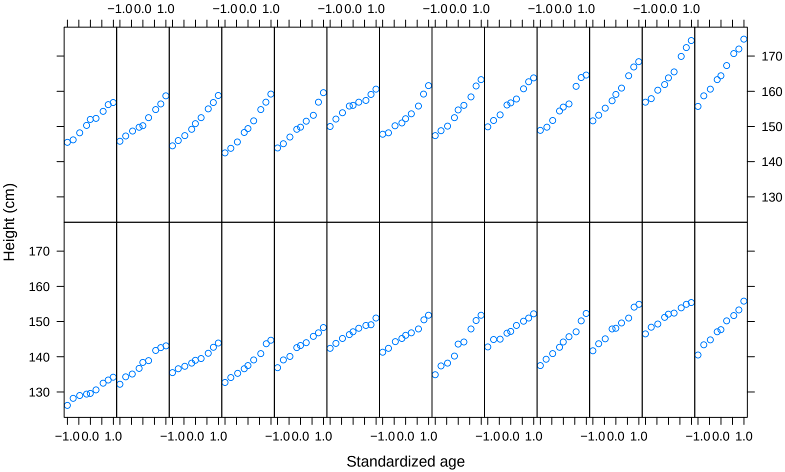
- Goal: highlight some of these features
- Case studies
  1. Adding regression lines to scatter plots
  2. Reordering levels of a factor
- Hopefully, the principles involved are easily generalizable

Example 1: Growth curves

- Heights of boys from Oxford over time
- 26 boys, height measured on 9 occasions

```
> data(Oxboys, package = "nlme")
> head(Oxboys)
  Subject     age height Occasion
1       1 -1.0000  140.5        1
2       1 -0.7479  143.4        2
3       1 -0.4630  144.8        3
4       1 -0.1643  147.1        4
5       1 -0.0027  147.7        5
6       1  0.2466  150.2        6
```

```
> xyplot(height ~ age | Subject, data = Oxboys,
         strip = FALSE, aspect = "xy",
         xlab = "Standardized age", ylab = "Height (cm)")
```



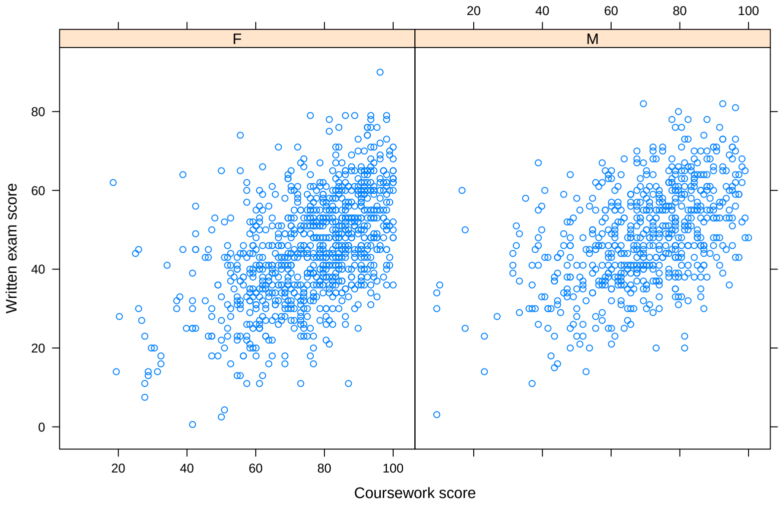
Example 2: Exam scores

- GCSE exam scores on a science subject. Two components:
  - course work
  - written paper
- 1905 students

```
> data(Gcsemv, package = "mlmRev")
> head(Gcsemv)
```

```
  school student gender written course
1  20920      16      M      23     NA
2  20920      25      F      NA   71.2
3  20920      27      F      39   76.8
4  20920      31      F      36   87.9
5  20920      42      M      16   44.4
6  20920      62      F      36     NA
```

```
> xyplot(written ~ course | gender, data = Gcsemv,
         xlab = "Coursework score",
         ylab = "Written exam score")
```



Adding to a Lattice display

- Traditional R graphics encourages incremental additions
- The Lattice analogue is to write panel functions

Digression: a simple panel function

- Things to know:
  - Panel functions are functions (!)
  - They are responsible for graphical content inside panels
  - They get executed once for every panel
  - Every high level function has a default panel function e.g., xyplot() has default panel function panel.xyplot()

Digression: a simple panel function

- So, equivalent call:

```
> xyplot(written ~ course | gender, data = Gcsemv,
         xlab = "Coursework score",
         ylab = "Written exam score",
         panel = panel.xyplot)
```

Digression: a simple panel function

- So, equivalent call:

```
> xyplot(written ~ course | gender, data = Gcsemv,
         xlab = "Coursework score",
         ylab = "Written exam score",
         panel = function(...) {
             panel.xyplot(...)
         })
```

Digression: a simple panel function

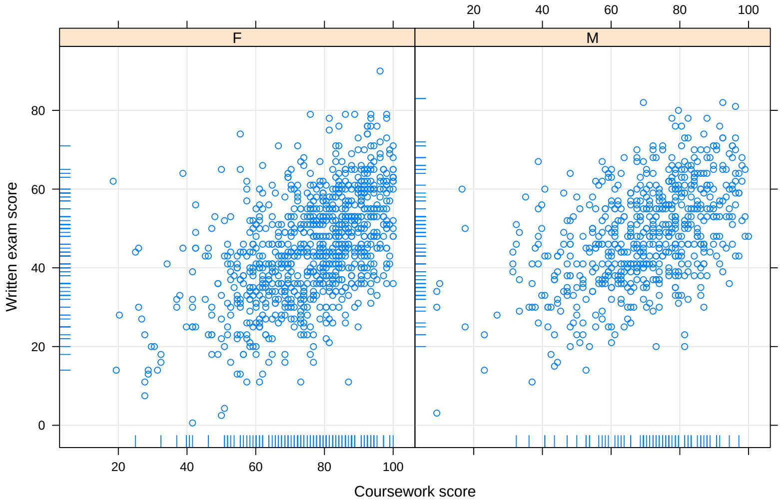
- So, equivalent call:

```
> xyplot(written ~ course | gender, data = Gcsemv,
         xlab = "Coursework score",
         ylab = "Written exam score",
         panel = function(x, y, ...) {
             panel.xyplot(x, y, ...)
         })
```

Digression: a simple panel function

- Now, we can add a couple of elements:

```
> xyplot(written ~ course | gender, data = Gcsemv,
         xlab = "Coursework score",
         ylab = "Written exam score",
         panel = function(x, y, ...) {
             panel.grid(h = -1, v = -1)
             panel.xyplot(x, y, ...)
             panel.rug(x = x[is.na(y)],
                       y = y[is.na(x)])
         })
```


F
M
Written exam score
Coursework score
0
20
40
60
80
20
40
60
80
100

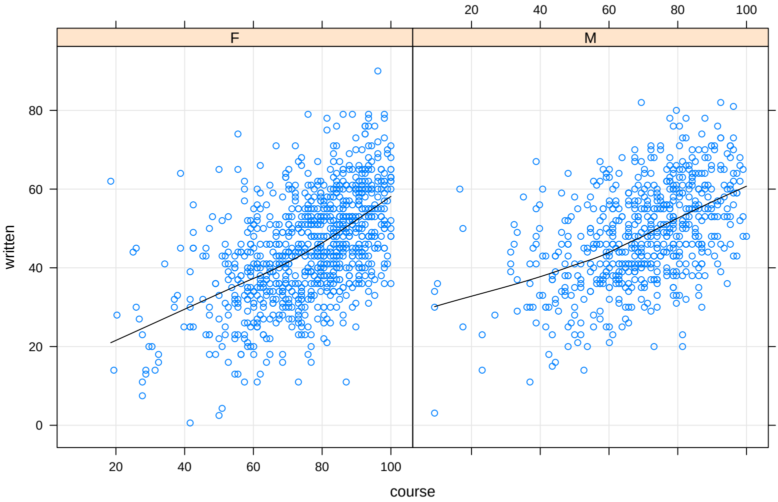
Panel functions

- Another useful feature: argument passing

```
> xyplot(written ~ course | gender, data = Gcsemv,
         panel = function(x, y, ...) {
             panel.xyplot(x, y, ...,
                          type = c("g", "p", "smooth"),
                          col.line = "black")
         })
```

is equivalent to

```
> xyplot(written ~ course | gender, data = Gcsemv,
         type = c("g", "p", "smooth"), col.line = "black")
```


F
M
written
course
0
20
40
60
80
20
40
60
80
100

Passing arguments to panel functions

- Requires knowledge of arguments supported by panel function
- For the rest of this talk, we will
  - *not* use explicit panel functions
  - instead use features of the default panel function panel.xyplot()

Back to regression lines

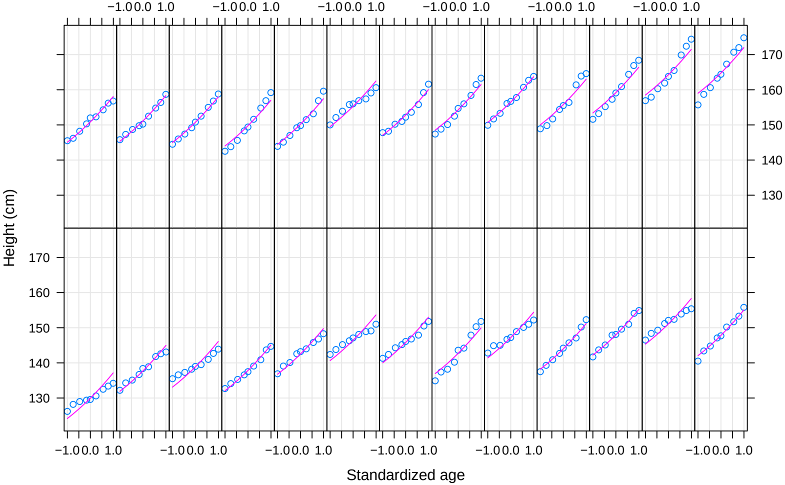
- Oxboys: model height on age

$$\mathbf{y}_{ij} = \mu + \mathbf{b}_i + \mathbf{x}_{ij} + \mathbf{x}_{ij}^2 + \varepsilon_{ij}$$

- Mixed effect model that can be fit with lme4

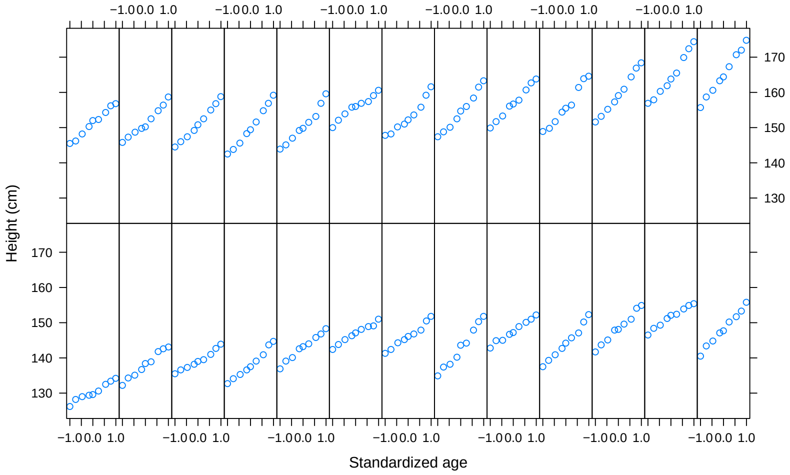
```
> library(lme4)
> fm.poly <-
      lmer(height ~ poly(age, 2) + (1 | Subject),
           data = Oxboys)
```

- Goal: plot of data with fitted curve superposed

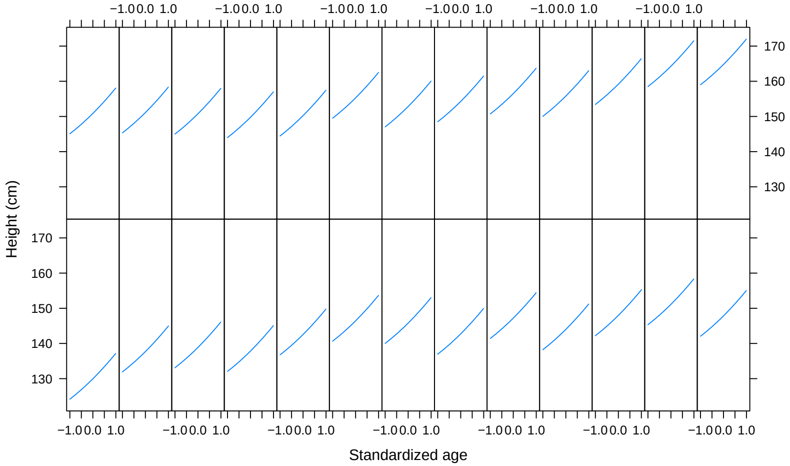


Height (cm)
Standardized age
−1.0 0.0 1.0
130
140
150
160
170

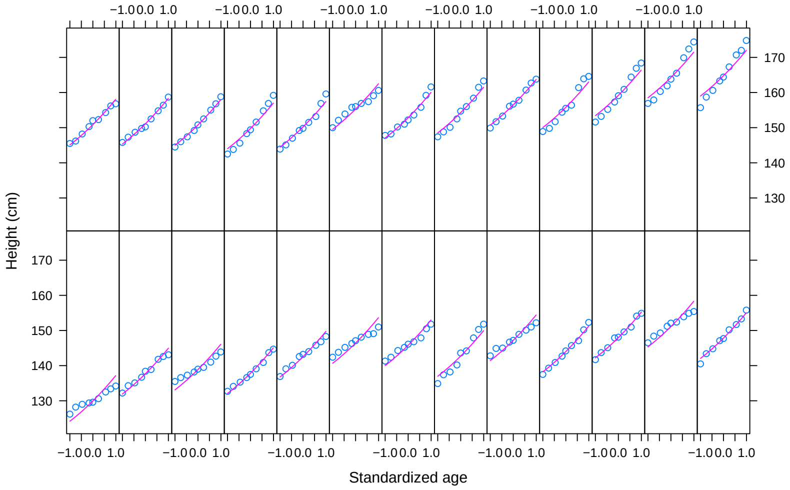
```
> xyplot(height ~ age | Subject,
         data = Oxboys, strip = FALSE, aspect = "xy",
         type = "p",
         xlab = "Standardized age", ylab = "Height (cm)")
```



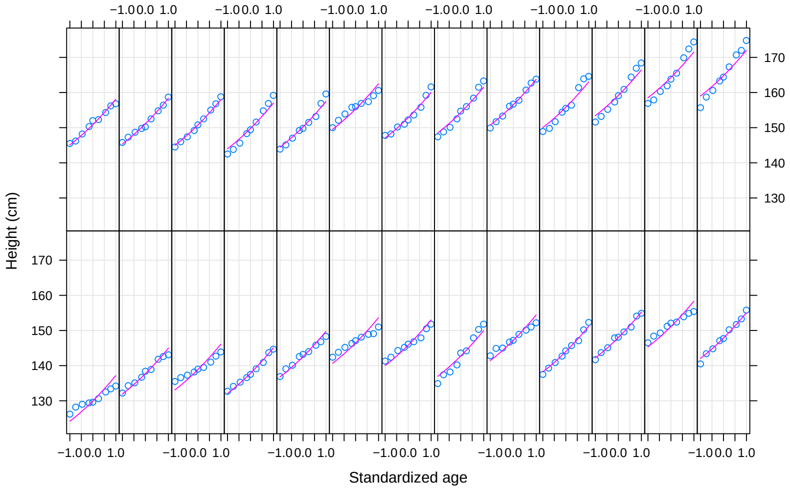
```
> xyplot(fitted(fm.poly) ~ age | Subject,
         data = Oxboys, strip = FALSE, aspect = "xy",
         type = "l",
         xlab = "Standardized age", ylab = "Height (cm)")
```



```
> xyplot(height + fitted(fm.poly) ~ age | Subject,
         data = Oxboys, strip = FALSE, aspect = "xy",
         type = c("p", "l"), distribute.type = TRUE,
         xlab = "Standardized age", ylab = "Height (cm)")
```



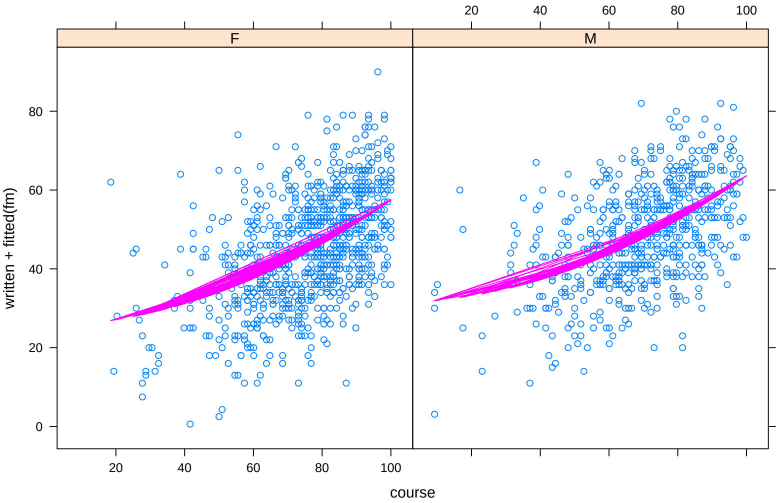
```
> xyplot(height + fitted(fm.poly) ~ age | Subject,
         data = Oxboys, strip = FALSE, aspect = "xy",
         type = list(c("p", "g"), "l"), distribute.type = TRUE,
         xlab = "Standardized age", ylab = "Height (cm)")
```



GCSE exam scores

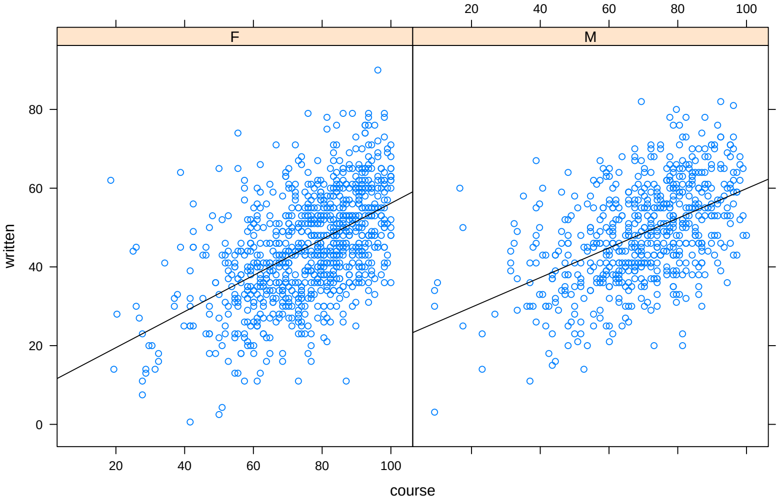
- Gcsemv: model written score by coursework and gender
- A similar approach does not work as well
  - x values are not ordered
  - missing values are omitted from fitted model

```
> fm <- lm(written ~ course + I(course^2) + gender, Gcsemv)
> xyplot(written + fitted(fm) ~ course | gender,
         data = subset(Gcsemv, !(is.na(written) | is.na(course))),
         type = c("p", "l"), distribute.type = TRUE)
```



- Built-in solution: Simple Linear Regression in each panel

```
> xyplot(written ~ course | gender, Gcsemv,
        type = c("p", "r"), col.line = "black")
```

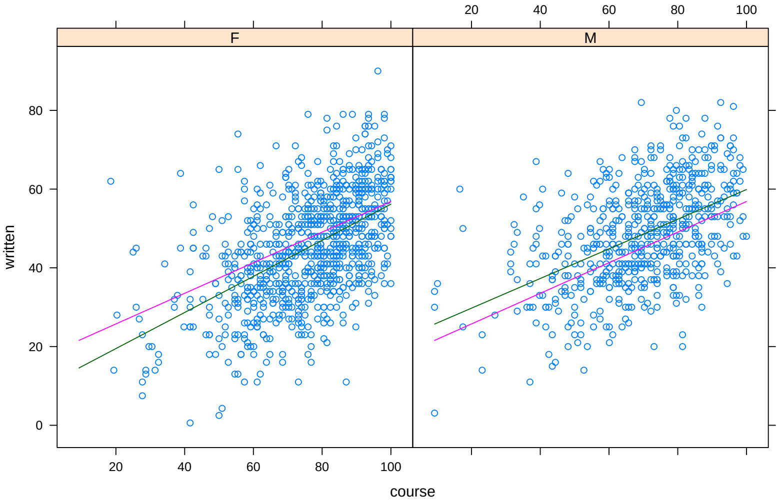


GCSE exam scores

- More complex models need a little more work
- Consider three models:

```
> fm0 <- lm(written ~ course, Gcsemv)
> fm1 <- lm(written ~ course + gender, Gcsemv)
> fm2 <- lm(written ~ course * gender, Gcsemv)
```

- Goal: compare fm2 and fm1 with fm0


F
M
written
course
0
20
40
60
80
20
40
60
80
100

One Approach

- Evaluate fits separately and combine

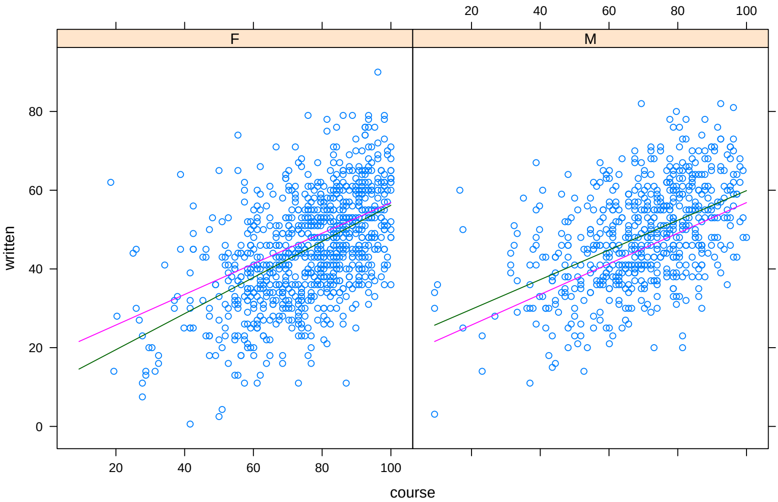
```
> course.rng <- range(Gcsemv$course, finite = TRUE)
> grid <-
      expand.grid(course = do.breaks(course.rng, 30),
                  gender = unique(Gcsemv$gender))
> fm0.pred <-
      cbind(grid,
            written = predict(fm0, newdata = grid))
> fm1.pred <-
      cbind(grid,
            written = predict(fm1, newdata = grid))
> fm2.pred <-
      cbind(grid,
            written = predict(fm2, newdata = grid))
> orig <- Gcsemv[c("course", "gender", "written")]
```

```
> str(orig)
'data.frame': 1905 obs. of  3 variables:
 $ course : num  NA 71.2 76.8 87.9 44.4 NA 89.8 17.5 32.4 84.2 .
 $ gender : Factor w/ 2 levels "F","M": 2 1 1 1 2 1 1 2 2 1 ...
 $ written: num  23 NA 39 36 16 36 49 25 NA 48 ...
> str(fm0.pred)
'data.frame': 62 obs. of  3 variables:
 $ course : num   9.25 12.28 15.30 18.32 21.35 ...
 $ gender : Factor w/ 2 levels "F","M": 2 2 2 2 2 2 2 2 2 2 ...
 $ written: num  21.6 22.7 23.9 25.1 26.3 ...
```

```
> combined <-
      make.groups(original = orig,
                  fm0 = fm0.pred,
                  fm2 = fm2.pred)
> str(combined)
```

```
'data.frame': 2029 obs. of  4 variables:
 $ course : num  NA 71.2 76.8 87.9 44.4 NA 89.8 17.5 32.4 84.2 .
 $ gender : Factor w/ 2 levels "F","M": 2 1 1 1 2 1 1 2 2 1 ...
 $ written: num  23 NA 39 36 16 36 49 25 NA 48 ...
 $ which  : Factor w/ 3 levels "original","fm0",..: 1 1 1 1 1 1
```

```
> xyplot(written ~ course | gender,
         data = combined, groups = which,
         type = c("p", "l", "l"), distribute.type = TRUE)
```



- Generalizes to
    - More than two fitted models
    - Non-linear models

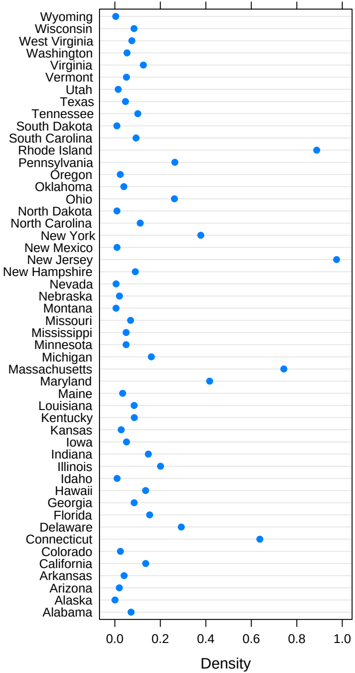
Reordering factor levels

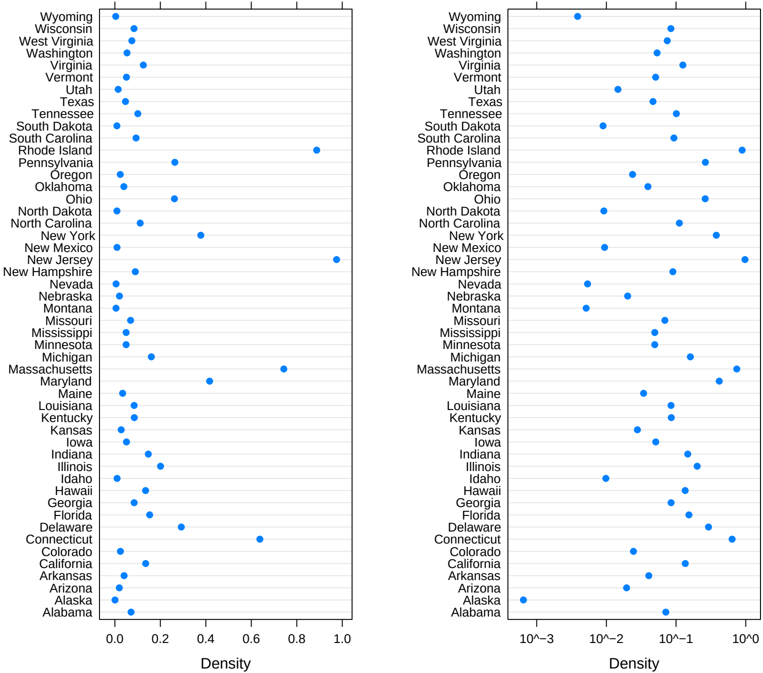
- Levels of categorical variables often have no intrinsic order
- The default in factor() is to use sort(unique(x))
  - Implies alphabetical order for factors converted from character
- Usually irrelevant in analyses
- Can strongly affect impact in a graphical display

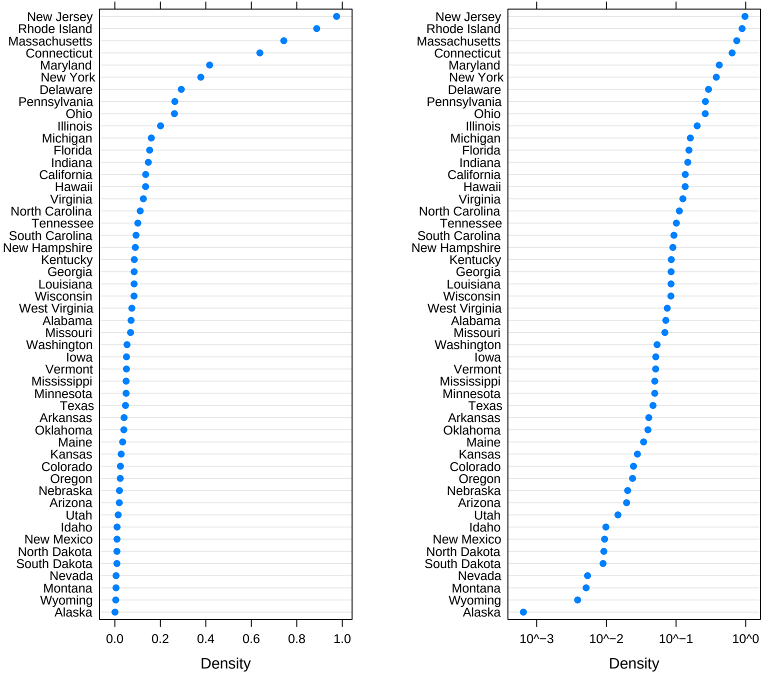
Example

- Population density in US states in 1975

```
> state <-
    data.frame(name = state.name,
               region = state.region,
               state.x77)
> state$Density <- with(state, Population / Area)
> dotplot(name ~ Density, state)
> dotplot(name ~ Density, state,
          scales = list(x = list(log = TRUE)))
```


Wyoming
Wisconsin
West Virginia
Washington
Virginia
Vermont
Utah
Texas
Tennessee
South Dakota
South Carolina
Rhode Island
Pennsylvania
Oregon
Oklahoma
Ohio
North Dakota
North Carolina
New York
New Mexico
New Jersey
New Hampshire
Nevada
Nebraska
Montana
Missouri
Mississippi
Minnesota
Michigan
Massachusetts
Maryland
Maine
Louisiana
Kentucky
Kansas
Iowa
Indiana
Illinois
Idaho
Hawaii
Georgia
Florida
Delaware
Connecticut
Colorado
California
Arkansas
Arizona
Alaska
Alabama
0.0
0.2
0.4
0.6
0.8
1.0
Density


Wyoming
Wisconsin
West Virginia
Washington
Virginia
Vermont
Utah
Texas
Tennessee
South Dakota
South Carolina
Rhode Island
Pennsylvania
Oregon
Oklahoma
Ohio
North Dakota
North Carolina
New York
New Mexico
New Jersey
New Hampshire
Nevada
Nebraska
Montana
Missouri
Mississippi
Minnesota
Michigan
Massachusetts
Maryland
Maine
Louisiana
Kentucky
Kansas
Iowa
Indiana
Illinois
Idaho
Hawaii
Georgia
Florida
Delaware
Connecticut
Colorado
California
Arkansas
Arizona
Alaska
Alabama
0.0
0.2
0.4
0.6
0.8
1.0
Density
Wyoming
Wisconsin
West Virginia
Washington
Virginia
Vermont
Utah
Texas
Tennessee
South Dakota
South Carolina
Rhode Island
Pennsylvania
Oregon
Oklahoma
Ohio
North Dakota
North Carolina
New York
New Mexico
New Jersey
New Hampshire
Nevada
Nebraska
Montana
Missouri
Mississippi
Minnesota
Michigan
Massachusetts
Maryland
Maine
Louisiana
Kentucky
Kansas
Iowa
Indiana
Illinois
Idaho
Hawaii
Georgia
Florida
Delaware
Connecticut
Colorado
California
Arkansas
Arizona
Alaska
Alabama
10^−3
10^−2
10^−1
10^0
Density


New Jersey
Rhode Island
Massachusetts
Connecticut
Maryland
New York
Delaware
Pennsylvania
Ohio
Illinois
Michigan
Florida
Indiana
California
Hawaii
Virginia
North Carolina
Tennessee
South Carolina
New Hampshire
Kentucky
Georgia
Louisiana
Wisconsin
West Virginia
Alabama
Missouri
Washington
Iowa
Vermont
Mississippi
Minnesota
Texas
Arkansas
Oklahoma
Maine
Kansas
Colorado
Oregon
Nebraska
Arizona
Utah
Idaho
New Mexico
North Dakota
South Dakota
Nevada
Montana
Wyoming
Alaska
0.0
0.2
0.4
0.6
0.8
1.0
Density
10^−3
10^−2
10^−1
10^0
Density

The reorder() function

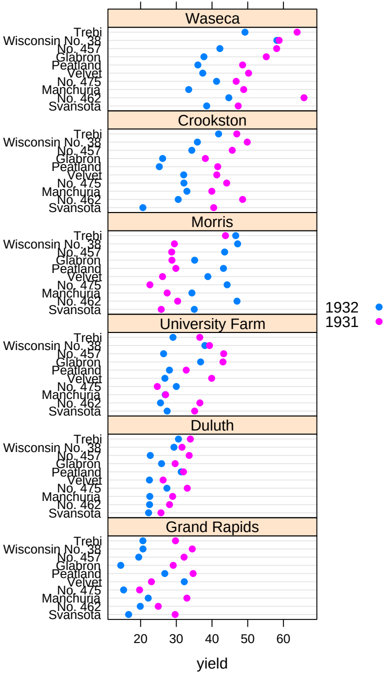
```
> dotplot(reorder(name, Density) ~ Density, state)
> dotplot(reorder(name, Density) ~ Density, state,
          scales = list(x = list(log = TRUE)))
```

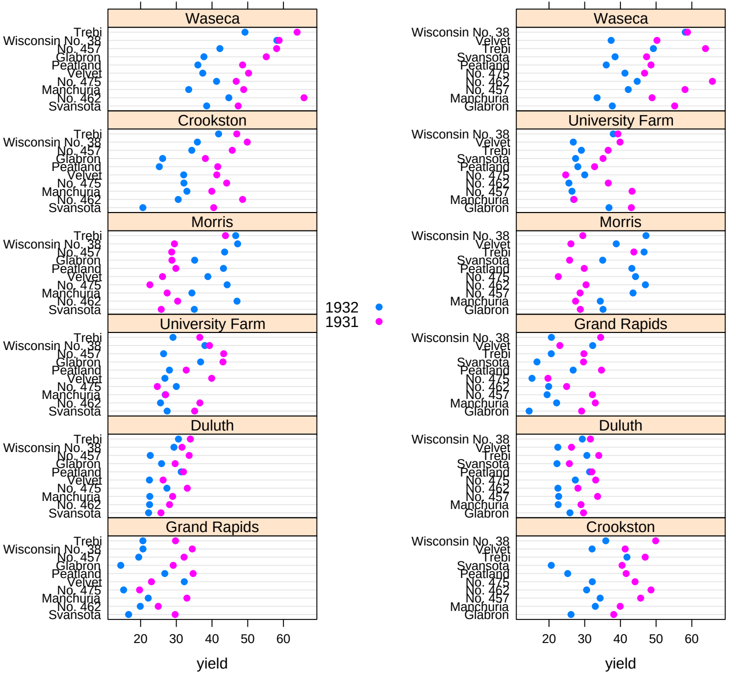
- Reorders levels of a factor by another variable
- optional summary function, default mean()

The barley example

- Response: yield of barley
- Terms: 10 varieties, 6 sites, 2 years

```
> dotplot(variety ~ yield | site, barley,
          groups = year, layout = c(1, 6))
```


Waseca
Crookston
Morris
University Farm
Duluth
Grand Rapids
Trebi
Wisconsin No. 38
No. 457
Glabron
Peatland
Velvet
No. 475
Manchuria
No. 462
Svansota
20
30
40
50
60
yield
1932
1931


Waseca
Crookston
Morris
University Farm
Duluth
Grand Rapids
Trebi
Wisconsin No. 38
No. 457
Glabron
Peatland
Velvet
No. 475
Manchuria
No. 462
Svansota
Waseca
University Farm
Morris
Grand Rapids
Duluth
Crookston
Wisconsin No. 38
Velvet
Trebi
Svansota
Peatland
No. 475
No. 462
No. 457
Manchuria
Glabron
1932
1931
20 30 40 50 60
yield
20 30 40 50 60
yield

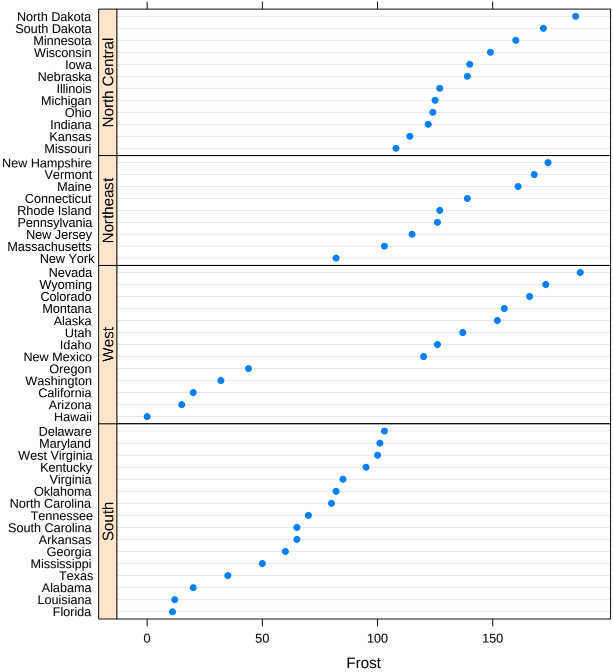
```
> dotplot(reorder(variety, yield) ~ yield | reorder(site, yield),
          data = barley, groups = reorder(year, yield), ...)
```

- The barley data has reordering already done

Reordering by multiple variables

- Not directly supported, but...
- Order is preserved within ties

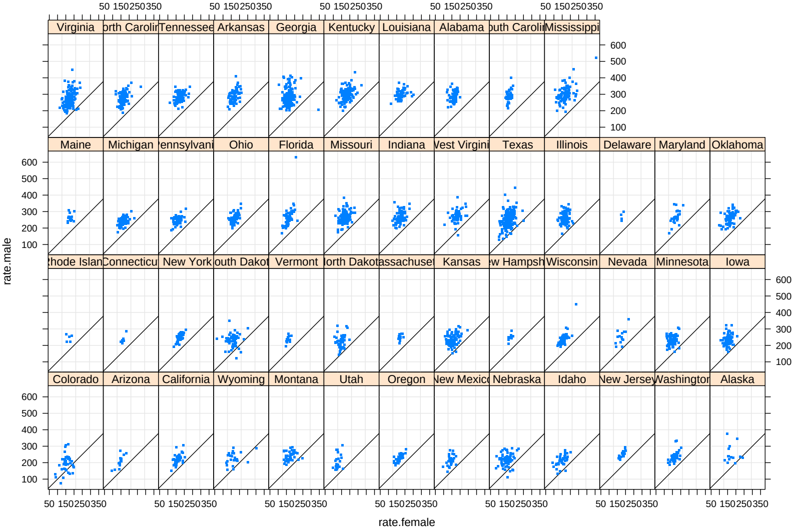
```
> state$region <- with(state, reorder(region, Frost, median))
> state$name <- with(state,
                     reorder(reorder(name, Frost),
                             as.numeric(region)))
> p <-
    dotplot(name ~ Frost | region, state,
            strip = FALSE, strip.left = TRUE, layout = c(1, 4),
            scales = list(y = list(relation = "free", rot = 0)))
> plot(p,
       panel.height = list(x = table(state$region),
                           units = "null"))
```


North Dakota
South Dakota
Minnesota
Wisconsin
Iowa
Nebraska
Illinois
Michigan
Ohio
Indiana
Kansas
Missouri
North Central
New Hampshire
Vermont
Maine
Connecticut
Rhode Island
Pennsylvania
New Jersey
Massachusetts
New York
Northeast
Nevada
Wyoming
Colorado
Montana
Alaska
Utah
Idaho
New Mexico
Oregon
Washington
California
Arizona
Hawaii
West
Delaware
Maryland
West Virginia
Kentucky
Virginia
Oklahoma
North Carolina
Tennessee
South Carolina
Arkansas
Georgia
Mississippi
Texas
Alabama
Louisiana
Florida
South
0
50
100
150
Frost

Ordering panels using index.cond

- Order panels by some summary of panel data
- Example: death rates due to cancer in US counties, 2001-2003

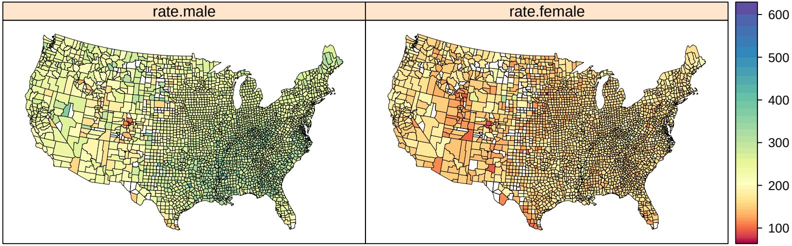
```
> data(USCancerRates, package = "latticeExtra")
> xyplot(rate.male ~ rate.female | state, USCancerRates,
         index.cond = function(x, y, ...) {
             median(y - x, na.rm = TRUE)
         },
         aspect = "iso",
         panel = function(...) {
             panel.grid(h = -1, v = -1)
             panel.abline(0, 1)
             panel.xyplot(...)
         },
         pch = ".")
```



Virginia
orth Carolin
Tennessee
Arkansas
Georgia
Kentucky
Louisiana
Alabama
outh Caroli
Mississippi
Maine
Michigan
ennsylvani
Ohio
Florida
Missouri
Indiana
Vest Virgini
Texas
Illinois
Delaware
Maryland
Oklahoma
Rhode Islan
Connecticu
New York
outh Dakot
Vermont
lorth Dakot
assachuse
Kansas
w Hampsh
Wisconsin
Nevada
Minnesota
Iowa
Colorado
Arizona
California
Wyoming
Montana
Utah
Oregon
New Mexic
Nebraska
Idaho
New Jersey
Washingtor
Alaska
rate.male
rate.female
50 150 250 350
100 200 300 400 500 600

A new Trellis function

```
> mapplot(rownames(USCancerRates) ~ rate.male + rate.female,
          data = USCancerRates,
          map = map("county", plot = FALSE,
                    fill = TRUE, projection = "tetra"),
          breaks = breaks, scales = list(draw = FALSE), xlab = "
```



A new Trellis function

- Critical piece: a new panel function

```
> panel.mapplot
function (x, y, map, breaks, colramp, lwd = 0.01, ...)
{
    names(x) <- as.character(y)
    interval <- cut(x[map$names], breaks = breaks, labels = FALS
        include.lowest = TRUE)
    col.regions <- colramp(length(breaks) - 1)
    col <- col.regions[interval]
    panel.polygon(map, col = col, lwd = lwd, ...)
}
<environment: namespace:latticeExtra>
```

Take home message

- Panel functions provide finest level of control
- Built-in panel functions are also powerful
    - Easily taken advantage of using argument passing
    - Requires knowledge of arguments (read documentation!)
    - Special function panel.superpose() useful for grouping
- Sometimes a brand new function is the best solution
- Many useful features that make life a little simpler
    - reorder(), make.groups(), etc.