Lattice Graphics: An Introduction

Deepayan Sarkar

Fred Hutchinson Cancer Research Center

13 February 2008

The lattice package

- Trellis graphics for R (originally developed in S)
- Powerful high-level data visualization system
- Provides common statistical graphics with conditioning
  - emphasis on multivariate data
  - sufficient for typical graphics needs
  - flexible enough to handle most nonstandard requirements
- Traditional user interface:
  - collection of high level functions: xyplot(), dotplot(), etc.
  - interface based on formula and data source

Outline

- Introduction, simple examples
- Overview of features
- A few case studies

Further reading

- The Lattice book (Springer's UseR! series, March 2008)
- Handouts (including Chapter 2 of the Lattice book)
- Online documentation in the lattice package

High-level functions in lattice

Function	Default Display
histogram()	Histogram
densityplot()	Kernel Density Plot
qqmath()	Theoretical Quantile Plot
qq()	Two-sample Quantile Plot
stripplot()	Stripchart (Comparative 1-D Scatter Plots)
bwplot()	Comparative Box-and-Whisker Plots
barchart()	Bar Plot
dotplot()	Cleveland Dot Plot
xyplot()	Scatter Plot
splom()	Scatter-Plot Matrix
contourplot()	Contour Plot of Surfaces
levelplot()	False Color Level Plot of Surfaces
wireframe()	Three-dimensional Perspective Plot of Surfaces
cloud()	Three-dimensional Scatter Plot
parallel()	Parallel Coordinates Plot

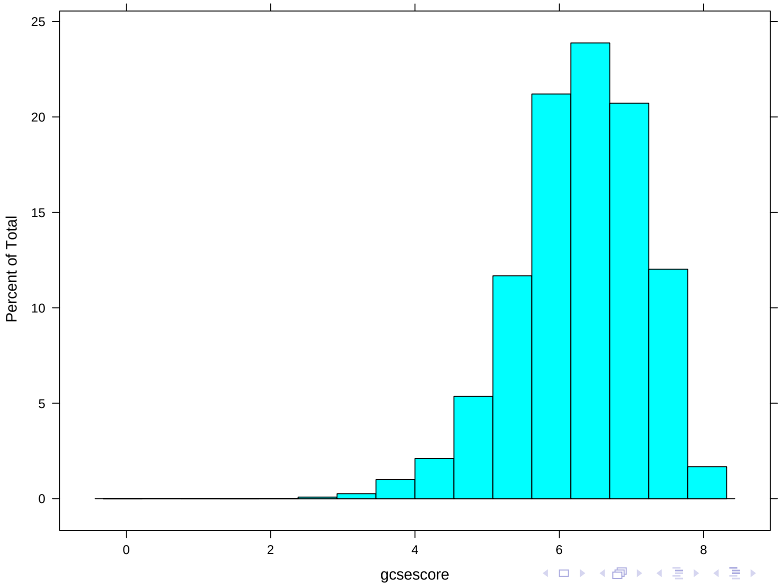
The Chem97 dataset

- 1997 A-level Chemistry examination in Britain

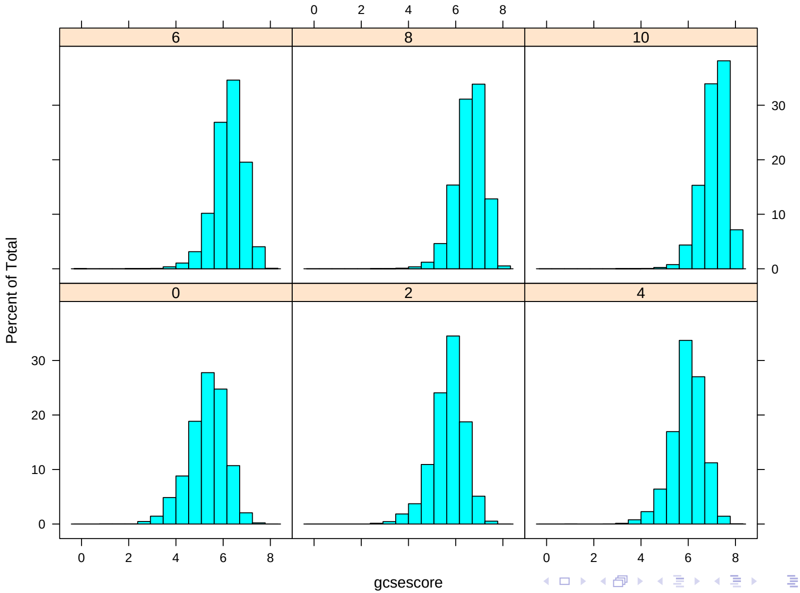
```
> data(Chem97, package = "mlmRev")
> head(Chem97[c("score", "gender", "gcsescore")])
```

```
  score gender gcsescore
1     4      F     6.625
2    10      F     7.625
3    10      F     7.250
4    10      F     7.500
5     8      F     6.444
6    10      F     7.750
```

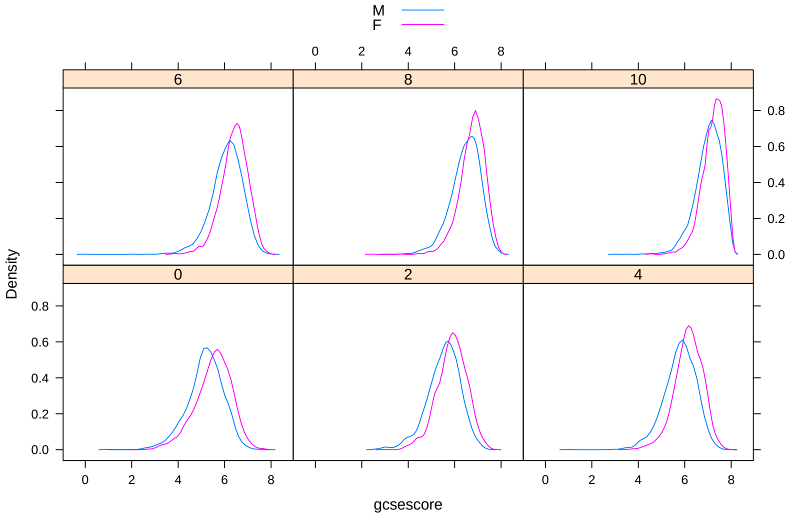
```
> histogram(~ gcsescore, data = Chem97)
```



```
> histogram(~ gcsescore | factor(score), data = Chem97)
```



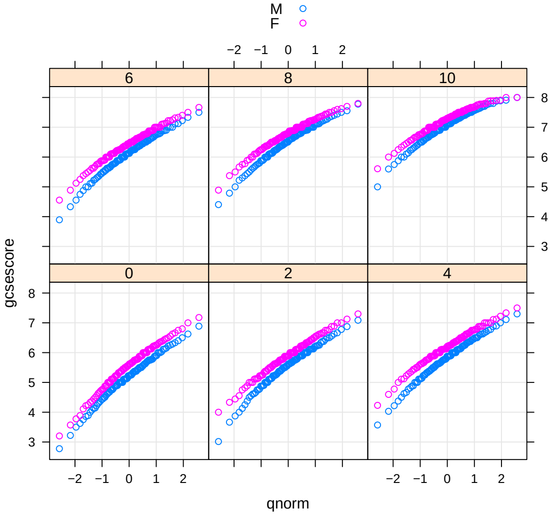
```
> densityplot(~ gcsescore | factor(score), Chem97,
              plot.points = FALSE,
              groups = gender, auto.key = TRUE)
```



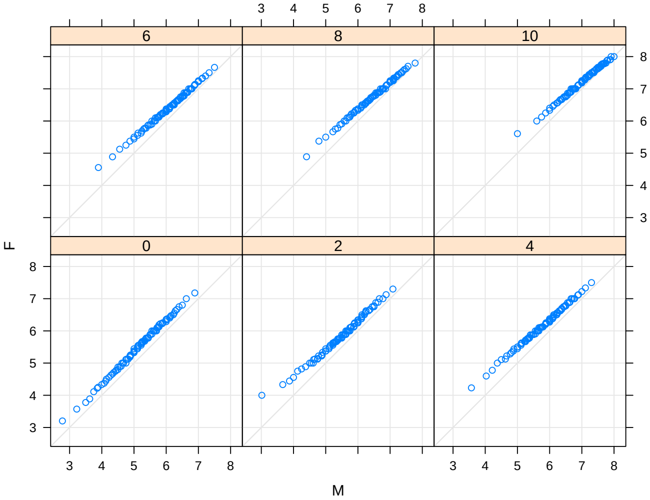
Trellis Philosophy: Part I

- Display specified in terms of
  - Type of display (histogram, densityplot, etc.)
  - Variables with specific roles
- Typical roles for variables
  - Primary variables: used for the main graphical display
  - Conditioning variables: used to divide into subgroups and juxtapose (multipanel conditioning)
  - Grouping variable: divide into subgroups and superpose
- Primary interface: high-level functions
  - Each function corresponds to a display type
  - Specification of roles depends on display type
    - Usually specified through the formula and the groups argument

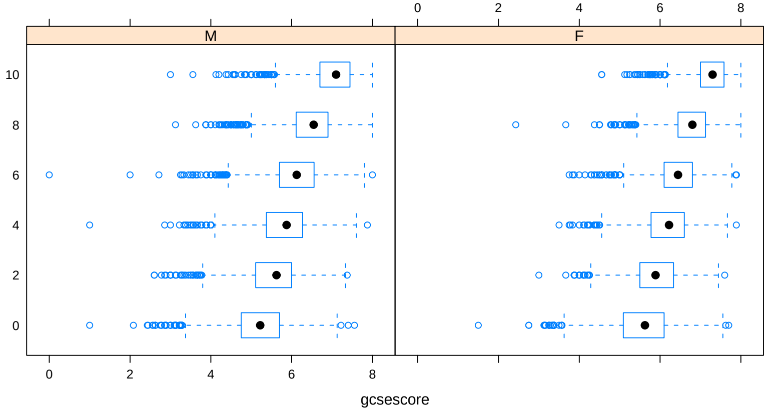
```
> qqmath(~ gcsescore | factor(score), Chem97, groups = gender,
         f.value = ppoints(100), auto.key = TRUE,
         type = c("p", "g"), aspect = "xy")
```



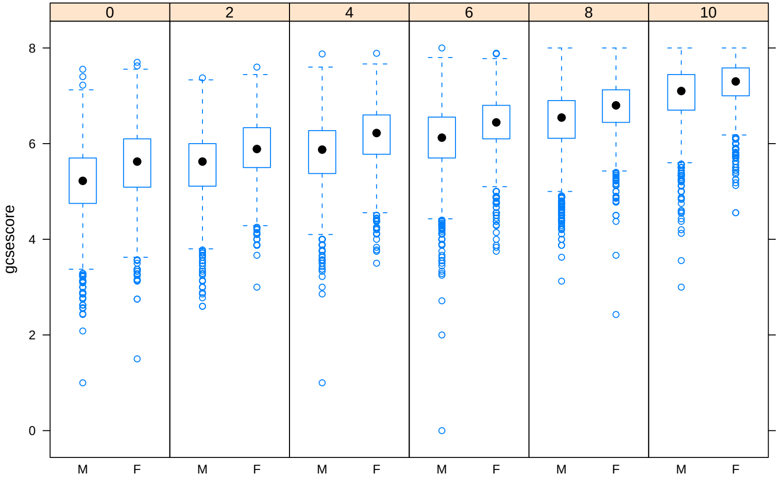
```
> qq(gender ~ gcsescore | factor(score), Chem97,
     f.value = ppoints(100), type = c("p", "g"), aspect = 1)
```



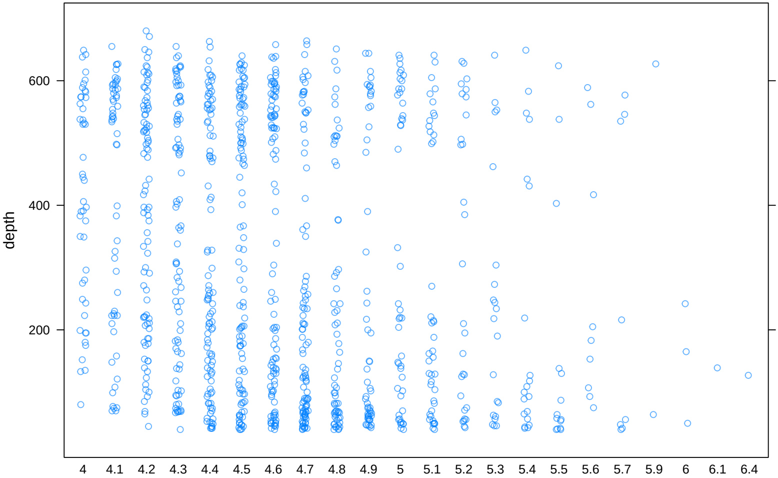
```
> bwplot(factor(score) ~ gcsescore | gender, Chem97)
```



```
> bwplot(gcsescore ~ gender | factor(score), Chem97,
         layout = c(6, 1))
```



```
> stripplot(depth ~ factor(mag), data = quakes,
            jitter.data = TRUE, alpha = 0.6)
```

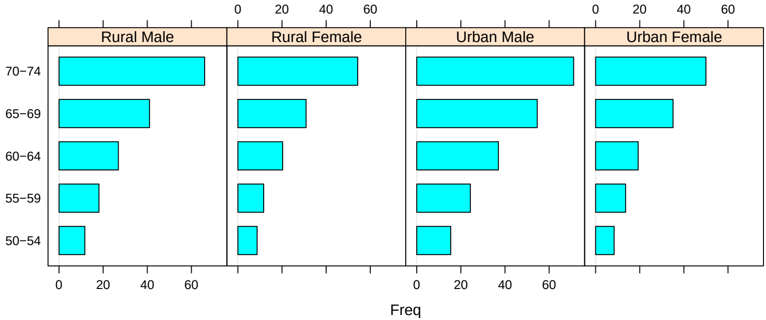


The VADeaths dataset

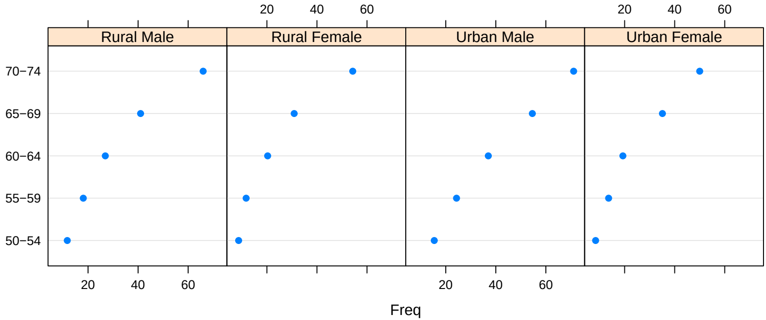
- Death rates in Virginia, 1941, among different population subgroups

```
> VADeaths

      Rural Male Rural Female Urban Male Urban Female
50-54       11.7          8.7       15.4          8.4
55-59       18.1         11.7       24.3         13.6
60-64       26.9         20.3       37.0         19.3
65-69       41.0         30.9       54.6         35.1
70-74       66.0         54.3       71.1         50.0
```

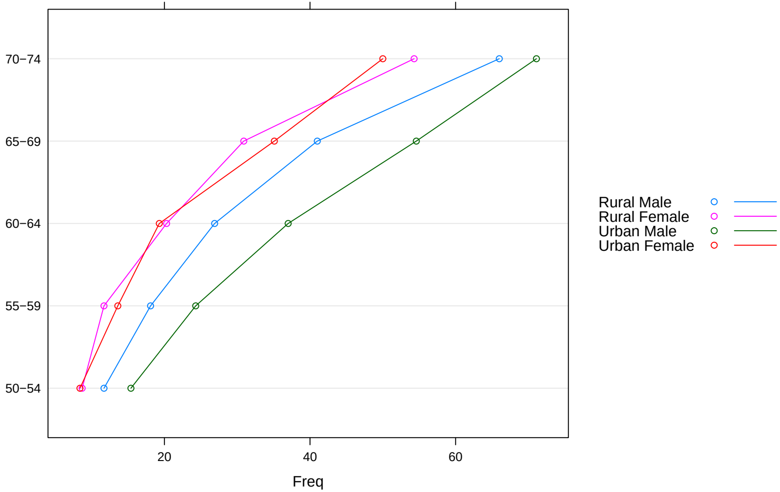
```
> barchart(VADeaths, groups = FALSE, layout = c(4, 1))
```



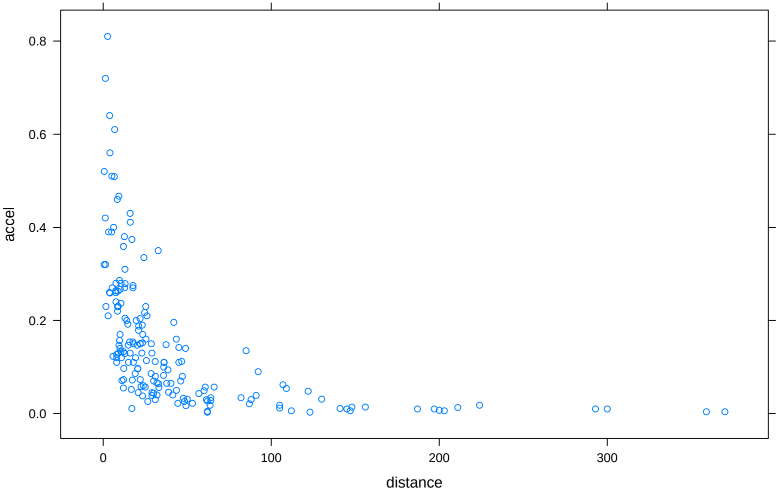
```
> dotplot(VADeaths, groups = FALSE, layout = c(4, 1))
```



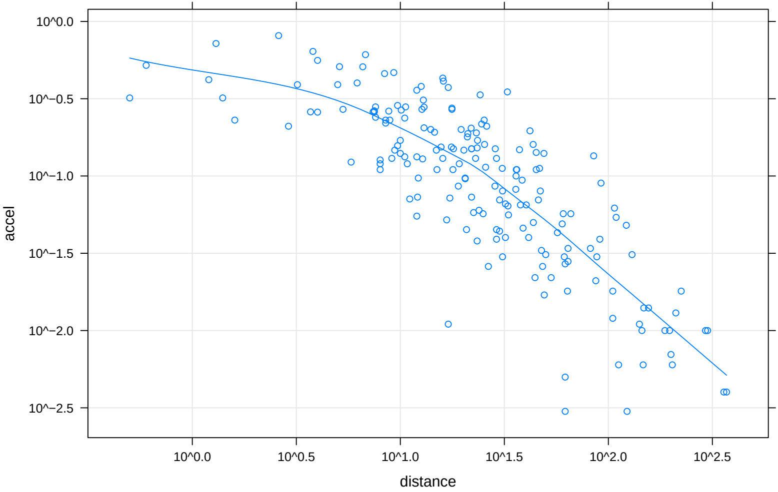
```
> dotplot(VADeaths, type = "o",
          auto.key = list(points = TRUE, lines = TRUE,
                          space = "right"))
```



```
> data(Earthquake, package = "nlme")
> xyplot(accel ~ distance, data = Earthquake)
```

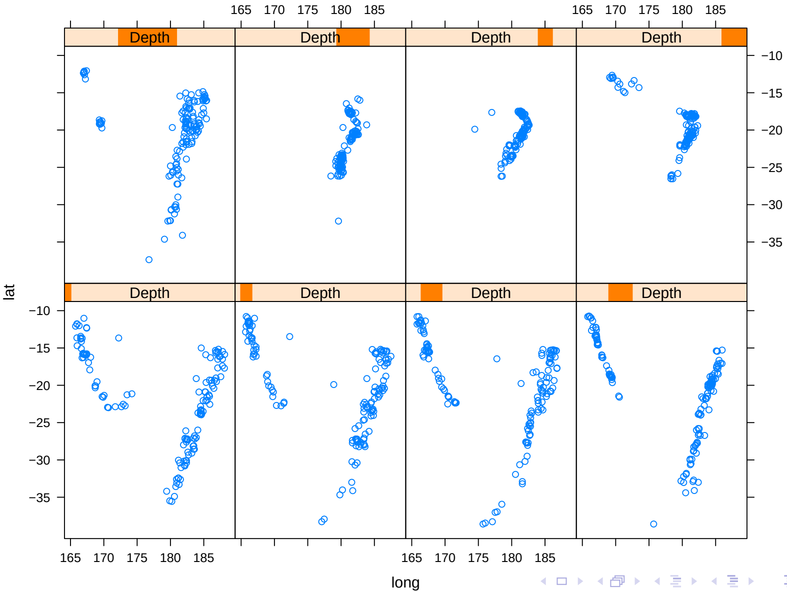


```
> xyplot(accel ~ distance, data = Earthquake,
         scales = list(log = TRUE),
         type = c("p", "g", "smooth"))
```

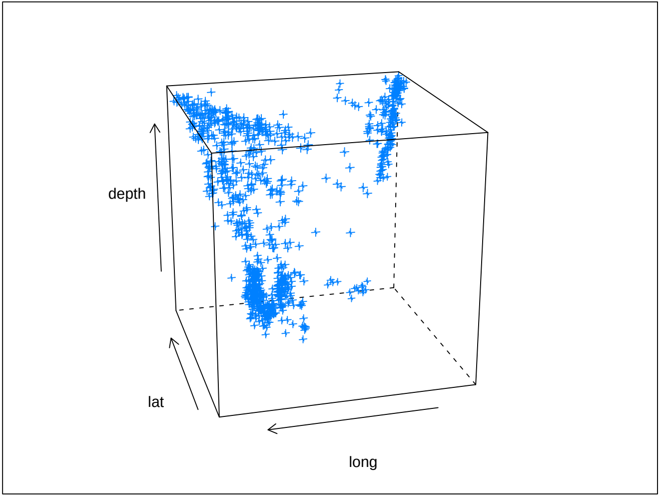


```
> Depth <- equal.count(quakes$depth, number=8, overlap=.1)
> summary(Depth)

Intervals:
    min   max count
1  39.5  63.5   138
2  60.5 102.5   138
3  97.5 175.5   138
4 161.5 249.5   142
5 242.5 460.5   138
6 421.5 543.5   137
7 537.5 590.5   140
8 586.5 680.5   137

Overlap between adjacent intervals:
[1] 16 14 19 15 14 15 15
```

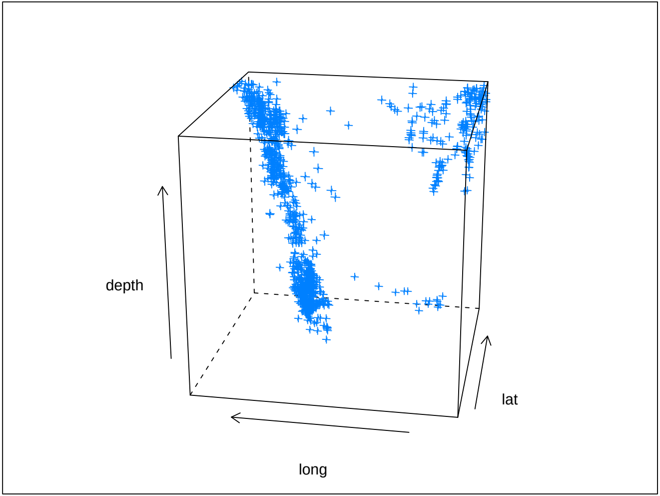
```
> xyplot(lat ~ long | Depth, data = quakes)
```



```
> cloud(depth ~ lat * long, data = quakes,
        zlim = rev(range(quakes$depth)),
        screen = list(z = 105, x = -70), panel.aspect = 0.75)
```



```
> cloud(depth ~ lat * long, data = quakes,
        zlim = rev(range(quakes$depth)),
        screen = list(z = 80, x = -70), panel.aspect = 0.75)
```



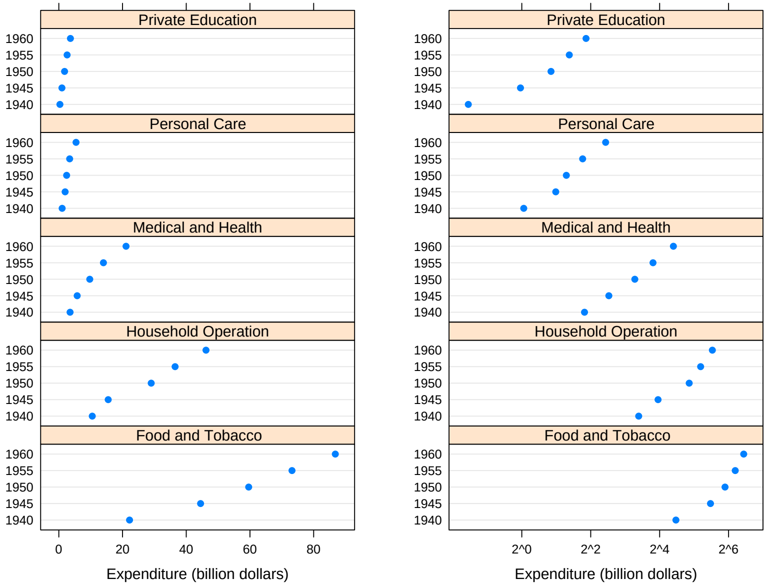
More high-level functions

- More high-level functions in lattice
    - Won't discuss, but examples in manual page
- Other Trellis high-level functions can be defined in other packages, e.g.,
    - ecdfplot(), mapplot() in the latticeExtra package
    - hexbinplot() in the hexbin package

The *"trellis"* object model

- One important feature of lattice:
  - High-level functions do not actually plot anything
  - They return an object of class *"trellis"*
  - Display created when such objects are print()-ed or plot()-ed
- Usually not noticed because of automatic printing rule
- Can be used to arrange multiple plots
- Other uses as well

```
> dp.uspe <-
      dotplot(t(USPersonalExpenditure),
              groups = FALSE, layout = c(1, 5),
              xlab = "Expenditure (billion dollars)")
> dp.uspe.log <-
      dotplot(t(USPersonalExpenditure),
              groups = FALSE, layout = c(1, 5),
              scales = list(x = list(log = 2)),
              xlab = "Expenditure (billion dollars)")
> plot(dp.uspe,     split = c(1, 1, 2, 1))
> plot(dp.uspe.log, split = c(2, 1, 2, 1), newpage = FALSE)
```


Private Education
Personal Care
Medical and Health
Household Operation
Food and Tobacco
1960
1955
1950
1945
1940
0
20
40
60
80
2^0
2^2
2^4
2^6
Expenditure (billion dollars)

Trellis Philosophy: Part I

- Display specified in terms of
  - Type of display (histogram, densityplot, etc.)
  - Variables with specific roles
- Typical roles for variables
  - Primary variables: used for the main graphical display
  - Conditioning variables: used to divide into subgroups and juxtapose (multipanel conditioning)
  - Grouping variable: divide into subgroups and superpose
- Primary interface: high-level functions
  - Each function corresponds to a display type
  - Specification of roles depends on display type
    - Usually specified through the formula and the groups argument

Trellis Philosophy: Part II

- Design goals:
  - Enable effective graphics by encouraging good graphical practice (e.g., Cleveland, 1985)
  - Remove the burden from the user as much as possible by building in good defaults into software
- Some obvious examples:
  - Use as much of the available space as possible
  - Encourage direct comparsion by superposition (grouping)
  - Enable comparison when juxtaposing (conditioning):
    - use common axes
    - add common reference objects (such as grids)
- Inevitable departure from traditional R graphics paradigms

Trellis Philosophy: Part III

- Any serious graphics system must also be flexible
- lattice tries to balance flexibility and ease of use using the following model:
  - A display is made up of various elements
  - Coordinated defaults provide meaningful results, but
  - Each element can be controlled independently
  - The main elements are:
    - the primary (panel) display
    - axis annotation
    - strip annotation (describing the conditioning process)
    - legends (typically describing the grouping process)

- The full system would take too long to describe
- Handouts cover the important aspects
- Online documentation has details; start with ?Lattice
- We discuss a few advanced ideas using some case studies

Case studies

- Adding regression lines to scatter plots
- Reordering levels of a factor

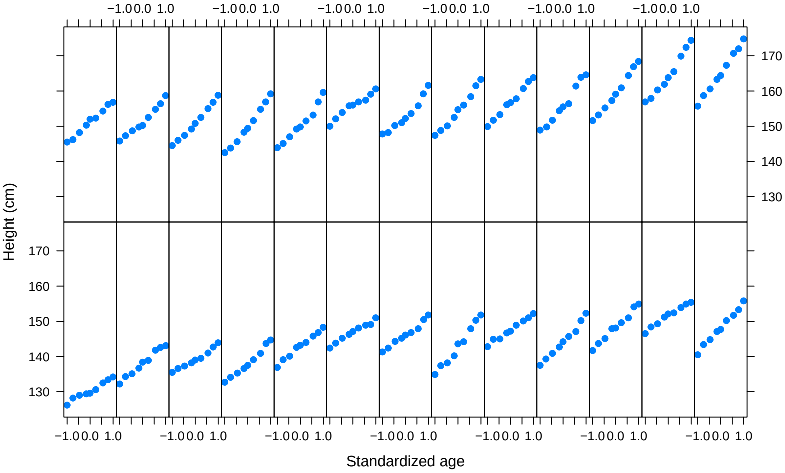
Example 1: Growth curves

- 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", pch = 16,
         xlab = "Standardized age", ylab = "Height (cm)")
```



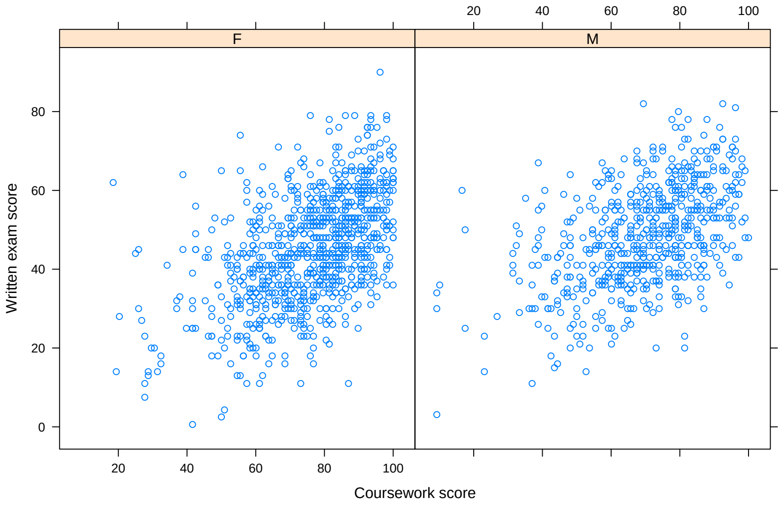
Example 2: Exam scores

- 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

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()

A simple panel function

- So, equivalent call:

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

A simple panel function

- So, equivalent call:

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

A simple panel function

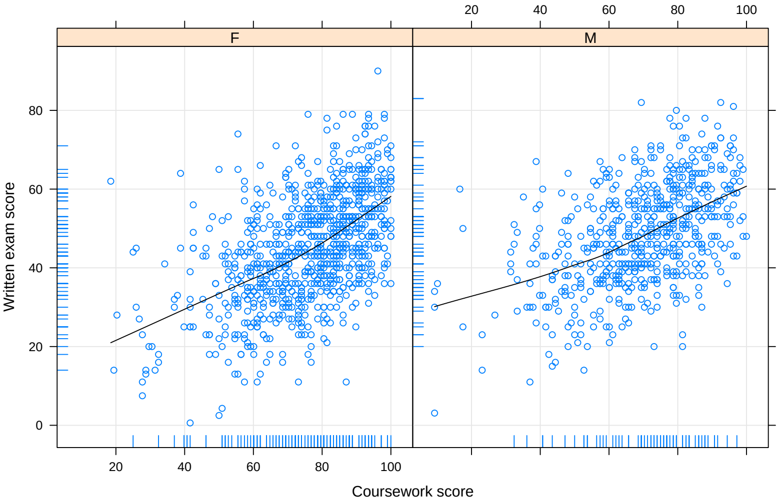
- So, equivalent call:

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

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.loess(x, y, ..., col = "black")
             panel.rug(x = x[is.na(y)],
                       y = y[is.na(x)])
         })
```


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

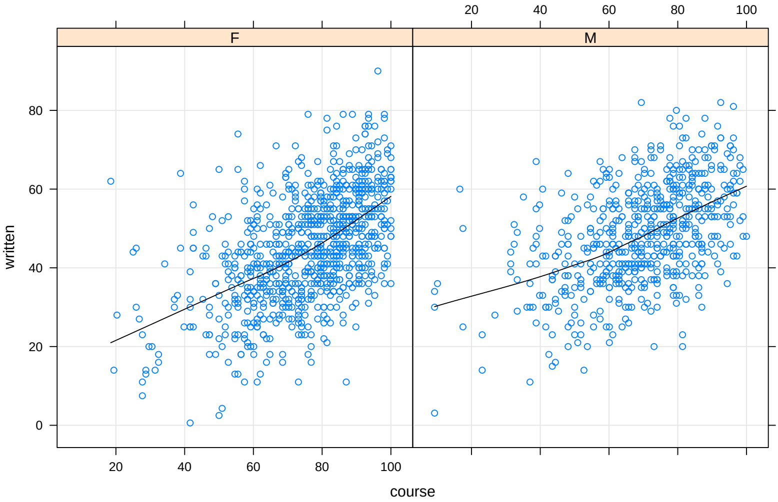
Panel functions

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
- Each high-level function has a corresponding *default* panel function, named as "panel." followed by the function name. For example,
  - histogram() has panel function panel.histogram
  - dotplot() has panel function panel.dotplot
- Most have useful arguments that support common variants

Back to regression lines

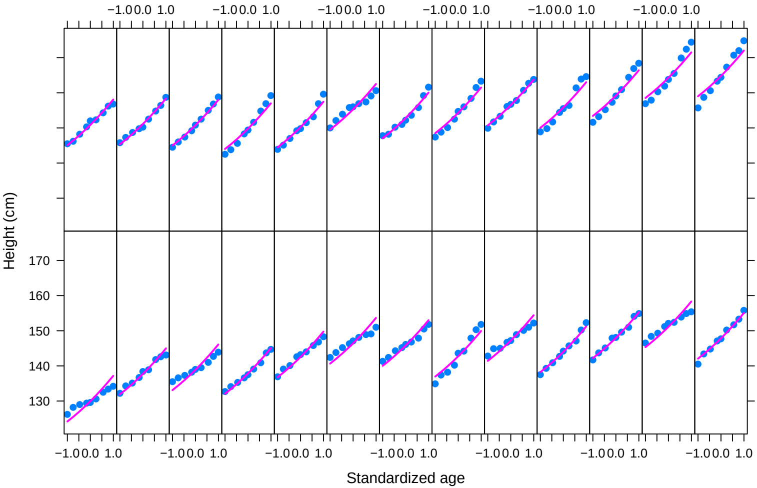
- Oxboys: model height on age

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

- Mixed effect model that can be fit with lme4

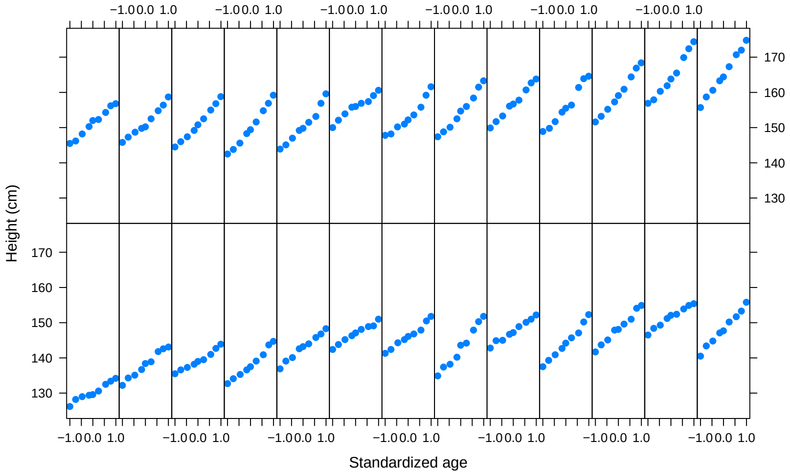
```
> library(lme4)
> fm.poly <-
      lmer(height ~ poly(age, 2) + (1 | Subject),
           data = Oxboys)
```

- Goal: plot of data with fitted curve superposed

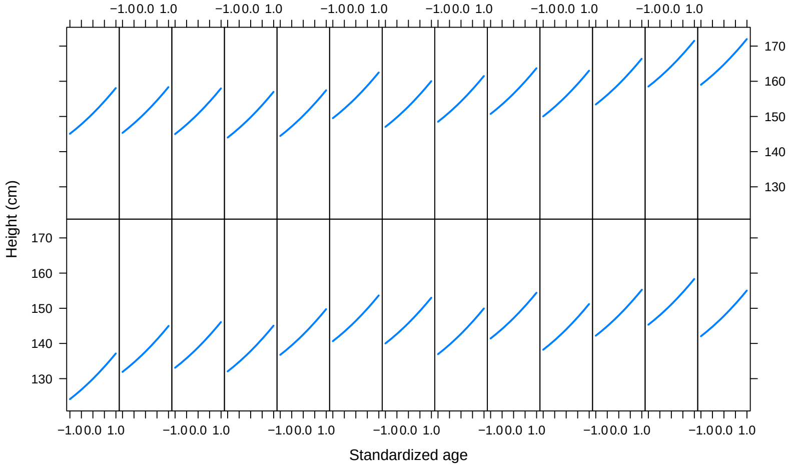


Height (cm)
Standardized age
−1.0 0.0 1.0
130
140
150
160
170

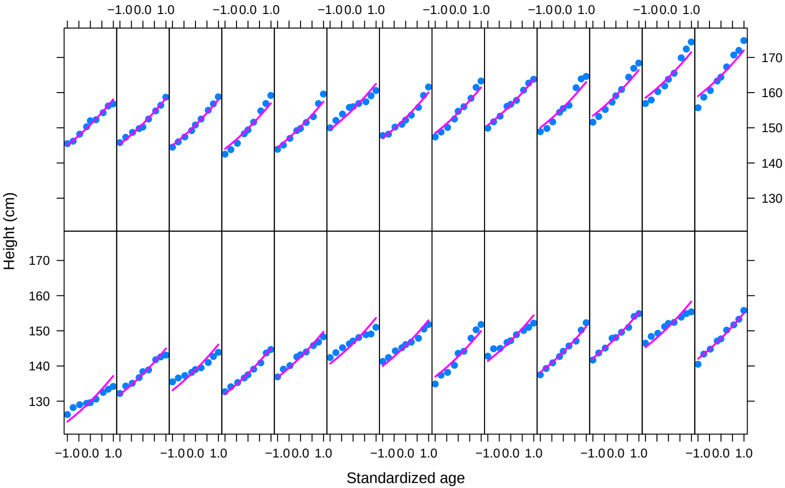
```
> xyplot(height ~ age | Subject,
         data = Oxboys, strip = FALSE, aspect = "xy",
         type = "p", pch = 16,
         xlab = "Standardized age", ylab = "Height (cm)")
```



```
> xyplot(fitted(fm.poly) ~ age | Subject,
         data = Oxboys, strip = FALSE, aspect = "xy",
         type = "l", lwd = 2,
         xlab = "Standardized age", ylab = "Height (cm)")
```



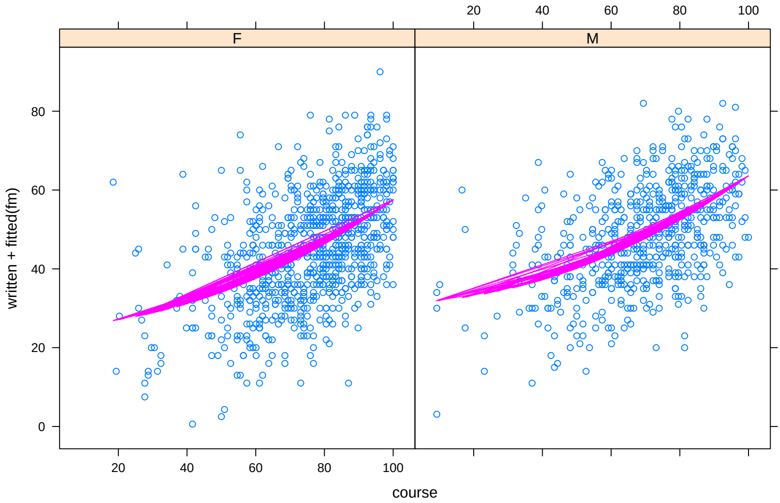
```
> xyplot(height + fitted(fm.poly) ~ age | Subject,
         data = Oxboys, strip = FALSE, aspect = "xy", pch = 16,
         lwd = 2, type = c("p", "l"), distribute.type = TRUE,
         xlab = "Standardized age", ylab = "Height (cm)")
```



GCSE exam scores

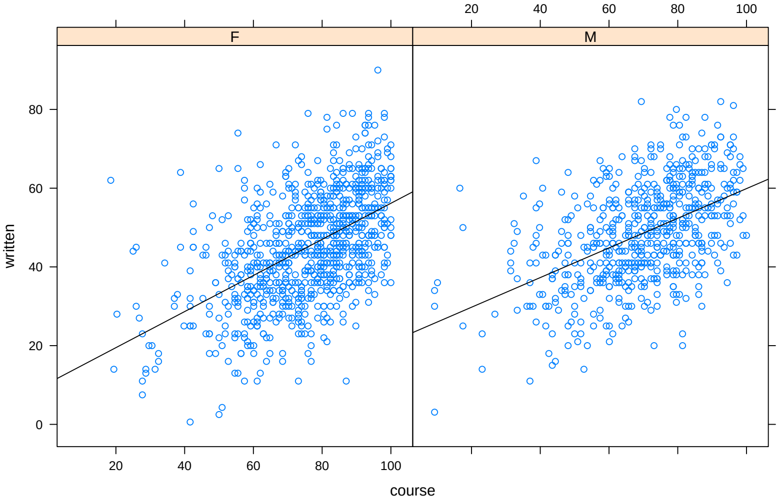
- Gcsemv: model written score by coursework and gender
- A similar approach does not work as well
  - x values are not ordered
  - missing values are omitted from fitted model

```
> 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")
```

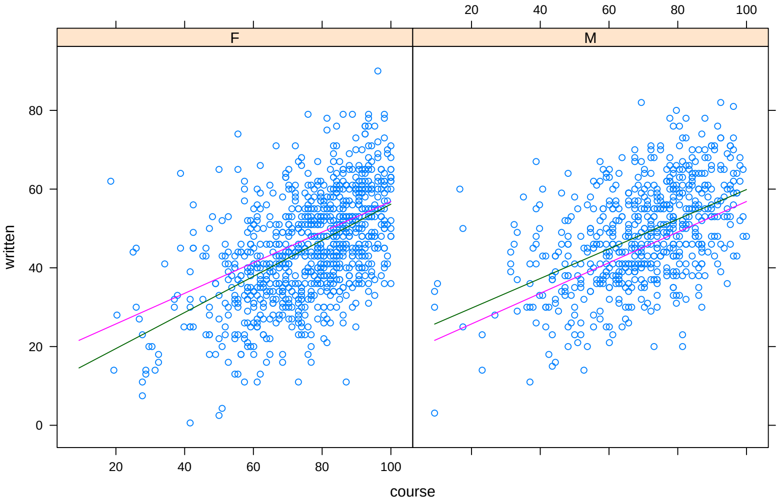


GCSE exam scores

- 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
100

- Solution: evaluate fits separately and combine

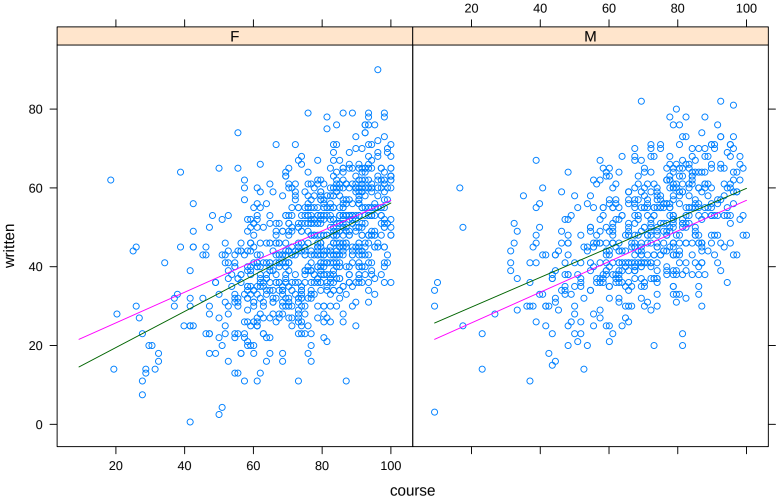
```
> course.rng <- range(Gcsemv$course, finite = TRUE)
> grid <-
      expand.grid(course = do.breaks(course.rng, 30),
                  gender = unique(Gcsemv$gender))
> fm0.pred <-
      cbind(grid,
            written = predict(fm0, newdata = grid))
> fm1.pred <-
      cbind(grid,
            written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



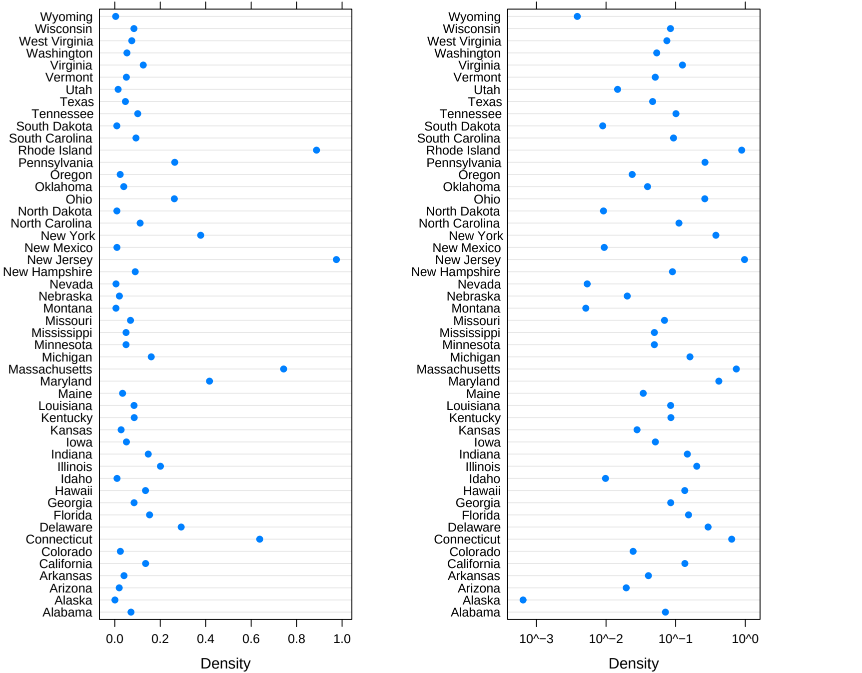
Reordering factor levels

- 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

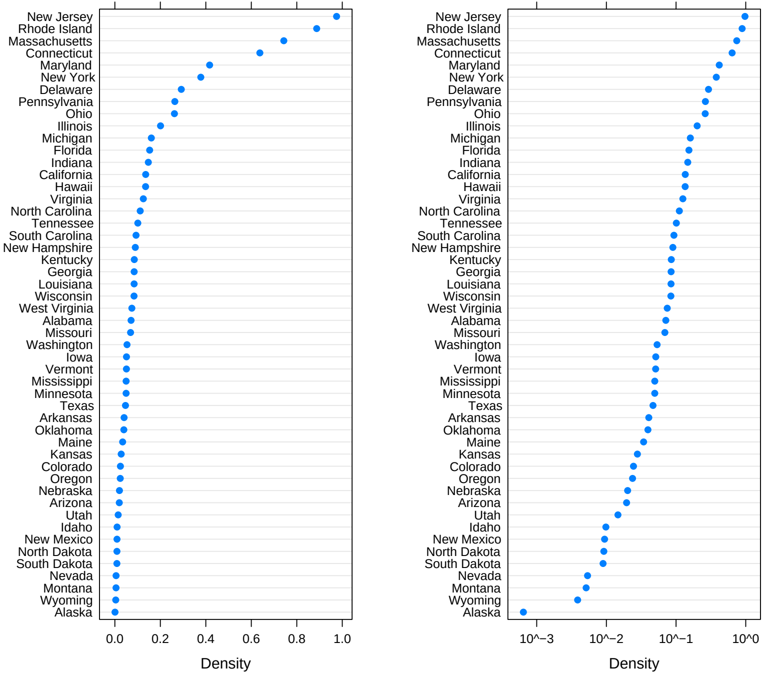
Example

- 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
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

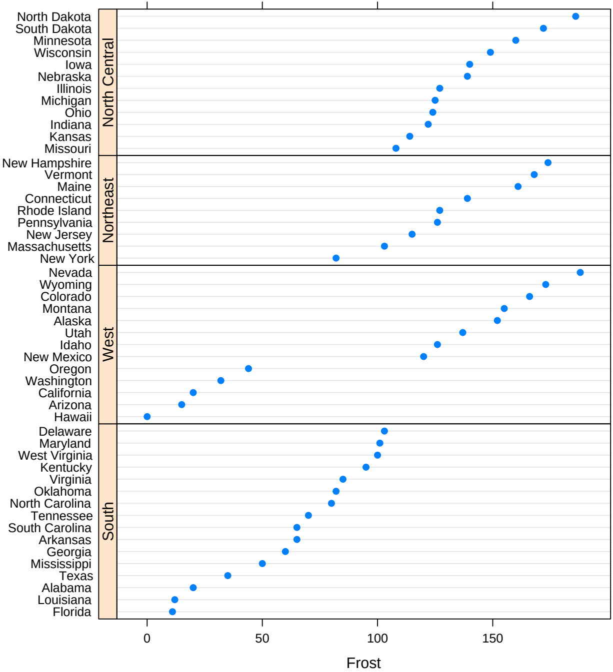
```
> dotplot(reorder(name, Density) ~ Density, state)
> dotplot(reorder(name, Density) ~ Density, state,
          scales = list(x = list(log = TRUE)))
```

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

Reordering by multiple variables

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

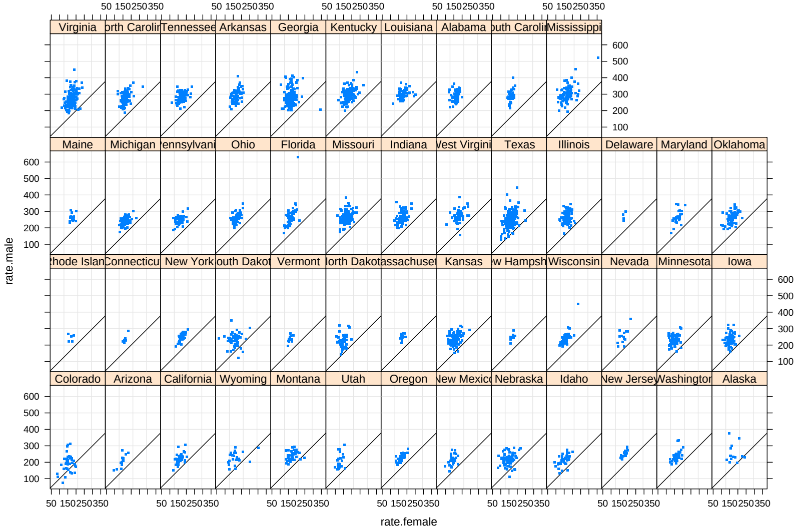
```
> state$region <- with(state, reorder(region, Frost, median))
> state$name <- with(state,
                     reorder(reorder(name, Frost),
                             as.numeric(region)))
> p <-
    dotplot(name ~ Frost | region, state,
            strip = FALSE, strip.left = TRUE, layout = c(1, 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
assachuset
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

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
- Several useful functions make life a little simpler
  - reorder(), make.groups(), etc.