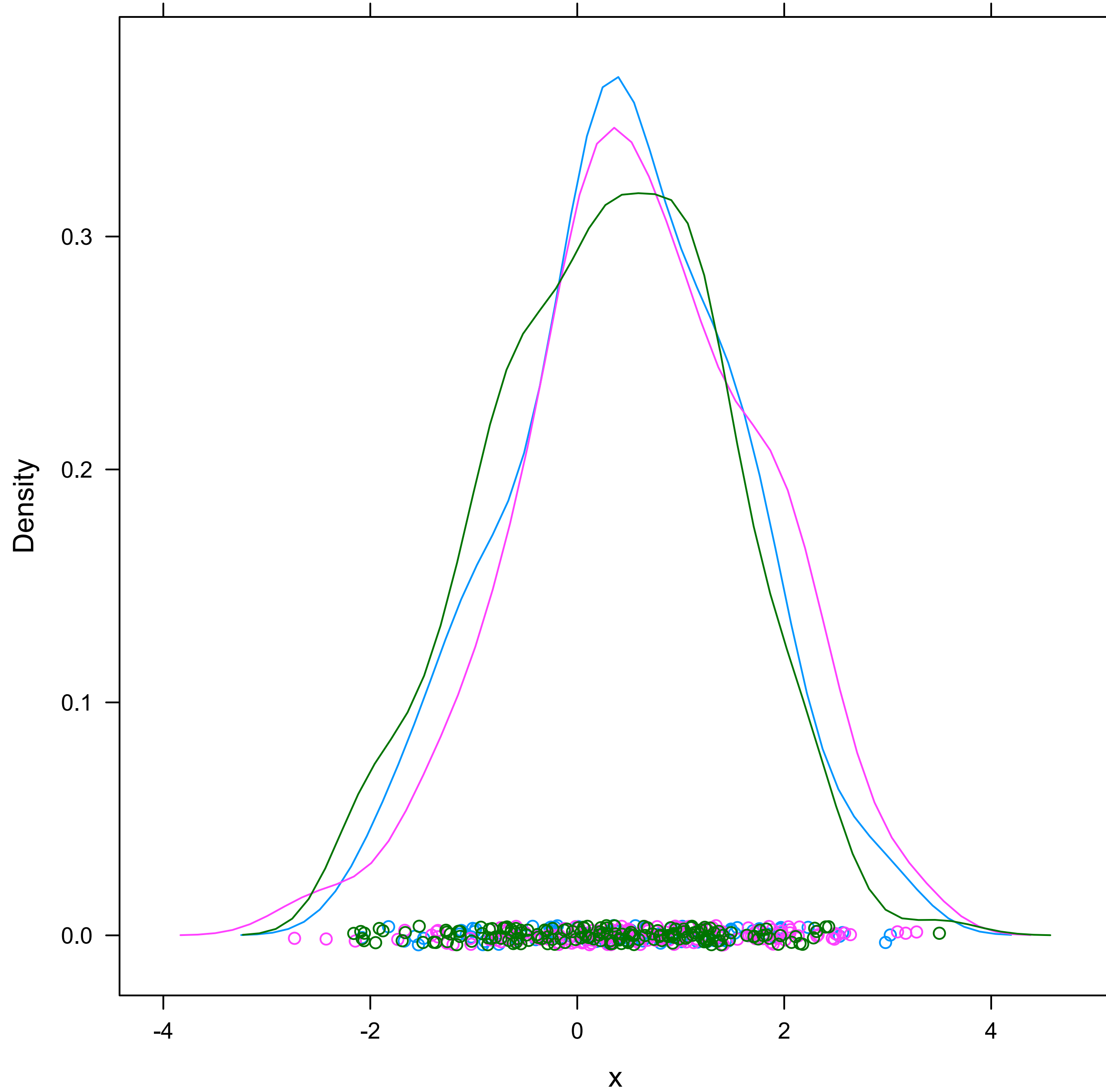
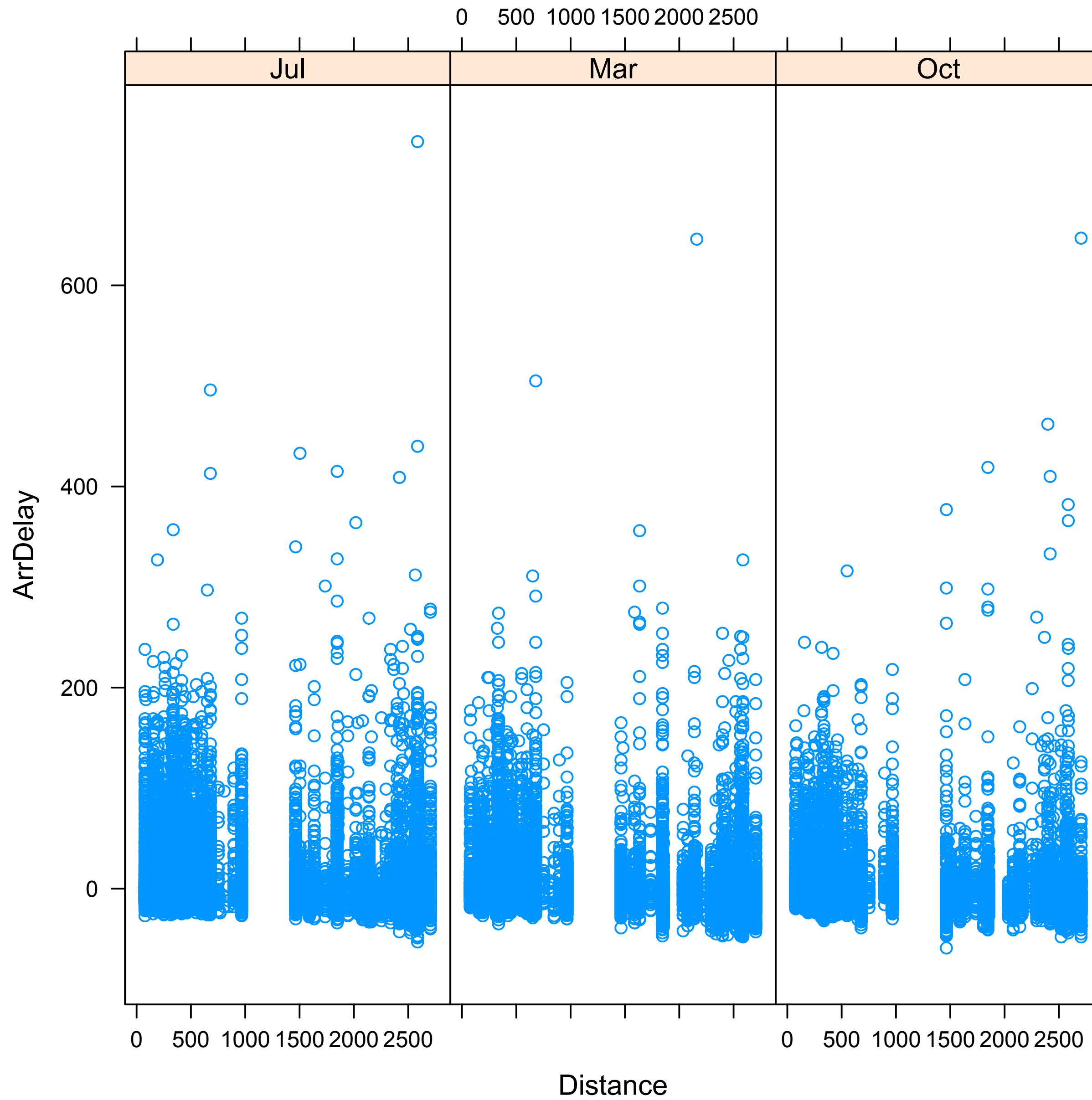
Alternative Graphics System
Lattice

- Lattice/trellis is another high-level graphics system that makes many complex things easy, but annotating plots can be initially complex.
- This material is "optional". Feel free to use it but don't feel you have to.

- Consider the following (simulated) data
  - we have 3 groups of 100, 150 and 175 people
  - we measure their blood pressure
- Hard to put into a data frame or matrix because the groups are of different size.
- So hard to use matplot().


Density
0.3
0.2
0.1
0.0
-4
-2
0
2
4
x

- Draw density of the first
- plot(density(bp [[“A”]]))
- Then add the other densities
  - lines(density(bp[[“B”]]))
    lines(density(bp[[“C”]]))
- Have to compute the xlim and ylim to handle all 3 groups



Jul
Mar
Oct
ArrDelay
0
200
400
600
0 500 1000 1500 2000 2500
Distance

- lattice is an R package and provides many functions for creating high-level plots
- library(lattice)
- xyplot(Arr ~ x, data)
- Note that we are using a formula to specify the variables for the horizontal and vertical axes.
- We are specify the data frame in which these are found.
- (Or if the variables are in the workspace, we don't have to specify the data.)

Plot types & Functions

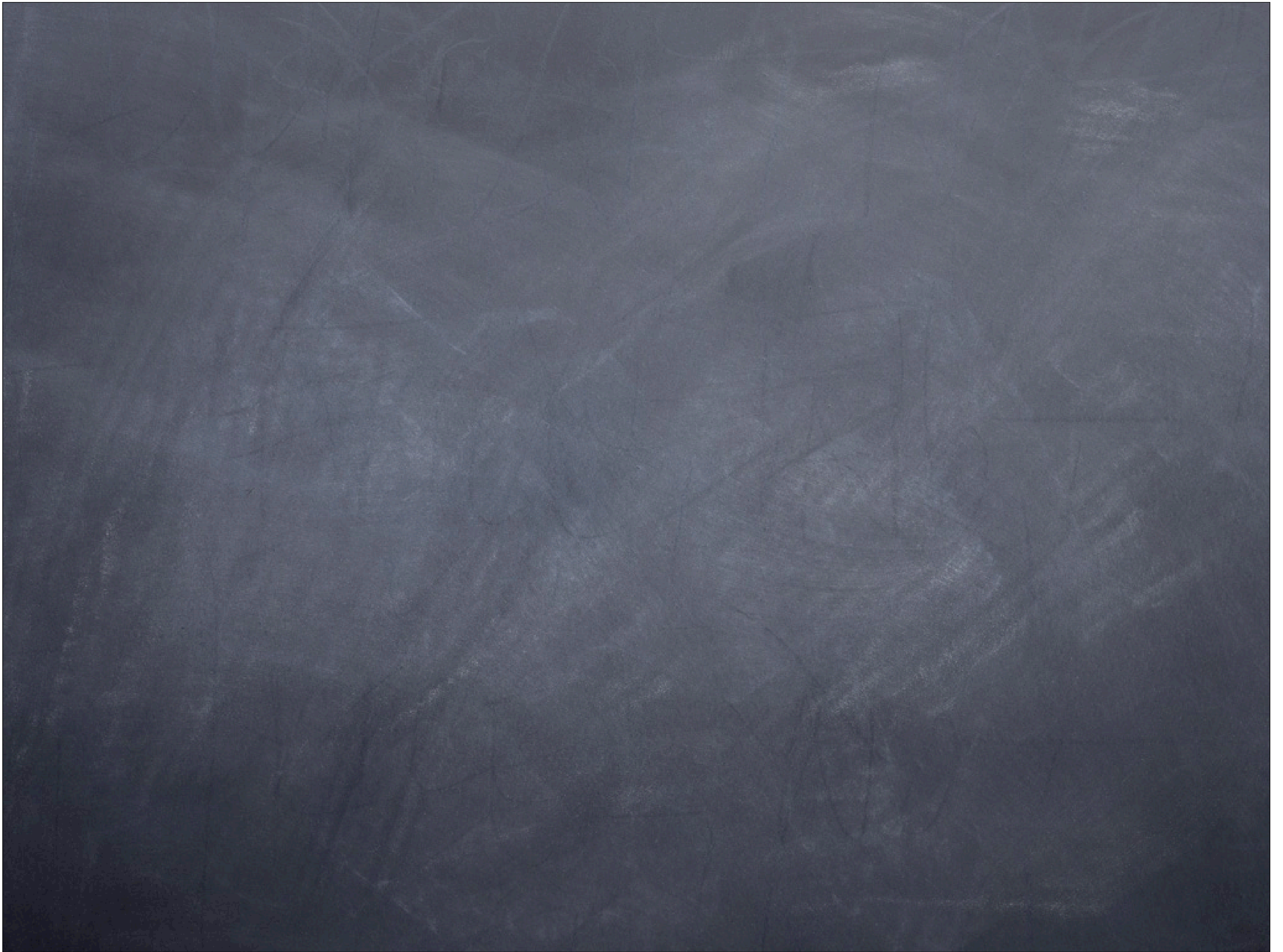
- histogram(), densityplot(), bwplot(), barchart()
- stripplot(), dotplot()
- qq(), qqmath()
- bwplot()
- xyplot()
- splom()
- contourplot() & levelplot()
- cloud() & wireframe()
- parallel()

- histogram(~ AirTime, sfo.origin)
- densityplot(~ AirTime, sfo.origin, plot.points = FALSE)
- bwplot(DayOfWeek ~ ArrDelay, air08)
- splom(~ ArrDelay + DepDelay + Cancelled, sfo.origin)

- You can use lattice as a replacement for many of the “traditional” plots
  - easy to display different groups on the same plot (via the groups argument.
  - easy to add legends
  - you get “better” default colors, etc.

- Lattice functions accept common “traditional” arguments
  - xlab, ylab, main,
  - xlim, ylim
  - col, pch

- Like the par() command, lattice has facilities for setting global parameters for use in subsequent plots
- We use trellis.par.set() and trellis.par.get()
- Find the available settings with
  names(trellis.par.get())
- show.settings()
- Values are typically lists() with entries such as "alpha", "cex", "col", "font", "lineheight"

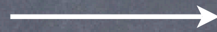


groups & superposition

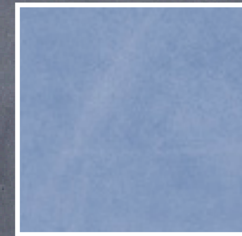
- Recall matplot(), or points(density(x2)), or plot(x, y, col = type) ...
  where we draw multiple variables/data on the same plot in different colors.
- Lattice makes this quite simple via the groups argument.
- This takes a variable (in the data frame of the data argument) and it separates the data based on the "levels"of this. Then it draws the plot for each of these subgroups.

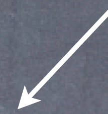
densityplot(~ x, groups = type)

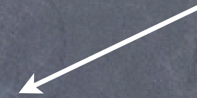
All data

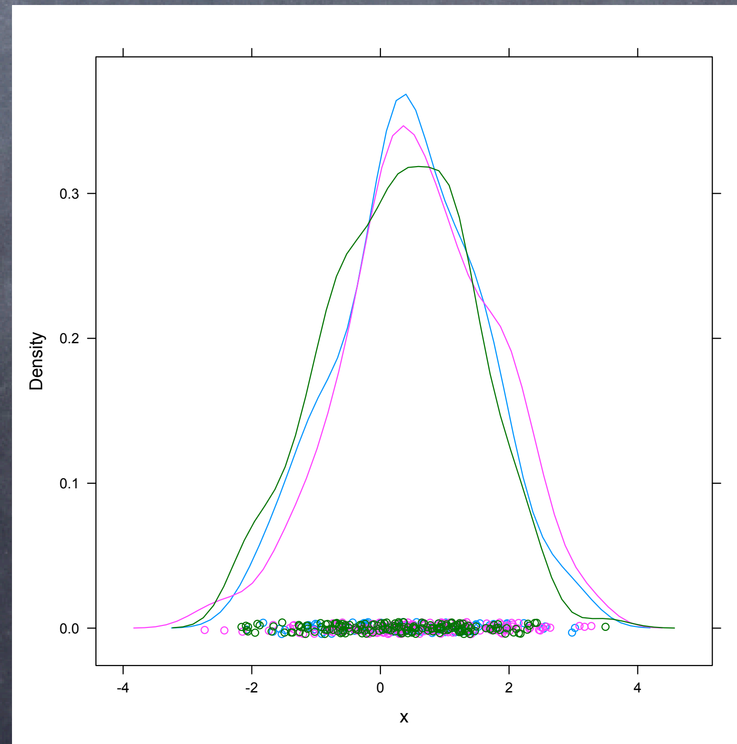


Subgroups





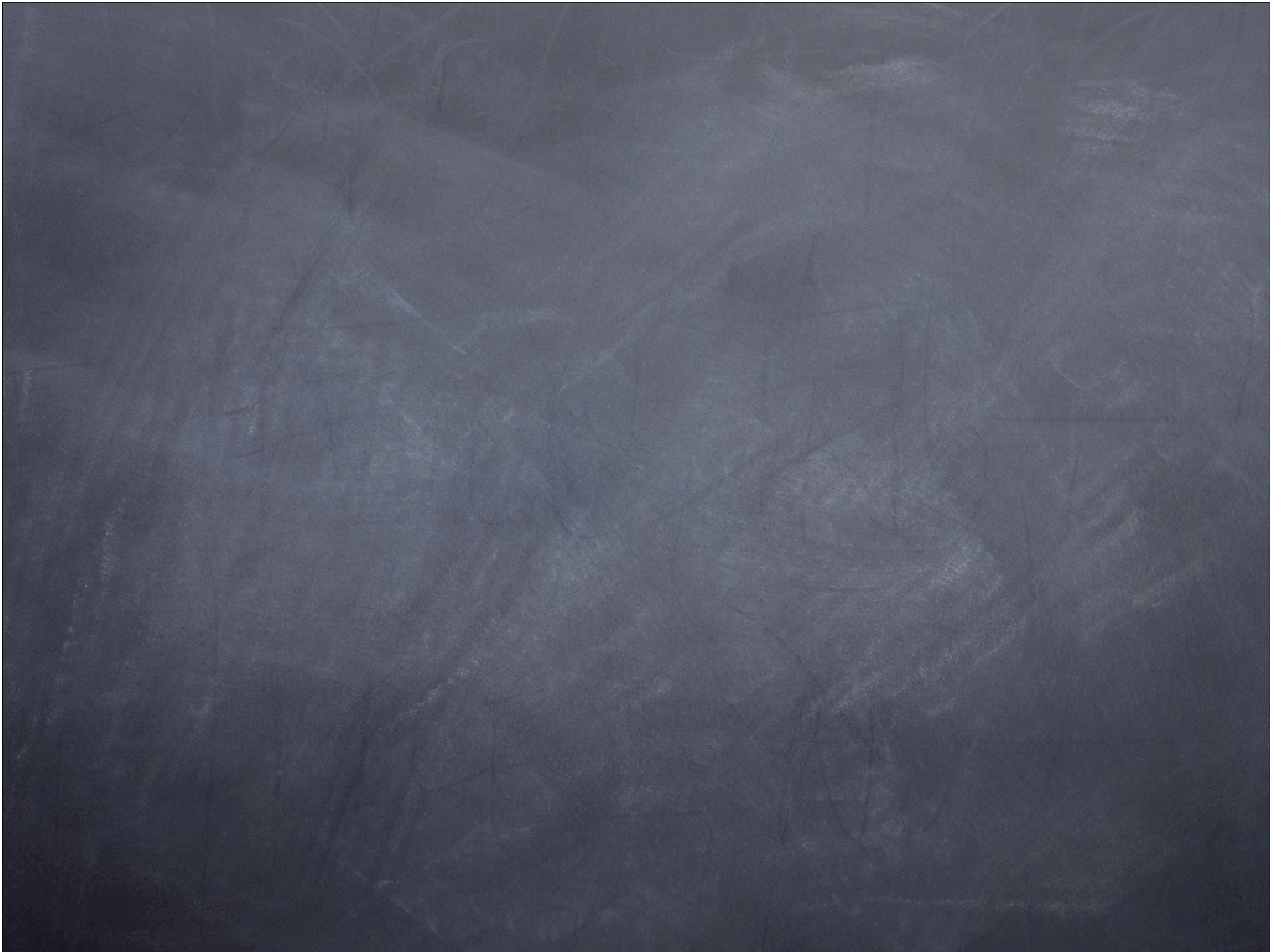




- The groups = var causes lattice to take care of collecting the different subgroups, computing the appropriate xlim and ylim and drawing the different pieces of the display.
- If we use
```
xyplot( y ~ x, data, groups = var)
```
we will get different colored points corresponding to the different levels of ‘var’.
- Of course, we need a legend to map the colors to the values of var....

Legends/keys

- Lattice provides very simple and also very advanced facilities for adding legends.
- It can take care of adding space to the plot, laying it out and filling in the pieces as part of a regular top-level plot.
- Use the auto.key parameter.
- densityplot(~ x, groups = type, auto.key = TRUE)
- densityplot(~ x, groups = type,
  auto.key = list(columns = length(levels(type)),
  title = "Type", space = "right"))

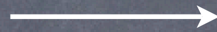


Separate plots - conditioning

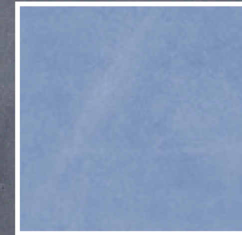
- Instead of using groups to put different sub-groups in the same display, we might have multiple different & separate panels showing different subsets.
- We do this with "conditional" plots.
- We divide the data into subgroups, but then we draw separate displays for each of these subgroups.

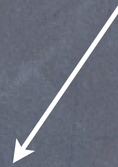
densityplot(~ x | type, D)

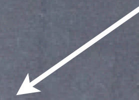
Subgroups

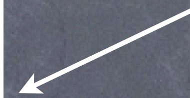


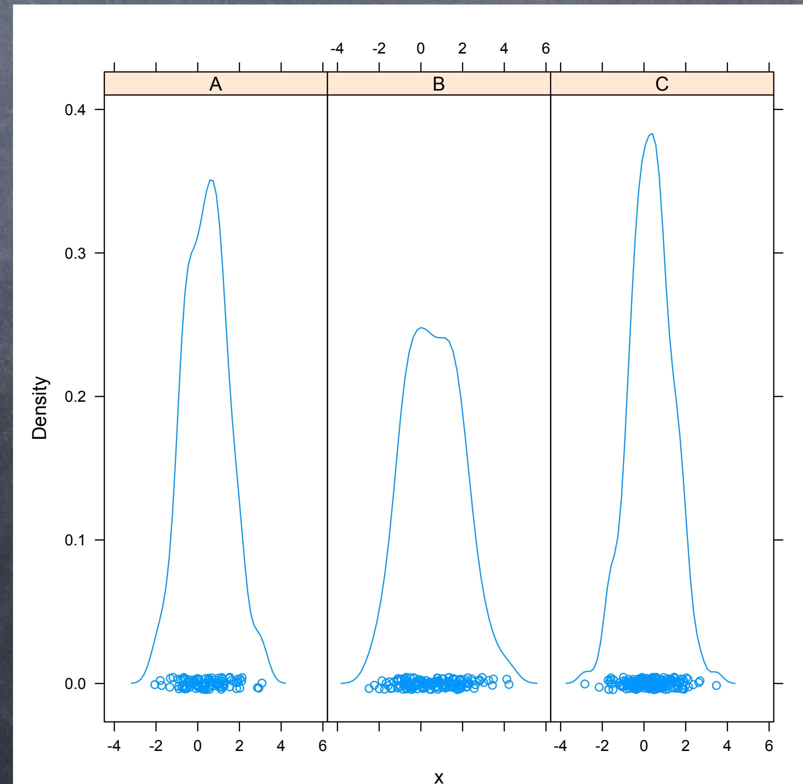
All data





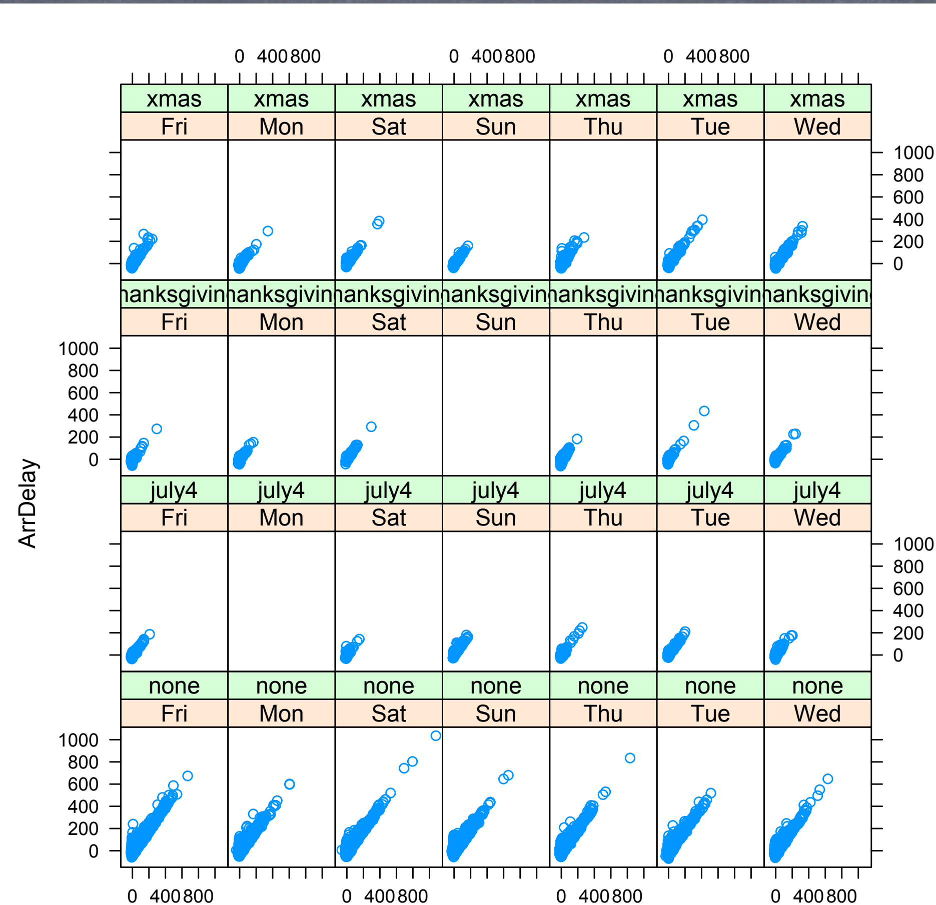






Multiple conditions

- We can condition on more than one variable
- xyplot(ArrDelay ~ DepDelay | DayOfWeek + holiday)



- We can use both conditioning and groups simultaneously.
- densityplot(~ ArrDelay | Month, sfo.origin,
  groups = UniqueCarrier)

Last (main) thing - layout

- When we use a conditional component of our formula, we end up with multiple panels.
- Lattice arranges these on the screen (/device). But we can control this with
-
```
xyplot(y ~ x | a + b + ..., data, groups = w,
       layout = c(columns, rows))
```

One last thing - panel functions

- How do we add text, lines, etc. to panels in our lattice plot?
- This is where lattice is a little more complex than "traditional" graphics, but is actually quite simple.
- We can draw all the panels and then go back to individual panels and draw on them.
  - use trellis.focus() or panel.identify()

```
trellis.focus("panel", 2, 1) # what, column and row
panel.abline( v = 100, col = "red")
trellis.unfocus()
```

- Instead of text(), abline(), points(), lines(), ...
  we use panel.text(), panel.abline(), panel.points(), panel.lines(), ...

- Alternative is to annotate each panel as we are render it.
- We do this by specifying our own panel function
- Often this is relatively straightforward

```
xyplot(y ~ x | var1 + var2, data,
         panel = function(...) {
                   panel.xyplot(...)
                   panel.abline( v = 100, col = "red")
                 })
```

- Our function calls the regular/default panel function for the plot type, then adds its own content.
- Often our annotations will access the data passed in the arguments.

What does a lattice function do?

- Examines the formula and identifies the conditional parts (i.e. the parts after the ‘|’)
- Separate the data into subsets based on the conditioning combinations - for panels.
- Determine the xlim and ylim for each subset.
- Compute the common xlim, ylim
- Create a panel for each subset, using common xlim,ylim.
- If there is a groups argument, create subsets within this subset. Draw elements within the panel.

```
print(xyplot(percent ~ percent | year,state.props.df,
       xlim = m$range[1:2], ylim = m$range[3:4],
       subscripts = TRUE,
       subset = year %in% seq(1992, by =4, len = 5),
       panel = function(x,y, ..., subscripts) {

          m = map('state', plot = FALSE, fill = TRUE)
          i = match(gsub(":.*", "", m$names),
                    tolower(state.props.df
$state[subscripts]))
          col = rgb(1-x, 0, x)[i]
          col[is.na(col)] = "#000000" # black
          panel.polygon(cbind(m$x, m$y), col = col)
        },
        strip = function(which.panel, ...) {
           year = c(1992, 1996, 2000, 2004, 2008)
[which.panel]
           grid.rect(gp = gpar(fill = "grey"))
           ltext(.5, .5, year)
          },
        scales = list(draw = FALSE), aspect = "iso"))
```