

# Freeway Traffic in California

- Loop detectors at 22,000 locations,
- Transmit data every 30 seconds
- Collect 2GB a day, and store 4TB

For each of three lanes,  
**flow** (number of cars) and  
**occupancy** (the proportion of time there was a car over the loop)  
were recorded in successive five minute intervals.

We have 1740 such five minute intervals.

Lane 1 is the leftmost lane, lane 2 is in the center, and lane 3 is the rightmost.

# Tasks

- Read the data directly from the web into R. Explain why you chose the function you did.
- Reshape the data (1740\*3), with lane as a factor and day and hour as numeric.
- Which lane typically serves the most traffic?
- Flow can be regarded as a measure of the throughput of the system. How does this throughput depend on congestion?
- Taxi drivers claim that when traffic breaks down, the fast lane breaks down first so they move immediately to the right lane. Can you see any such phenomena in the data?

# Deconstruct- Reconstruct a Plot

## California majority party by county

Graph

Table

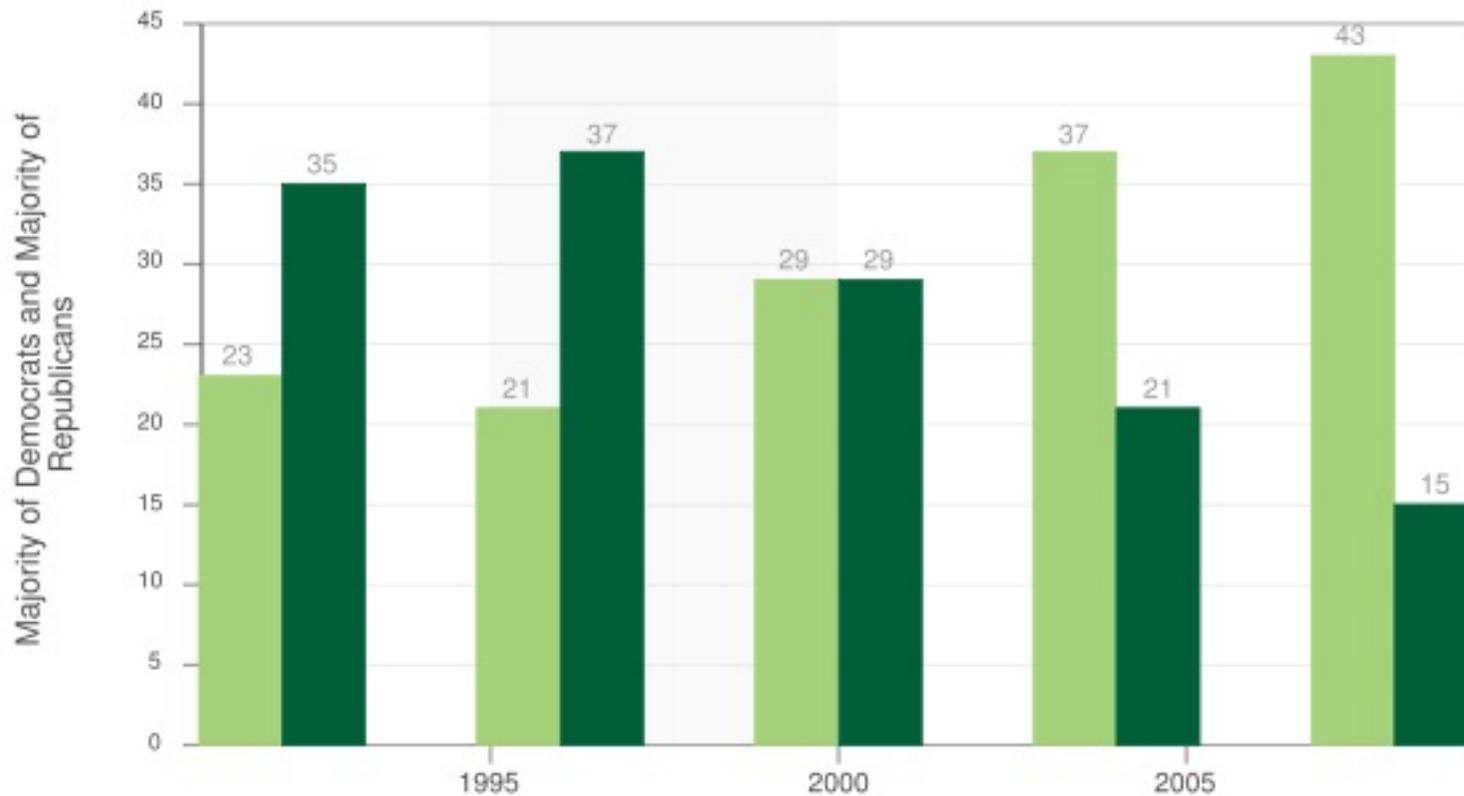
Cloud

Map

Absolute Relative All 12y 8y 1y

>

More Op



## Registered Voters in California

Majority of Democrats	Majority of Republicans	Election Year
21, 37		"2004"
23, 35		"2008"
29, 29		"2000"
37, 21		"1996"
43, 15		"1992"

Sources: California Secretary of State

[http://www.sos.ca.gov/elections/ror/  
60day\\\_presprim/hist\\\_reg\\\_stats.pdf](http://www.sos.ca.gov/elections/ror/60day\_presprim/hist\_reg\_stats.pdf)

What's the message?

Can you improve upon it?

# Critique

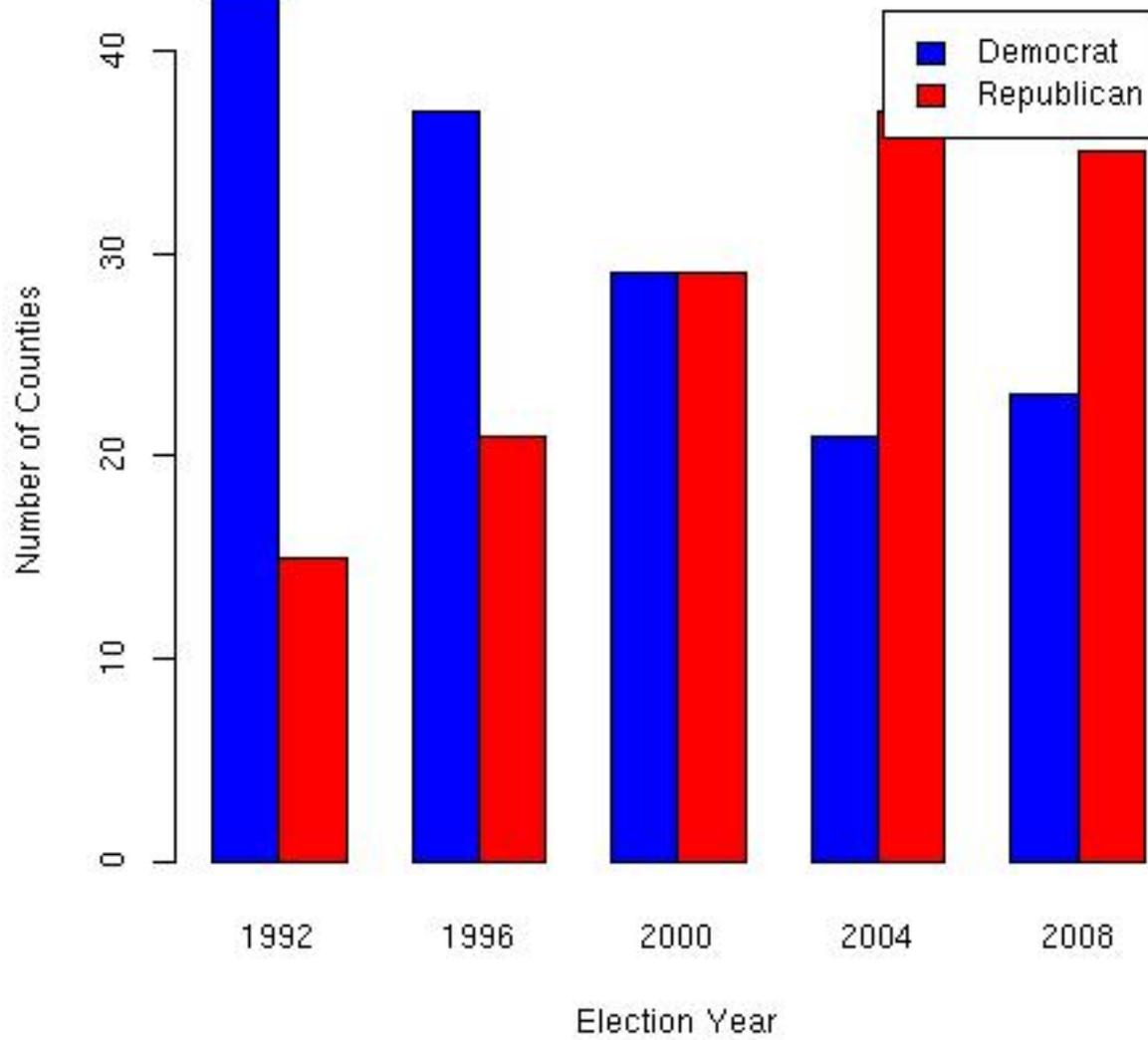
## Basic plotting issues

- x-axis tick marks poorly located - should be located at election years
- y-axis label misleading - it is number of counties
- use of color could be improved with red/blue recognizable party colors
- data are turned around, i.e. figures for 1996 are really 2008 data

Message: how party registration has changed over the past 5 presidential elections

- More informative if we have registration figures as people vote not counties
- County size may be a lurking variable - small counties tend to be rural and conservative

### California Counties Majority Party of Registered Voters

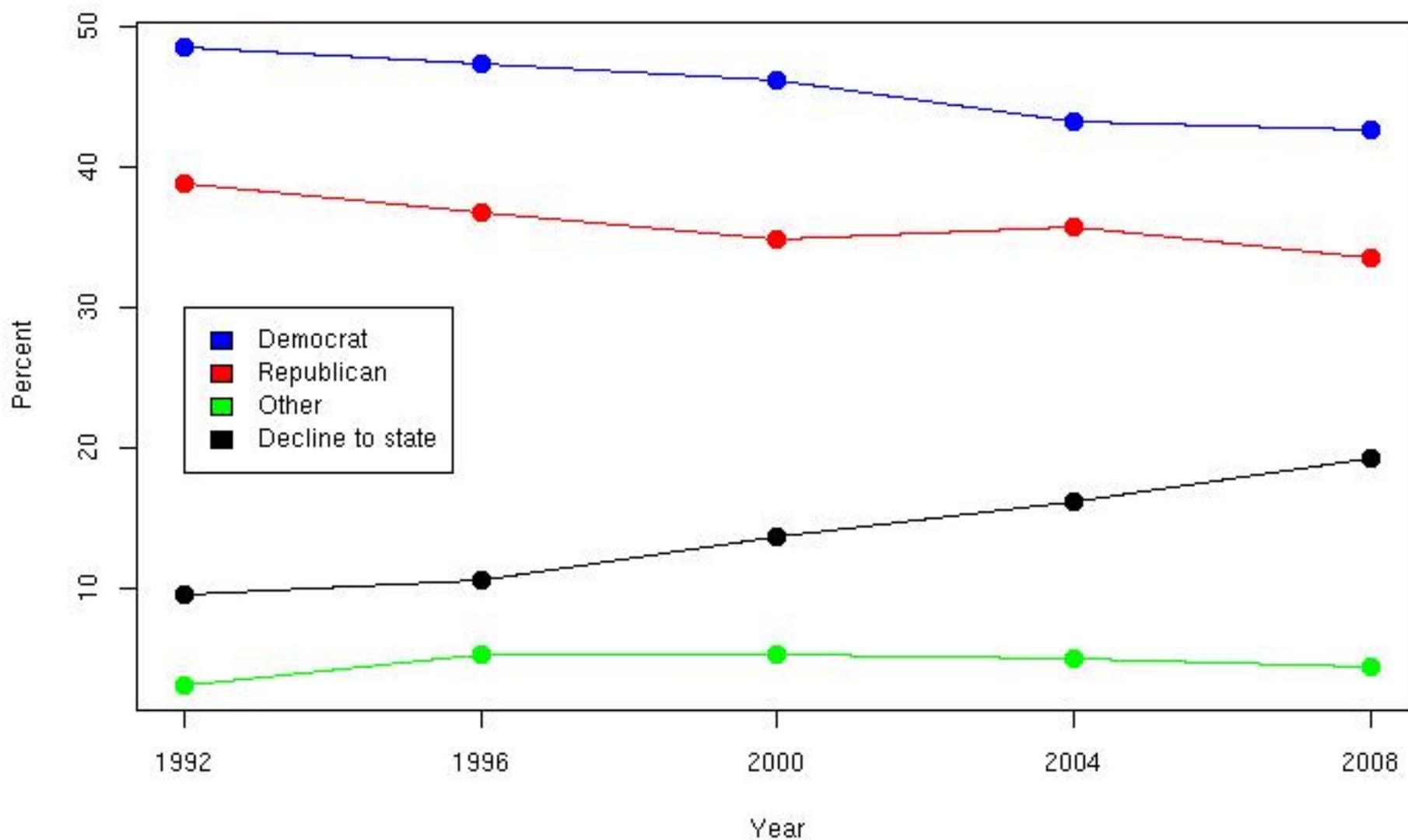


## Alternative Data from same report

Notice that the Other registrations and the “Decline to State” registrations make up nearly 25% of the registrations in 2008. Leaving these party affiliations out of the plot distorts the picture.

year,	eligible,	registered,	dem,	rep,	other,	decline
1992,	20612814,	13217022,	.485,	.389,	.031,	.095
1996,	19298379,	14314658,	.474,	.368,	.052,	.106
2000,	21190865,	14676174,	.462,	.349,	.052,	.137
2004,	21843202,	14945031,	.432,	.357,	.049,	.162
2008,	22987562,	15468551,	.427,	.336,	.044,	.193

### Party Affiliation of Registered Voters in California



## Motivation for this assignment

- How to critique a plot for effectiveness is not systematized, and is difficult to teach.
- Important to demonstrate how to figure out what went wrong, not just point out mistakes
- Find it can really highlight the point of a data analysis by answering the questions:
  - What is the message?
  - Is there a better comparison for bringing out the message?
- Acts as a good introduction to the basic plotting model in R

## Learning objectives

- Evaluate of effectiveness of a plot
- Identify the message in a plot, and improve the graphic to better present the message
- Express ideas through visualization
- Use basic plotting software to create graphics
- Use common language for describing a plot and its components

## Where do we get this material?

- Cleveland, *The Elements of Graphing Data*
- Cleveland, *Visualizing Data*
- Wainer, Dirty Dozen, TAS
- Tufte, *The Visual Display of Quantitative Information*
- Wainer, *Visual Revelations: Graphical Tales of Fate and Deception From Napoleon Bonaparte To Ross Perot*
- Robbins, *Creating More Effective Graphs*

# General Terminology

- Scale
  - line and label
  - tick mark and tick mark label
- Data: label, plotting symbol
- Key or legend
- Marker or reference: line or point, and label
- Title, subtitle, caption
- Juxtapose vs superpose vs inlay

# What are we looking for?

- Clear Vision
  - Make data stand out
  - Eliminate chart junk
  - Avoid clutter in data region
- Clear Understanding
  - Put major conclusion in graphical form
  - Provide reference information
  - Proof read for clarity and consistency

- Scale
  - Include or nearly include all data
  - Fill data region
  - Origin need not be on the scale
  - Choose a scale that improves resolution, e.g. percent change, log, ...
- General Strategy
  - Iterative process
  - Multiplicity is OK
  - Data rich

## Basic graphics model - Painter Model

- Start with a blank canvas
- High-level plotting function wipes the canvas clean and then "paints" a complete plot on the canvas
- Low-level functions can add to what is on the canvas, and will obscure what is below it
- Multitude of parameters available to the user to control details
- The canvas can be split up into multiple plotting regions, and the painter's model holds on each sub-region

## High-level plotting functions

- Show them the most common ones:

barplot, boxplot, curve, hist, plot, dotchart, image, matplot, mosaicplot, stripchart, contour

- plot is a generic plotting function - plot(x) and something useful should happen
  - x is a rpart object - dendrogram
  - x is a dataframe - pairs of scatterplots
  - x is an lm object - a series of plots
  - for more see `methods(plot)`

## Low-level functions

Low-level functions can be used to augment the plot.

- Add to the plotting region: abline, lines, segments, points, polygon, grid
- Add text: legend, text, mtext
- Modify/add axes: axis, box, rug

# Parameters

- The high and low -level functions take parameters that allow us to change the appearance of parts of the plot.
- Most high level functions contain arguments to modify the axis labels (**xlab**, **ylab**), plot title (**main**, **sub**), and scale of the axis, e.g. **xlim**, **ylim**.
- It is possible to specify some general parameters (i.e. **par()** params) in the high-level functions, e.g. **las**, **type**, **pch**, **mgp**

## Code to produce Revised voters plot

```
# prep work for colors and labels
party = c(dem = "Democrat", rep = "Republican",
          other = "Other", decline = "Decline to state")
partyColor = c(dem = "blue", rep = "red",
               other = "green", decline = "black")

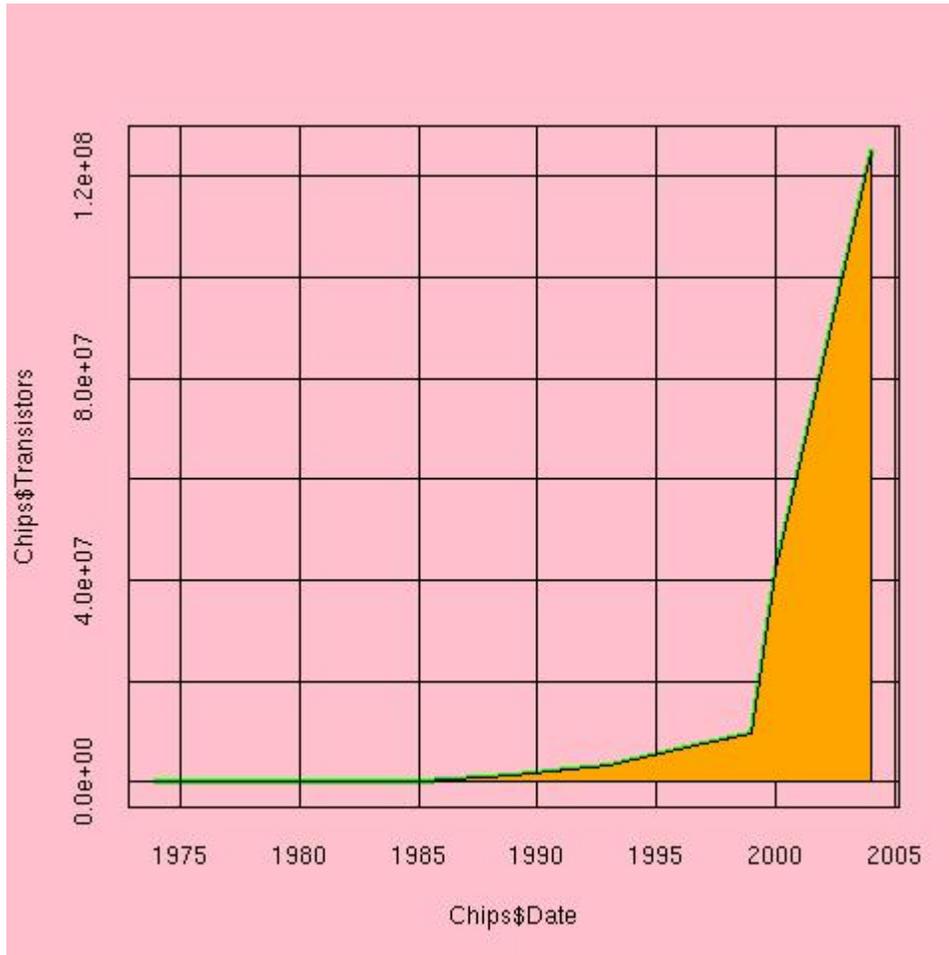
# There are many parameters that help you tailor the plot
matplot(voters$year, 100*voters[,4:7],
        type="l", xaxt = "n", lty=1,
        col= partyColor[names(voters)[4:7]],
        xlab="Year", ylab = "Percent",
        main = "Party Affiliation of Registered Voters in California")
```

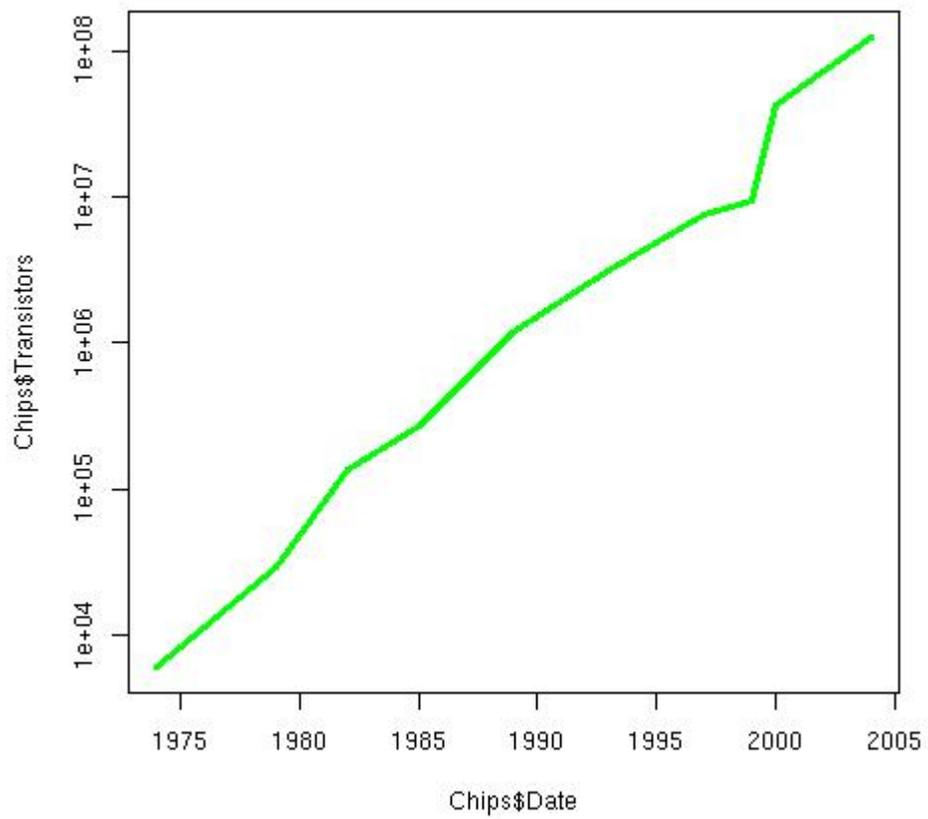
```
# There are also additional functions to get at greater detail
axis(1, at=seq(1992,2008,4))
legend(1992, 30, legend=party,
      fill=partyColor[names(party)])
points(voters$year, 100*voters[, "dem"],
      col=partyColor["dem"], pch = 19, cex=1.5)
```

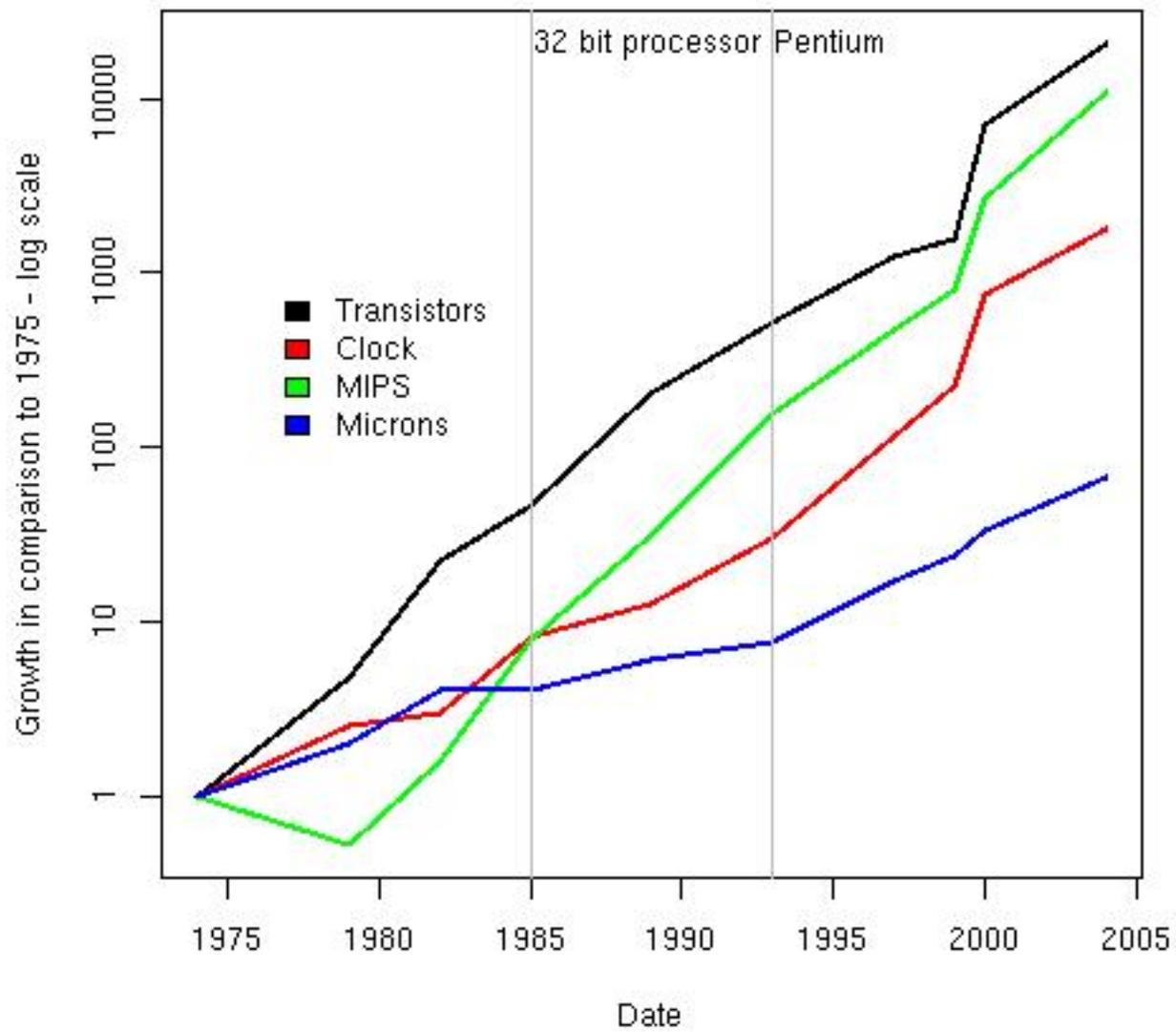
## par()

- Permanent: change parameters permanently
- Restore: **par** returns the par values before it was called, and these can be saved and restored
- Default: Calling **par()** allows your to reset to the default parameter values
- there are a multitude of graphical parameters that can be set
- Scope: parameter values set in high-level functions can have different effect than those set in **par()** and different from other high-level functions, e.g. **par(col="red")** and **plot(x, col="red")**.

## Iteration - from horrid to decent







## Color dimensions

**Hue** typically associated with color names, e.g. red, green, blue, yellow. It connects to the dominant wavelength: long-wavelength reds to short-wavelength blues; from red to orange to yellow to green to blue; purples result from mixing opposite ends of the spectrum together.

**Lightness** is a relative measure that describes how much light appears to reflect from an object compared to what looks like white in the scene.

**Saturation** measures the vividness of a color

Computer screens use a reduced set of light primaries for mixing all the other hues: red, green, and blue (RGB).

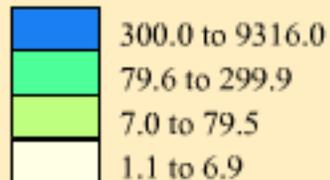
**Transparency** - semi-transparent color for quartz and pdf devices

In R, color can be specified using:

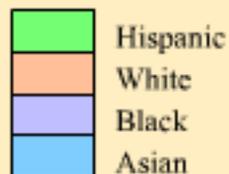
- an RGB hex triple;
- a name, e.g. "red";
- color generating function, e.g. `gray()`, `rgb()` ...
- functions that generate a coherent set of colors: `heat.colors`, `rainbow`, `colorRamp`, `terrain.colors`;
- the **RColorBrewer** and `colorspaces` packages.

## Learn More: *Legend Type*

People per sq. mile  
by state



Group with highest  
percent of  
state population



**1. Sequential schemes** are suited to ordered data that progress from low to high. Lightness steps dominate the look of these schemes, with light colors for low data values to dark colors for high data values.

**2. Diverging schemes** put equal emphasis on mid-range critical values and extremes at both ends of the data range. The critical class or break in the middle of the legend is emphasized with light colors and low and high extremes are emphasized with dark colors that have contrasting hues.

**3. Qualitative schemes** do not imply magnitude differences between legend classes, and hues are used to create the primary visual differences between classes. Qualitative schemes are best suited to representing nominal or categorical data.

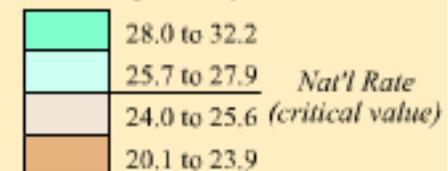
These scheme types grow from the teaching of Dr. Judy Olson.

For more information:

Brewer, Cynthia A. 1994. Color use guidelines for mapping and visualization. Chapter 7 (pp. 123-147) in *Visualization in Modern Cartography*, edited by A.M. MacEachren and D.R.F. Taylor, Elsevier Science, Tarrytown, NY ([Figures are online](#)).

[Other cartography publications](#) by Cynthia Brewer

Percent of total pop.  
under age 18 by state



close

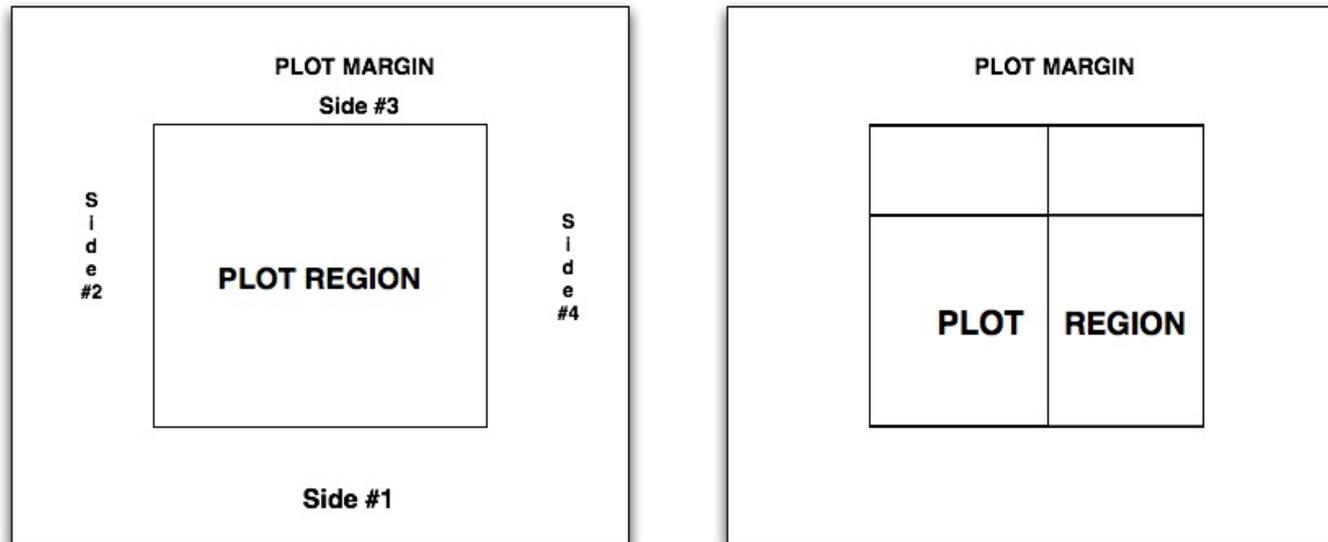
# Text

- **text()** Can place text anywhere in plotting region
- Control: font family, size, face, and color can be specified
- Margins: **mtext**
- Orientation:
- Math: **expression()**

```
main=expression(paste(italic("Poisson"), "(" , lambda == 1, ") ")
```

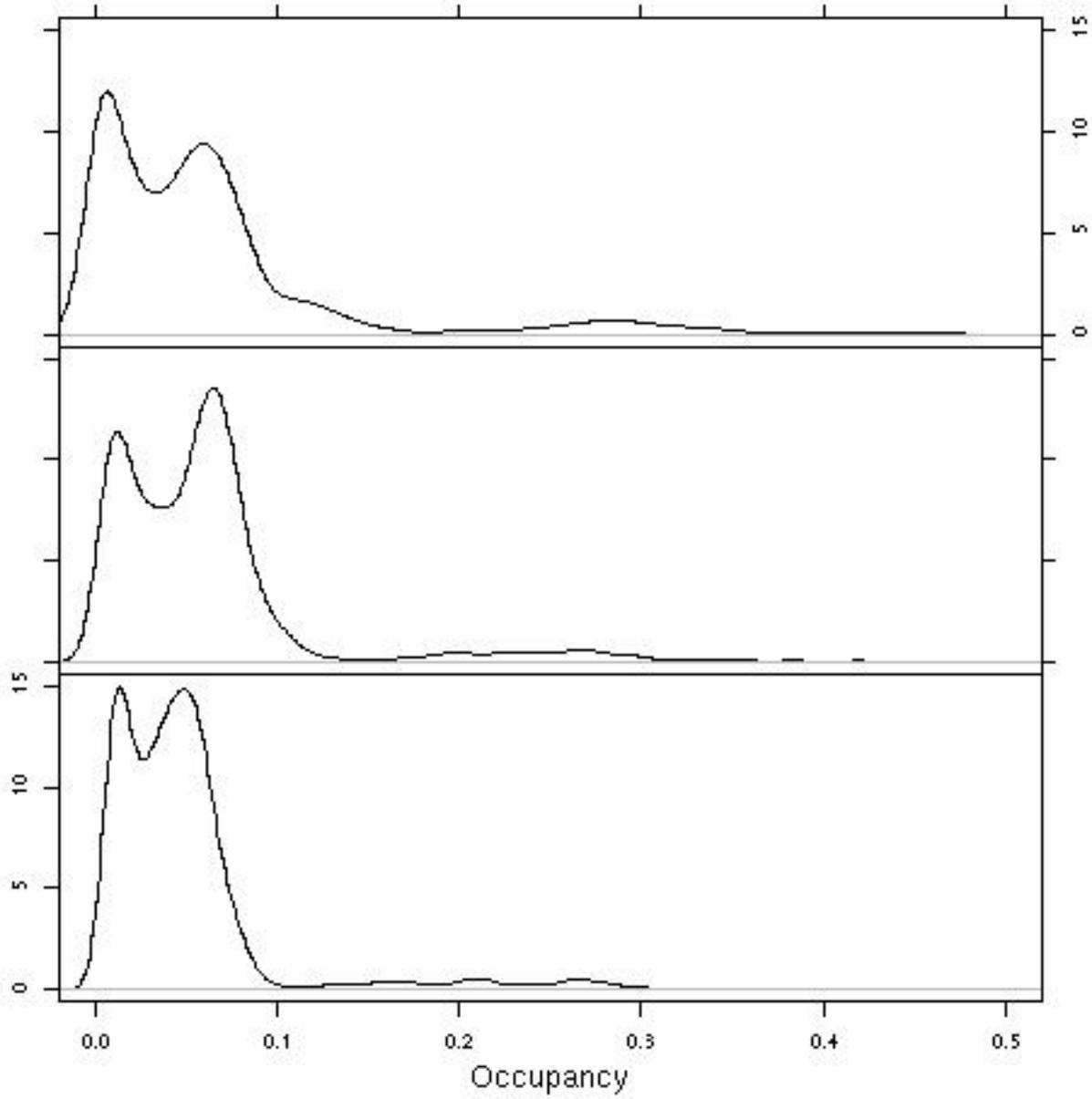
*Poisson*( $\lambda = 1$ )

# Arranging Plots on the canvas



- Multiple plots can appear in one plot region
- **mfrow** parameter cuts the region into a grid
- **layout()** provides a grid with possibly unequal heights and width, and can place a plot in a rectangular region that covers more than one grid section

Loop Detector Occupancy for Left (bottom), Middle, and Right (top) lanes



## Steps in drawing a plot:

- **plot.new()** set up plot region
- **plot.window()** set up coordinate system (**xlim**, **ylim**)
- **box()** Draw rectangle around plot
- **axis** Draw axes, **lines()**, **points()**, **text**, ...
- **usr** provides/sets the coordinate system

```

par(mfrow=c(3,1), mar=rep(0,4), oma = c(4,4,3,3))
plot(density(rtraff$Occ[rtraff$lane == "Slow"]),
      ylim=c(0,15), xlim=c(0, 0.5), main="", axes = FALSE)
mtext("Loop Detector Occupancy for Left (bottom),
      Middle, and Right (top) lanes", side = 3, line = 2)
box()
axis(2, labels=FALSE); axis(3, labels=FALSE); axis(4)
plot(density(rtraff$Occ[rtraff$lane == "Middle"]),
      ylim=c(0,15), xlim=c(0, 0.5), main="", axes = FALSE)
box(); axis(2, labels=FALSE); axis(4, labels=FALSE)
plot(density(rtraff$Occ[rtraff$lane == "Passing"]),
      ylim=c(0,15), xlim=c(0, 0.5), main="", axes = FALSE)
box(); axis(1); axis(2)
mtext("Occupancy", side = 1, line =2.5)

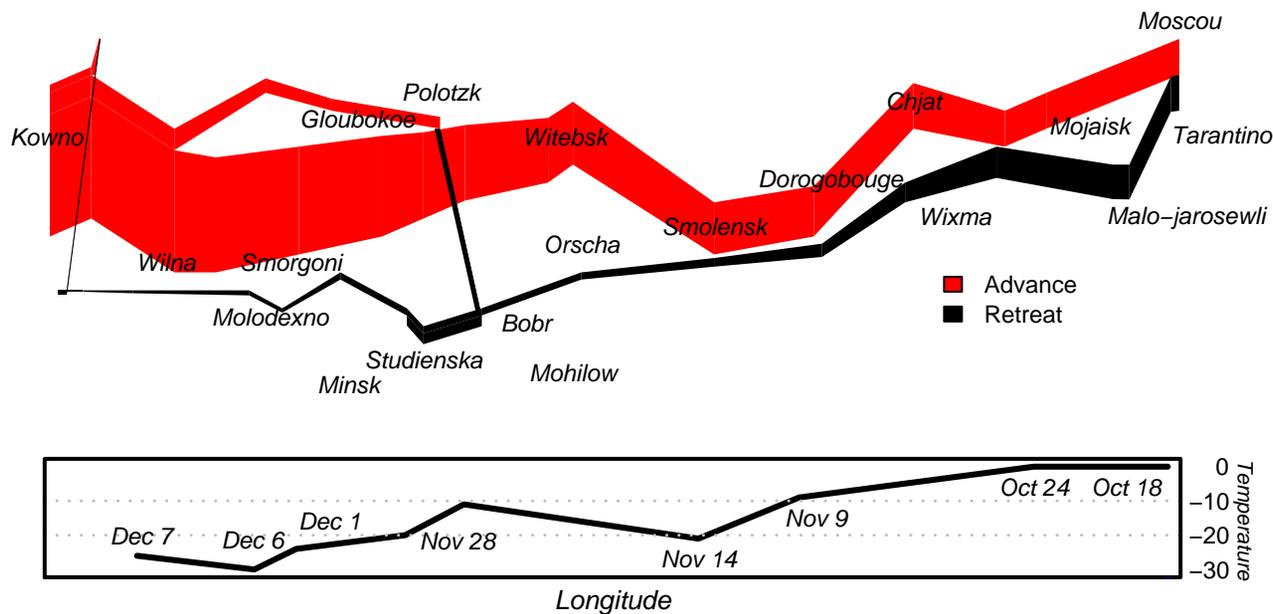
```



# Grammar of Graphics - Wilkinson

- **DATA** functions needed to create variables from data.
- **TRANS** Transformations, if any, to be applied to the variables
- **FRAME** The graphic frame describes the context of the plot. Use algebraic expression:
  - One-dimensional frame is typically specified by a single variable, e.g.  $x$ .
  - Two dimensional frame by  $x*y$ . Also,  $x * (y + z)$  superposes the units and ranges of  $y$  and  $z$  on the vertical axis.
- **SCALE** Dimensions on which the graphics orient themselves, e.g. categorical, interval, log, and power.
- **COORD** Coordinate system to use, such as polar and cartesian. Plus, information about how to reflect, rotate, stretch, dilate, and translate the coordinates.

- **GUIDE** Details of guiding notation such as axes, legends, markers, etc.
- **GRAPH** Functions to appear in the frame. Two simple examples are the *point* function and the *line* function.



```
mat = matrix(data=c(1,2),nrow = 2)
layout(mat, heights = c(3,1) )
```

# Learning Objectives

- Apprenticeship: Find out about and use a variety of parameters
- Compose multiple plots on one canvas
- Work from grammar of graphics to plot functions
- Raise the bar: they can create amazing plots

# Graphics Devices

Multiple devices

Print devices - **jpeg()**, **pdf()**, **png()**

Save current device

```
devSave = dev.cur()  
devSave
```

Open a new device

```
quartz()  
myDev = dev.cur()
```

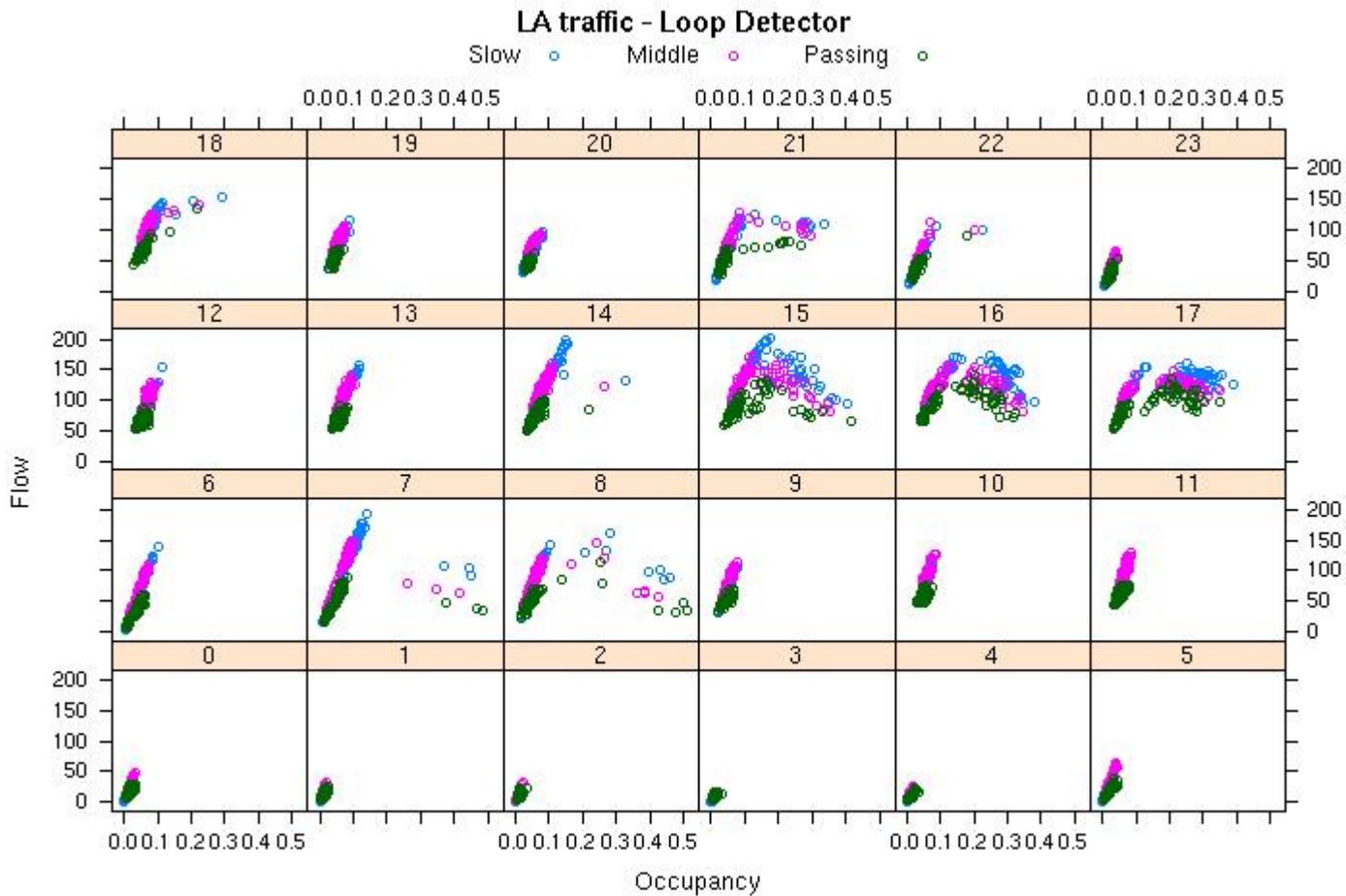
Plot and then reset to previous device

```
plot(x)  
dev.set(devSave)
```

Turn off the myDev device if no longer needed **dev.off()**

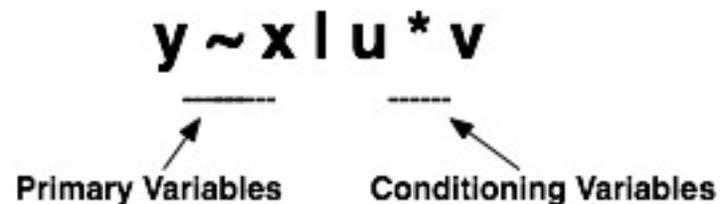
## Trellis - Lattice

- Rectangular array of plots (panels)
- Multi-panel conditioning - cross-tabs plots
- Coordinate scale, aspect ratio, labels across plots
- Make efficient use of display area
- Default display as useful as possible (clear vision)
- Abstraction: specify a plot through type of graphic and role of variables
- Parallel to base graphics:
  - High-level functions produce complete graphic
  - Low-level functions give tools to augment plot
  - User-modifiable parameters control details



```
xyplot(Flow ~ Occ | factor(hour), rtraffic, groups=lane,
       auto.key= list(columns = 3), layout = c(6,4),
       xlab="Occupancy", main="LA traffic - Loop Detector")
```

## Formula



- The formula specifies the variables involved in the plotting
- Primary variables appear in the plot region
  - $\sim x$  for univariate
  - $y \sim x$  for plot of  $y$  on  $x$
  - $y \sim z \sim x$  plots both  $y \sim x$  and  $z \sim x$  on same region
- One panel for each unique value/level of conditioning variable(s)
- Conditioning variables optional
  - histogram( $\sim x$ )** is like **hist( $x$ )**
  - xyplot( $y \sim x$ )** is like **plot( $x, y$ )**

- Conditioning variables appear in the strips along the top of a panel
- Conditioning variables are factors or shingles (may use **cut()**, or **equal.count()** )
- High-level lattice functions are generic, all have a “formula” method
- With **data** parameter, do not need to specify data frame in formula (also allows other types of input such as lists and environments)
- **groups** specifies a grouping variable to use in superposing plots

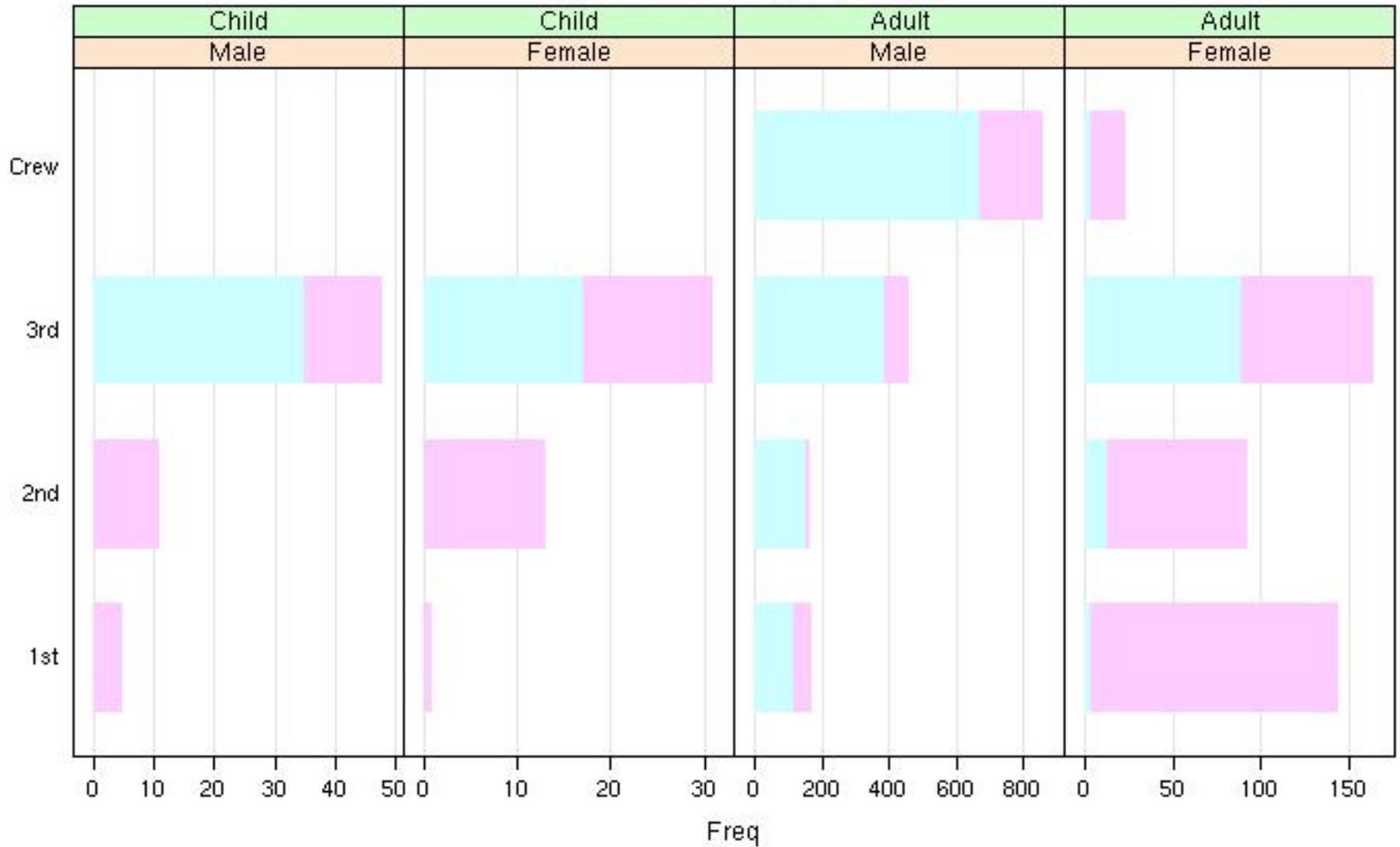
# The Panel

- Unique combination of levels of conditional variables results in a *packet of data*
- *One packet = one panel*
- *Aspect ratio can be specified, **aspect**="xy" gives Cleveland's banking rule*
- *Layout of panels can be customized*

```
trl = barchart(Class ~ Freq | Sex + Age,  
              as.data.frame(Titanic),  
              groups=Survived, stack=TRUE, layout = c(4,1),  
              auto.key = list(title="Survived", columns=2),  
              scales = list(x="free"), border = "transparent")  
update(trl, panel= function(...) {  
  panel.grid(h=0, v=-1);  panel.barchart(...)}))
```

# Survived

No █ Yes █



# Parameters

- Layout
  - **layout** = `c( #cols, #rows, #pages - optional)`
  - `(0, #)` means that this is a lower bound on the number of panels per page
- Scale
  - **scales** = `"free"`
  - Provide a list to control tick marks, position of labels, free x or y scale
- **between** adds spacing between panels
- **auto.key** provide a list of specifications for legend
- Titles can be specified via **main**, **sub**, **xlab**, **ylab**, **page**
- Many of the parameters from base graphics carry over, e.g. **pch**, **type**

# Panel Function

- Each high-level plot has a panel function responsible for plotting the packet
- The high-level function has `...` argument to accept extra arguments, and those it does not recognize will be passed to the panel function
- **panel** argument in the high-level function takes a panel function
- Default panel function **panel.functionName**, e.g. **panel.barchart**
- Default panel function has a `...` argument, which makes it handy to augment the panel function by overriding it, adding your extra stuff, and invoke the default with `....`.

**panel** = **myPanel(...)** { my code ; **panel.defaultFunction(...)** }

- There are three sorts of functions you can use in a panel function:
  - lattice panel functions, e.g. **panel.grid()**
  - Low-level plotting functions, e.g. **llines()**
  - Grid functions - the lowest level

## Trellis object

- Return value is a trellis object
- The object is a complete description of the plot
- The object can be printed/plotted **print(trl)** or **plot()**
- The object can be updated **update()**

# Grid

- Page: no plot region or margins, but page/canvas
- All functions are low-level, e.g. `grid.newpage()`, `grid.rect()`, `grid.curve()`, `grid.lines()`, `grid.text()`
- Build up a plot piecewise using these functions
- Parameter: `gpar()` creates a theme or context, and `gp` argument applies the context to output
- Coordinate systems: plotting can be relative to different coordinate systems: mm, npc, points, char, lines, ...

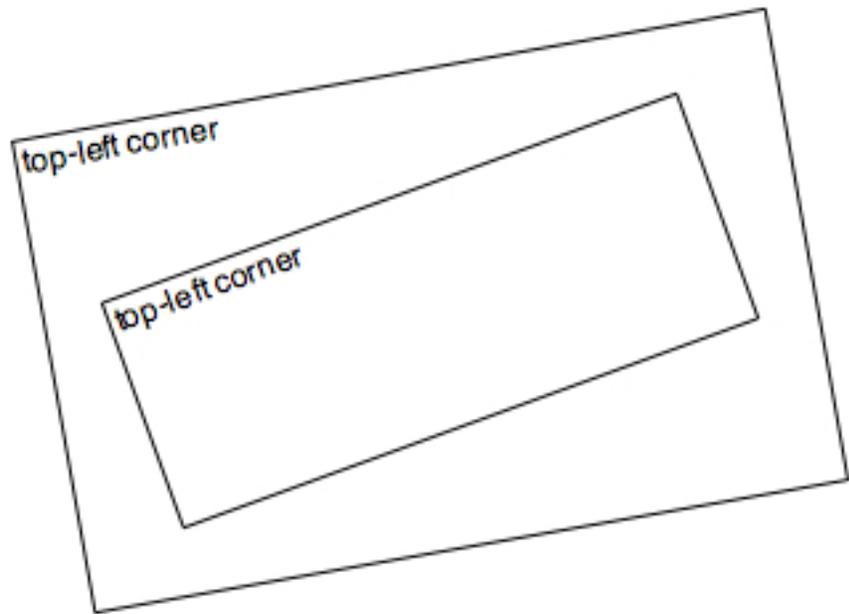
# Viewports

- Drawing is relative to the current viewport
- Viewport is a whole "page"
- **grid.layout()** splits up the viewport as with other layout functions, except viewports can be defined for particular rows and columns in the layout
- Graphical parameter contexts can be set for a viewport using the **vp** argument
- New viewports can be created and "pushed" to be the current viewport
- Navigate viewpoints with **popViewport()**, **upViewport()**, **downViewport()**
- **grid.ls()** provides a list of all viewports and **current.viewport()** gives current one

```

pushViewport (viewport (width=0.8, height=0.5,
                        angle=10, name="vp1"))
grid.rect ()
grid.text ("top-left corner", x=unit (1, "mm"),
          y=unit (1, "npc") - unit (1, "mm"), just = c ("left", "top"))
pushViewport (viewport (width=0.8, height=0.5,
                        angle = 10, name = "vp2"))
grid.rect ()
grid.text ("top-left corner", x=unit (1, "mm"),
          y=unit (1, "npc") - unit (1, "mm"), just = c ("left", "top"))

```



```

#Set up viewports in a stack
pvp = plotViewport(c(2,4,1,1), name="plotvp")
dvp = dataViewport(rtraffic$Occ, rtraffic$Flow, name="datavp")
pushViewport(vpStack(pvp, dvp))
upViewport(0)
# Draw rectangle in plotvp
grid.rect(vp= "plotvp")
# Draw points in datavp
grid.points(rtraffic$Occ, rtraffic$Flow, size = unit(2, "mm"))
      gp = gpar(col = c("red", "blue", "green")[rtraffic$lane],
      vp=vpPath("plotvp", "datavp"))
# Add the other pieces, text, axes, tickmarks, ...
grid.text("Flow", x = unit(-3, "lines"), rot=90, vp = "plotvp")
grid.text("Occupancy", y = unit(-1, "lines"), vp = "plotvp")

```

