Graphics

In the R language

Derived from Peng's and Nolan's Notes

Base Graphics

Base graphics are used most commonly and are a very powerful system for creating 2-D graphics.

- Calling `plot(x, y)` or `hist(x)` will launch a graphics device (if one is not already open) and draw the plot on the device
- If the arguments to `plot` are not of some special class, then the *default method* for `plot` is called; this function has *many* arguments, letting you set the title, x axis and y axis labels, x and y axis limits, etc.
- The base graphics system has *many* parameters that can set and tweaked; these parameters are documented in `?par`

A first assignment: 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?

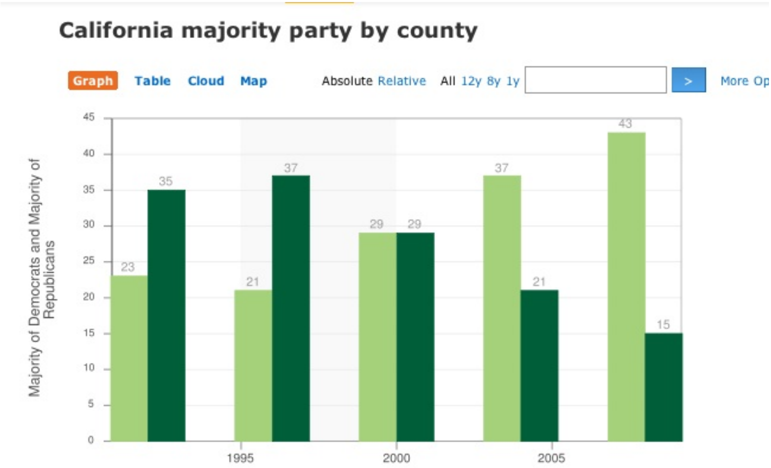
A second assignment: Deconstruct-Reconstruct a Plot

- Find a plot on swivel.com that you can improve
- Critique the plot
- Find the message, and create a new plot that better communicates that message.
- Improve the plot even more by for example adding auxiliary information.

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

An example for the students



California majority party by county

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
```

What's the message?
Can you improve upon it?

Critique

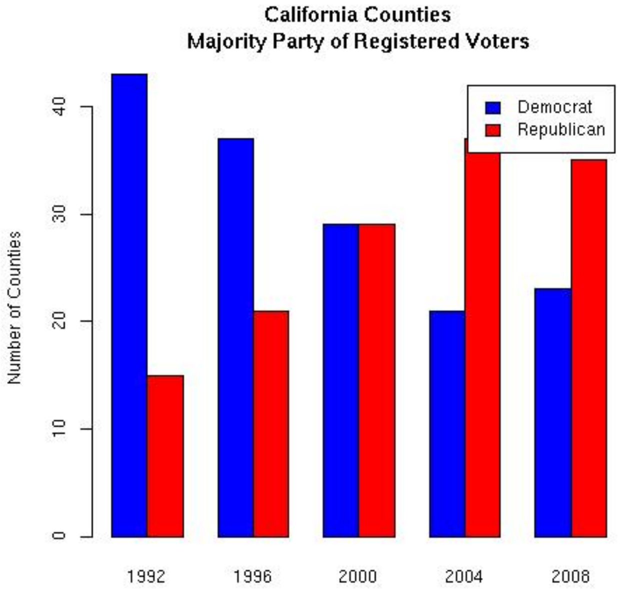
Basic plotting issues

- 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

Revision

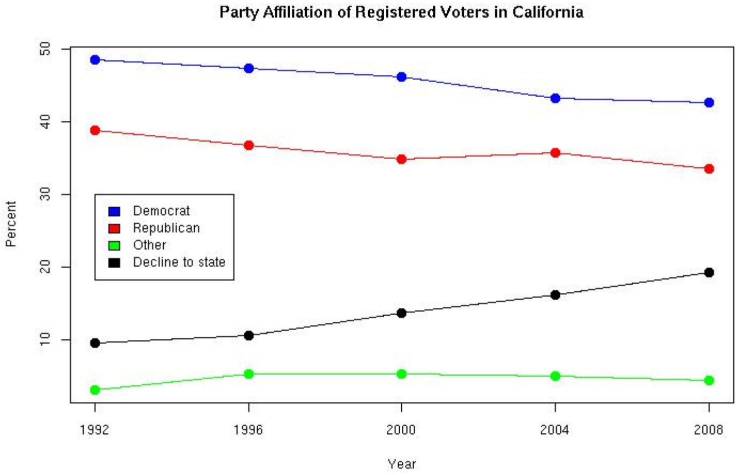

California Counties
Majority Party of Registered Voters
Number of Counties
0
10
20
30
40
1992
1996
2000
2004
2008
🟦 Democrat
🟥 Republican

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
```

Alternative Figure



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*
- Murrell, *R Graphics*
- Murrell, *Class Notes - some of these are excerpted from his notes*

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 - dendogram
  - 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

Some Important Base Plotting Functions

- `plot`: make a scatterplot, or other type of plot depending on the class of the object being plotted
- `lines`: add lines to a plot, given a vector x values and a corresponding vector of y values (or a 2-column matrix); this function just connects the dots
- `points`: add points to a plot
- `text`: add text labels to a plot using specified x, y coordinates
- `title`: add annotations to x, y axis labels, title, subtitle, outer margin
- `mtext`: add arbitrary text to the margins (inner or outer) of the plot
- `axis`: adding axis ticks/labels

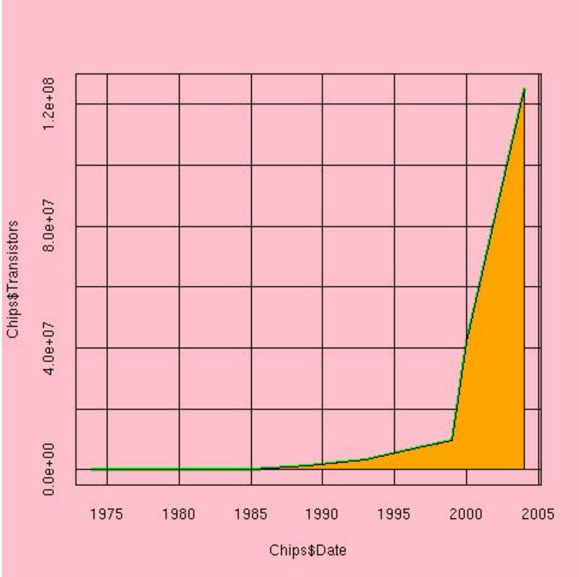
Parameters

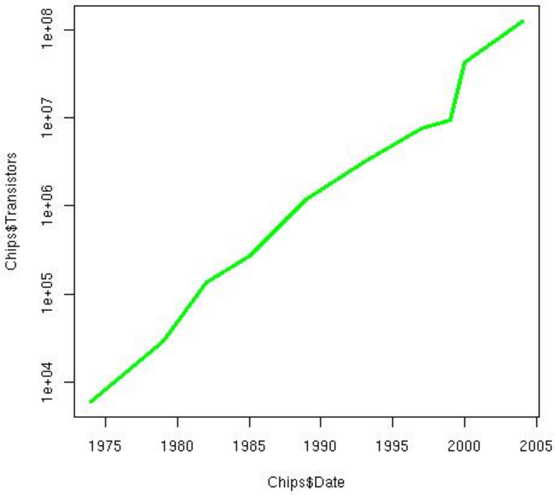
- The high and low -level functions take parameters that allow us to change the appearance of parts of the plot.
- Most high level functions contain arguments to modify the axis labels (`xlab, ylab`), plot title (`main, sub`), and scale of the axis, e.g. `xlim, ylim`.
- It is possible to specify some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

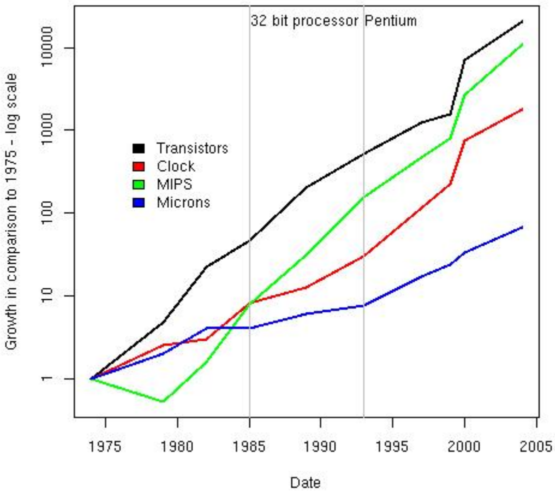
par()

- 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 horrible to decent




Chips$Transistors
1e+08
1e+07
1e+06
1e+05
1e+04
1975
1980
1985
1990
1995
2000
2005
Chips$Date


32 bit processor
Pentium
■ Transistors
🟥 Clock
🟩 MIPS
■ Microns
10000
1000
100
10
1
Growth in comparison to 1975 - log scale
1975
1980
1985
1990
1995
2000
2005
Date

Some Important Base Graphics Parameters

The `par` function is used to specify global graphics parameters that affect all plots in an R session. These parameters can often be overridden as arguments to specific plotting functions.

- pch: the plotting symbol (default is open circle)
- lty: the line type (default is solid line), can be dashed, dotted, etc.
- lwd: the line width, specified as an integer multiple
- col: the plotting color, specified as a number, string, or hex code; the `colors` function gives you a vector of colors by name
- las: the orientation of the axis labels on the plot

Some Important Base Graphics Parameters

- bg: the background color
- mar: the margin size
- oma: the outer margin size (default is 0 for all sides)
- mfrow: number of plots per row, column (plots are filled row-wise)
- mfcol: number of plots per row, column (plots are filled column-wise)

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.

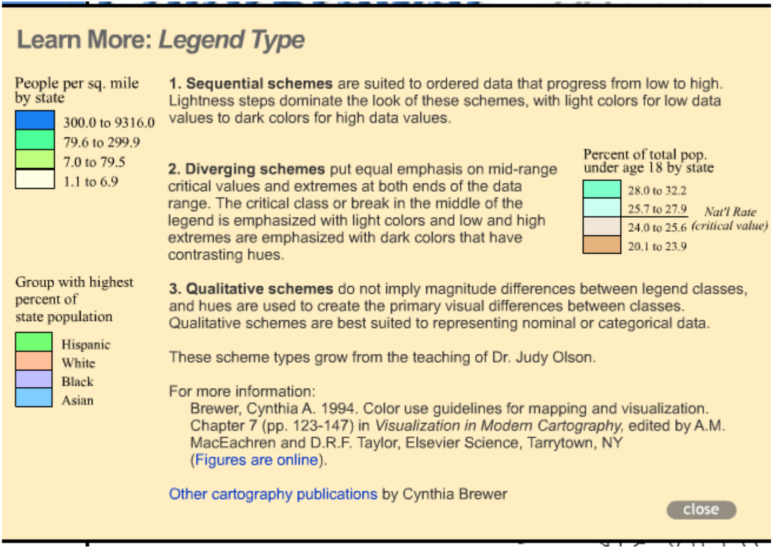
Saturation measures the vividness of a color

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, terraine.colors;
- the `RColorBrewer` and colorspaces packages.



Learn More: *Legend Type*

People per sq. mile by state

🟦	300.0 to 9316.0
🟩	79.6 to 299.9
🟩	7.0 to 79.5
⬜	1.1 to 6.9

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.

Percent of total pop. under age 18 by state

🟩	28.0 to 32.2
🟩	25.7 to 27.9
⬜	24.0 to 25.6
🟧	20.1 to 23.9

Nat'l Rate (critical value)

Group with highest percent of state population

🟩	Hispanic
🟧	White
🟪	Black
🟦	Asian

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).

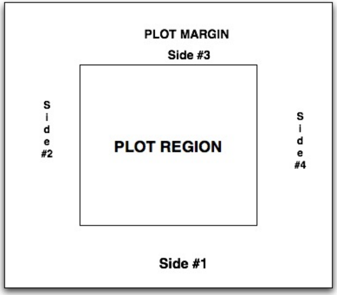
Other cartography publications by Cynthia Brewer

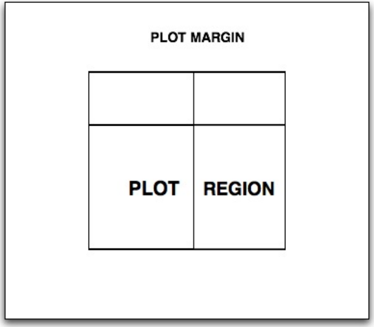
Text

- `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,")"))
```

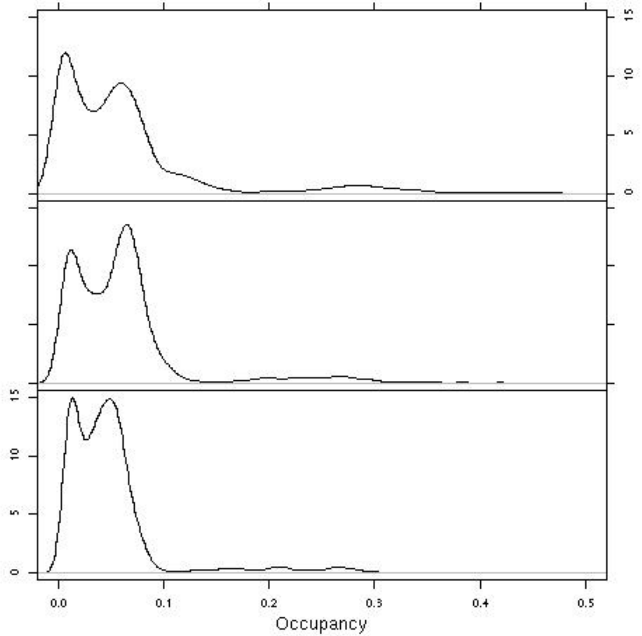
Arranging Plots on the canvas





- 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:

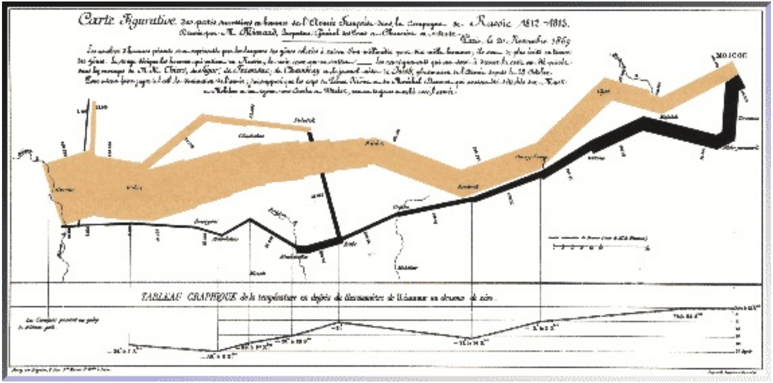
- `plot.new()` set up plot region
- `plot.window()` set up coordinate system (`xlim, ylim`)
- `box()` Draw rectangle around plot
- `axis` Draw axes, `lines()`, `points()`, `text`, ...
- `usr` provides/sets the coordinate system

Example

```
#set up canvas with 3 plot regions
par(mfrow=c(3,1), mar=rep(0,4), oma = c(4,4,3,3))

# make the top plot
plot(density(rtraffic$Occ[rtraffic$lane =="Slow"]),
     ylim=c(0,15), xlim=c(0, 0.5), main="", axes = FALSE)

#add text to top margin
mtext("Loop Detector Occupancy for Left (bottom),
     Middle, and Right (top) lanes", side = 3, line = 2)

# Add box around first plot region
box()

#Add axes to the right side
axis(2, labels=FALSE); axis(3, labels=FALSE); axis(4)
```

```
# Make  middle plot
plot(density(rtraffic$Occ[rtraffic$lane =="Middle"]),
    ylim=c(0,15), xlim=c(0, 0.5), main="", axes = FALSE)

# Draw box around it - no axes ticks or labels
box(); axis(2, labels=FALSE); axis(4,labels=FALSE)

# Draw bottom plot
plot(density(rtraffic$Occ[rtraffic$lane =="Passing"]),
  ylim=c(0,15), xlim=c(0, 0.5), main="", axes = FALSE)

# add box, then x and y axes
box();axis(1);axis(2)

# add text for a label on the y axes
mtext("Occupancy", side = 1, line =2.5)
```

Advanced Assignment: Replicate Napoleon's March


Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813.
Dressée par M. Minard, Inspecteur Général des Ponts et Chaussées en retraite.
Paris, le 20 Novembre 1869
MOSCOU
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.

Learning Objectives

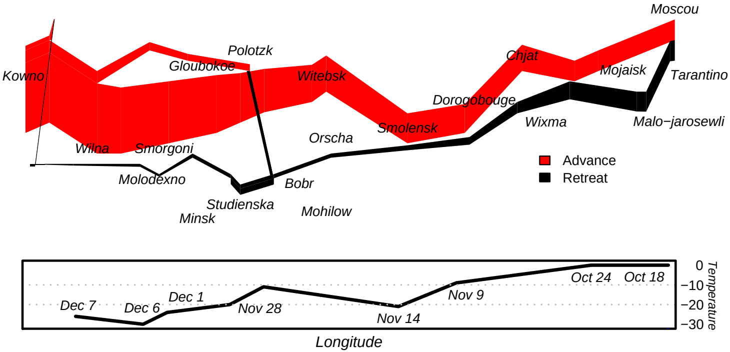
- Apprenticeship: Find out about and use a variety of parameters
- Compose multiple plots on one canvas
- Work from grammar of graphics to plot functions
- Raise the bar: they can create amazing plots

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.

One student's plot



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

Copying Plots

There are two basic approaches to plotting.

1. Launch a graphics device
2. Make a plot; annotate if needed
3. Close graphics device

Or

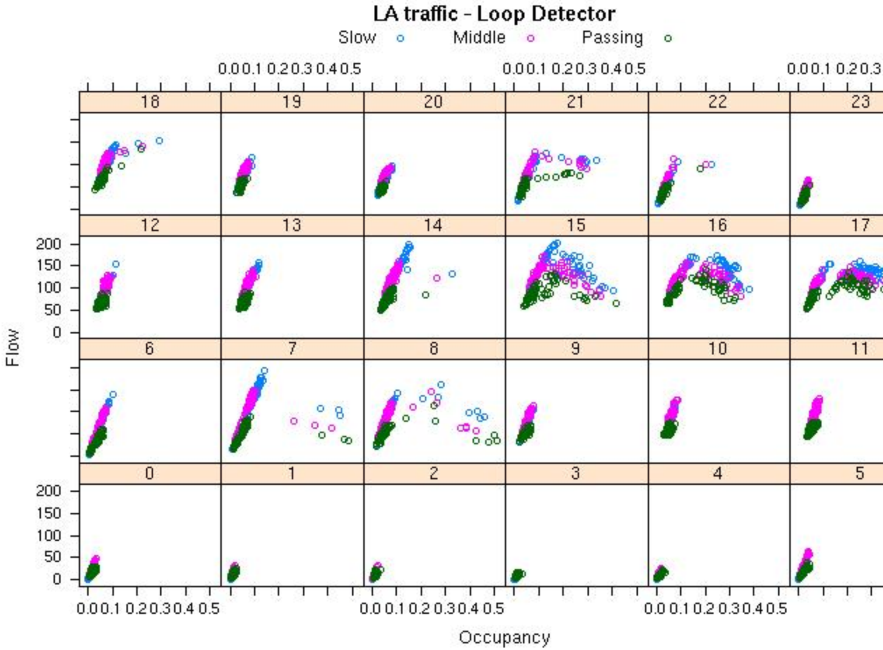
1. Make a plot on a screen device (default); annotate if needed
2. Copy the plot to another device if necessary (not an exact process)

Working with Graphics Devices

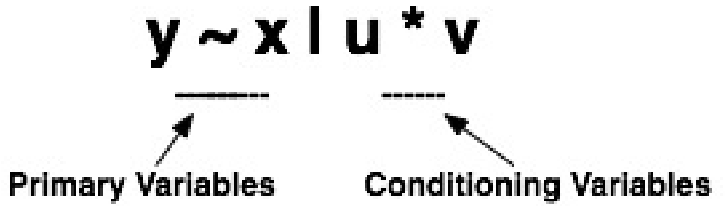
- Print devices - `jpeg()`, `pdf()`, `png()`
- Copy a plot from one device to another - `dev.copy`
- Save current device - `devSave = dev.cur()`
- Open a new device - `quartz()`
  Plot to current device `plot(x)` and then reset to previous device `dev.set(devSave)`
- Turn off the myDev device if no longer needed
  `dev.off(myDev)`

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


LA traffic - Loop Detector
Slow ○
Middle ○
Passing ○
Flow
Occupancy
0.0 0.1 0.2 0.3 0.4 0.5
0 50 100 150 200
18
19
20
21
22
23
12
13
14
15
16
17
6
7
8
9
10
11
0
1
2
3
4
5

Formula



y ~ x | u * v

Primary Variables

Conditioning Variables

- The formula specifies the variables involved in the plotting
- Primary variables appear in the plot region

  ˜ x for univariate

  y ˜ x for plot of y on x

  y z x+ plots both y ˜x and z˜x on same region
- One panel for each unique value/level of conditioning variable(s)
- Conditioning variables optional

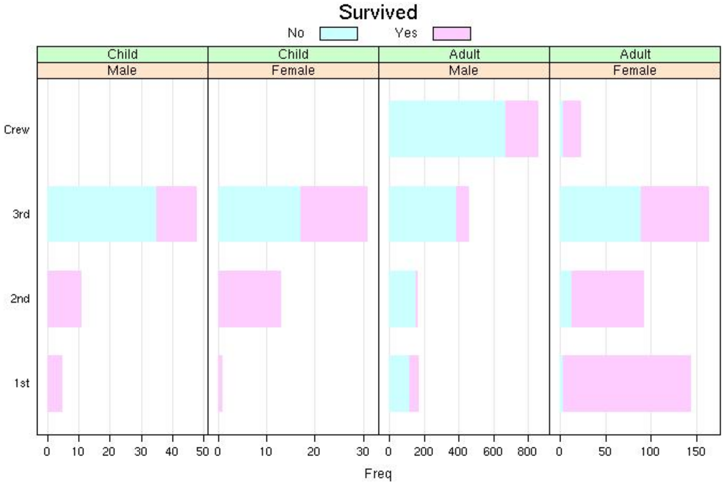
  `histogram(˜x)` is like `hist(x)`

  `xyplot(y˜x)` is like `plot(x,y)`

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
Child
Male
Child
Female
Adult
Male
Adult
Female
Crew
3rd
2nd
1st
0
10
20
30
40
50
0
10
20
30
0
200
400
600
800
0
50
100
150
Freq

Lattice Functions

- `xyplot`: this is the main function for creating scatterplots
- `bwplot`: box-and-whiskers plots ("boxplots")
- `histogram`: histograms
- `stripplot`: like a boxplot but with actual points
- `dotplot`: plot dots on "violin strings"
- `splom`: scatterplot matrix; like `pairs` in base graphics system
- `levelplot`, `contourplot`: for plotting "image" data

Lattice Behavior

Lattice functions behave differently from base graphics functions in one critical way.

- Base graphics functions plot data directly to the graphics device
- Lattice graphics functions return an object of class `trellis`.
- The print methods for lattice functions actually do the work of plotting the data on the graphics device.
- Lattice functions return "plot objects" that can, in principle, be stored (but it's usually better to just save the code + data).
- On the command line, `trellis` objects are *auto-printed* so that it appears the function is plotting the data

Lattice Panel Functions

Lattice functions have a `panel` function which controls what happens inside each panel of the entire plot.

```
x <- rnorm(100)
y <- x + rnorm(100, sd = 0.5)
f <- gl(2, 50, labels = c("Group 1", "Group 2"))
xyplot(y ~ x | f)
```

plots y vs. x conditioned on f.

Lattice Panel Functions

```
xyplot(y ~ x | f,
       panel = function(x, y, ...) {
               panel.xyplot(x, y, ...)
               panel.abline(h = median(y),
                            lty = 2)
       })
```

plots y vs. x conditioned on f with horizontal (dashed) line drawn at the median of y for each panel.

Lattice Panel Functions

Adding a regression line

```
xyplot(y ~ x | f,
       panel = function(x, y, ...) {
               panel.xyplot(x, y, ...)
               panel.lmline(x, y, col = 2)
       })
```

fits and plots a simple linear regression line to each panel of the plot.

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

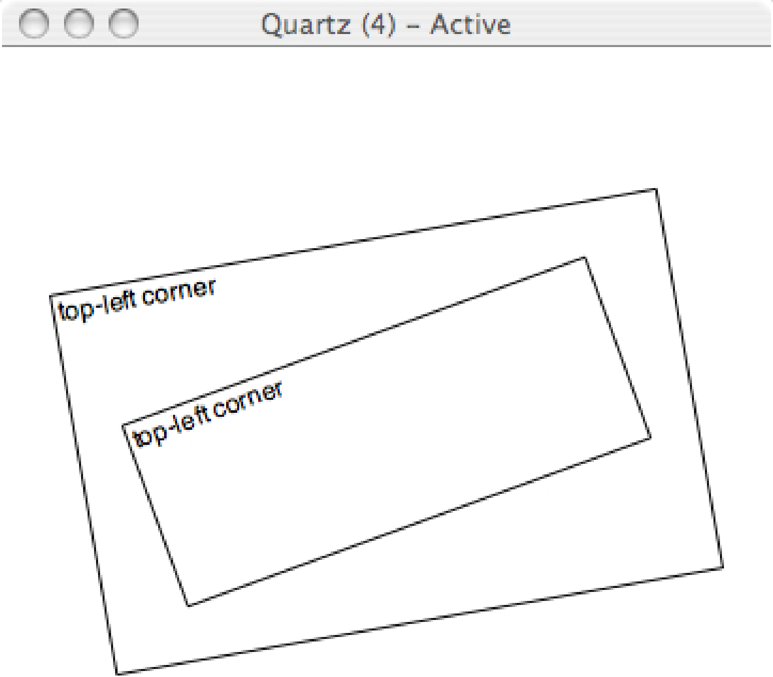
Grid

- 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, ...

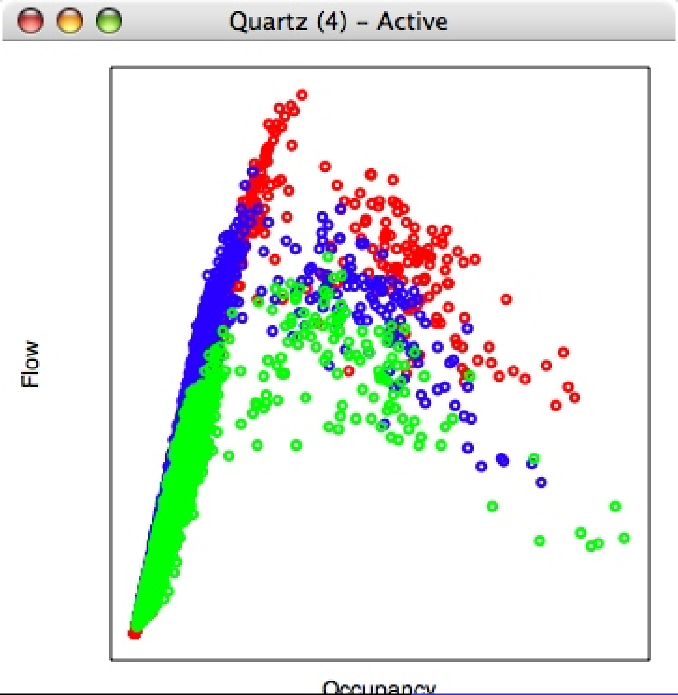
Viewports

- 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", "t
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
```


Quartz (4) – Active
top-left corner
top-left corner

```
#Set up viewports in a stack
pvp = plotViewport(c(2,4,1,1), name="plotvp")
dvp = dataViewport(rtraffic$Occ, rtraffic$Flow, name="datav
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 ="plotv
grid.text("Occupancy", y = unit(-1, "lines"), vp ="plotvp")
```


Quartz (4) – Active
Flow
Occupancy