Graphs and their usage

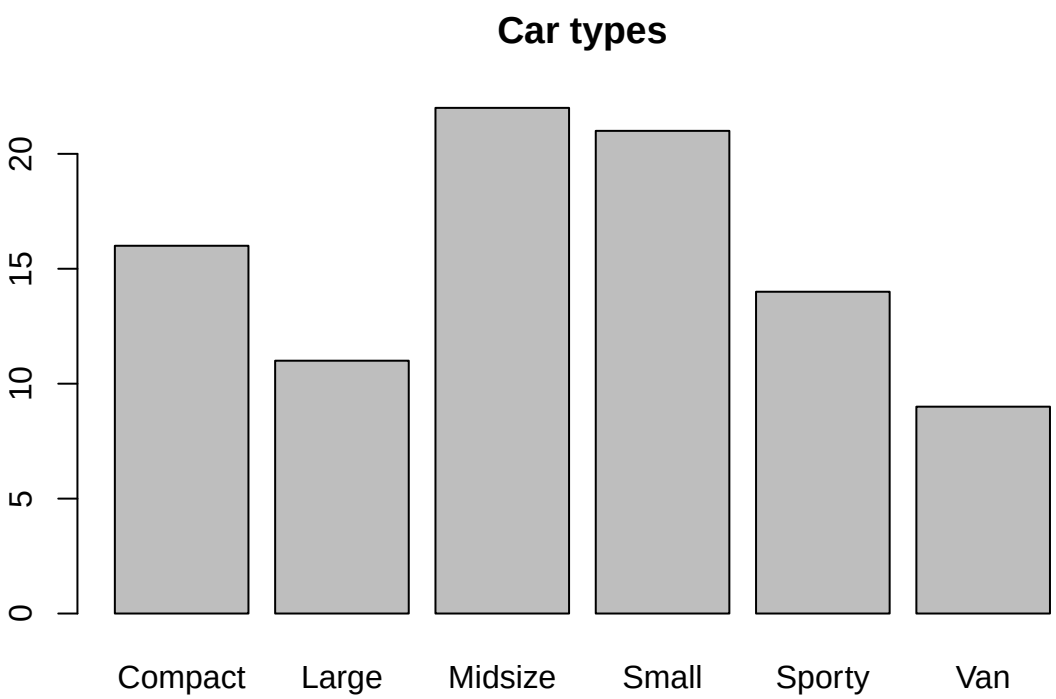
Through this semester we have looked at various data types and visual methods to extract from data information we would like to have. So far, we have looked at 4 types of data: 1-dimensional categorical, 2-dimensional categorical, 1-dimensional continuous, and 2-dimensional continuous.

1-dimensional categorical data

Most data sets have some categorical variables. We may be interested in the (related notions of) number of data points in each category or proportion of data points in our set that fall in each category. Many categorical variables are unordered (like 'red', 'green', 'blue') and some have some intrinsic order (like stages of a tumor size). For ordered categorical variables, we may be interested in a trend in the variables.

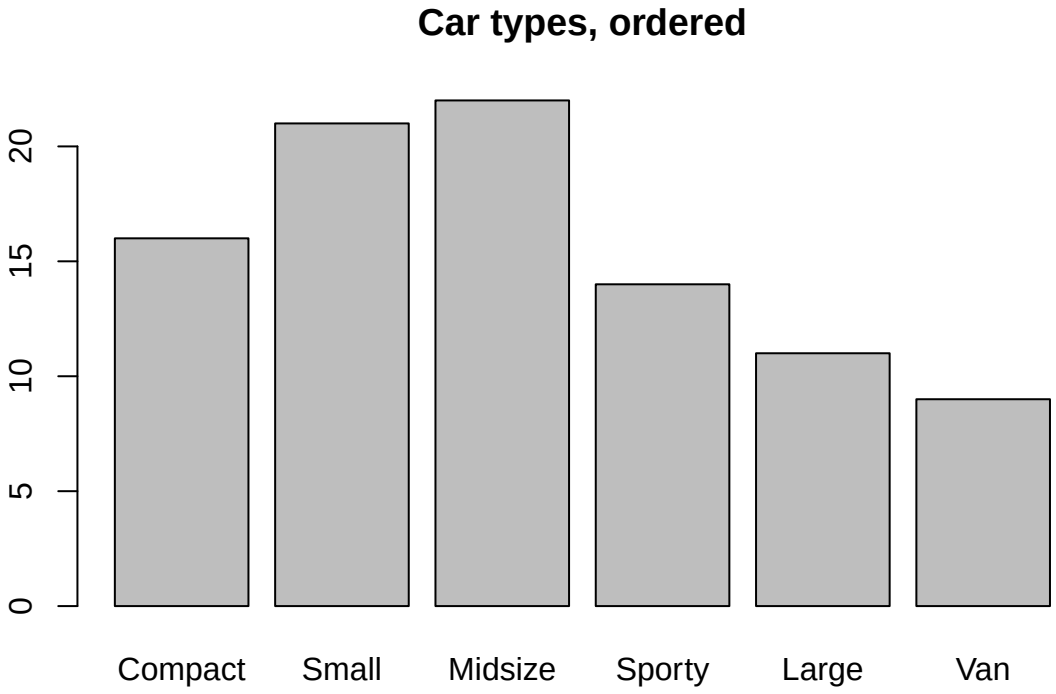
Barplot

```
library(MASS)
barplot(table(Cars93$Type), main="Car types")
```



R defaults to putting categories in alphabetical order. In this dataset, there is at least some notion of order, which is gas guzzlage. We can order our barplot thus

```
barplot(table(Cars93$Type)[c(1, 4, 3, 5, 2, 6)], main="Car types, ordered")
```



Now we can see something of a clear trend. There were more effecient cars models in this dataset, though the number of cars rose with from Compact to Midsize, after which there was a decline with cars associated with low fuel efficiency. It seems that, if this were representative of consumer choice in 1993, people wanted a fuel efficient, though spacious vehicle.

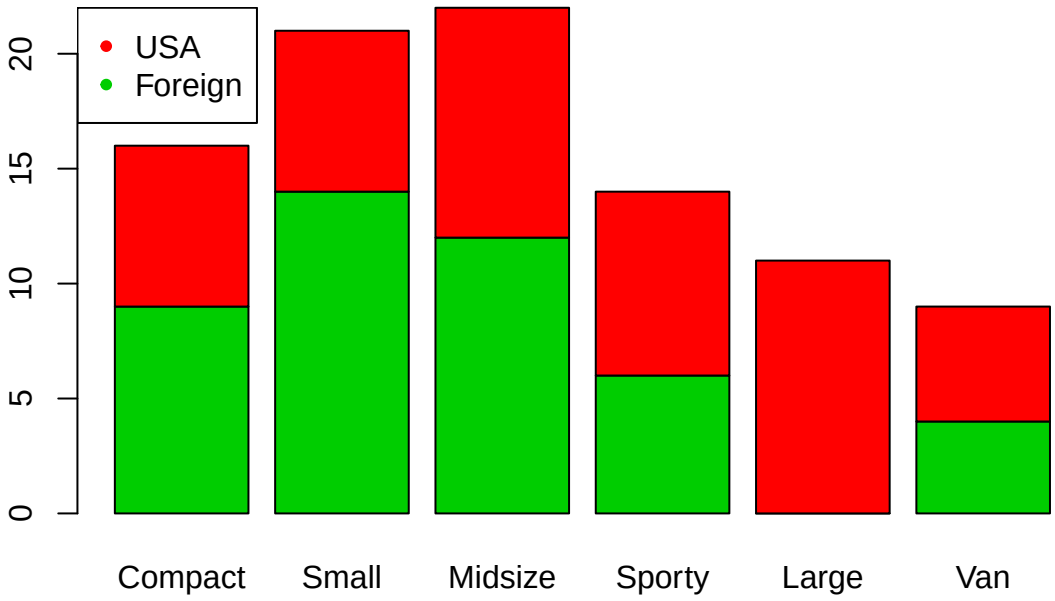
We might also want to compare categories from two different groups. For instance, we might want to compare majors for students but condition on gender (for instance we might want to see if some majors are popular regardless of gender, if some majors are more popular with one gender than another, etc.)

In this dataset, we can look at car type categories across maker nationality (US and non-US).

```
counts.usa <- table(Cars93$Type[Cars93$Origin=="USA"])
counts.foreign <- table(Cars93$Type[Cars93$Origin == "non-USA"])
counts.stacked <- rbind(counts.foreign, counts.usa)
counts.stacked #take a look at the format for the data
```

```
##                Compact Large Midsize Small Sporty Van
## counts.foreign       9     0      12    14      6   4
## counts.usa           7    11      10     7      8   5
```

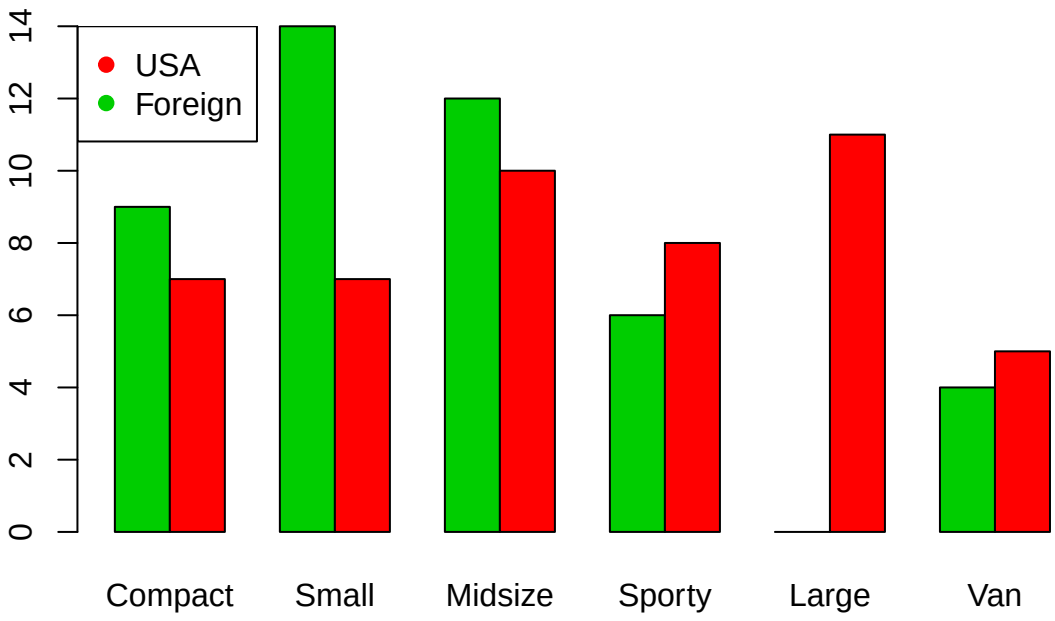
```
barplot(counts.stacked[,c(1, 4, 3, 5, 2, 6)], col=c(3,2))
legend(x="topleft", legend=c("USA","Foreign"), col=c(2,3), pch=20)
```



With stacked barplots, it's easy to see the total number in each category, as well as tell if certain categories are dominated by one group ('USA' or 'Foreign'). However, it's slightly more difficult to compare across categories (for instance in this example it's slighly difficult to compare USA models across categories).

Side-by-side barplots make it easy to compare across categories, as well as easy to compare groups within categories. On the other hand, it's more difficult to tell overall counts for categories.

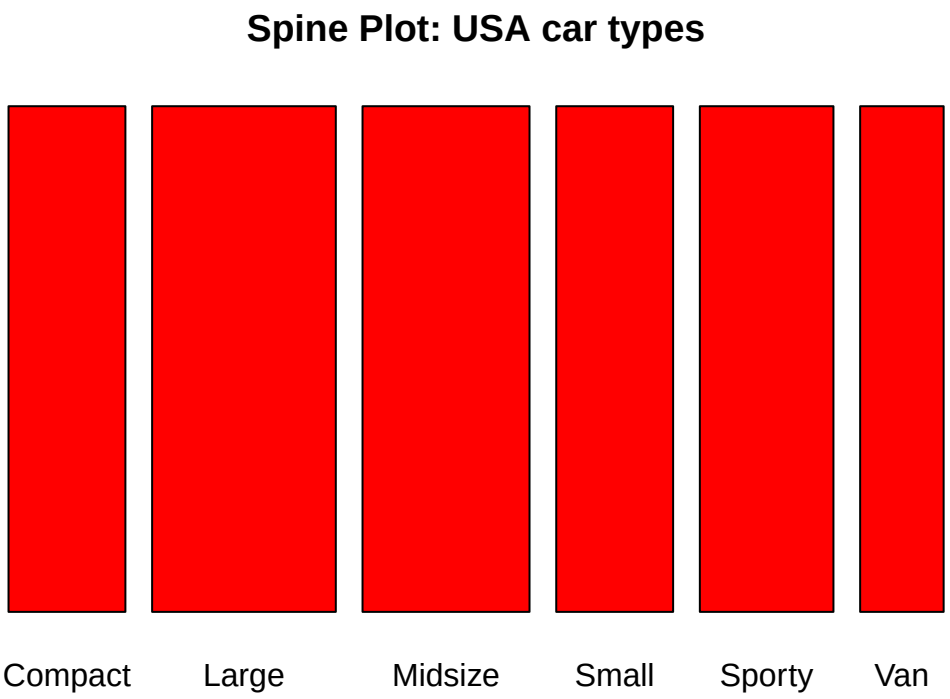
```
barplot(counts.stacked[,c(1, 4, 3, 5, 2, 6)], beside=TRUE, col=c(3,2))
legend(x="topleft", legend=c("USA","Foreign"), col=c(2,3), pch=19)
```



Spine Plots

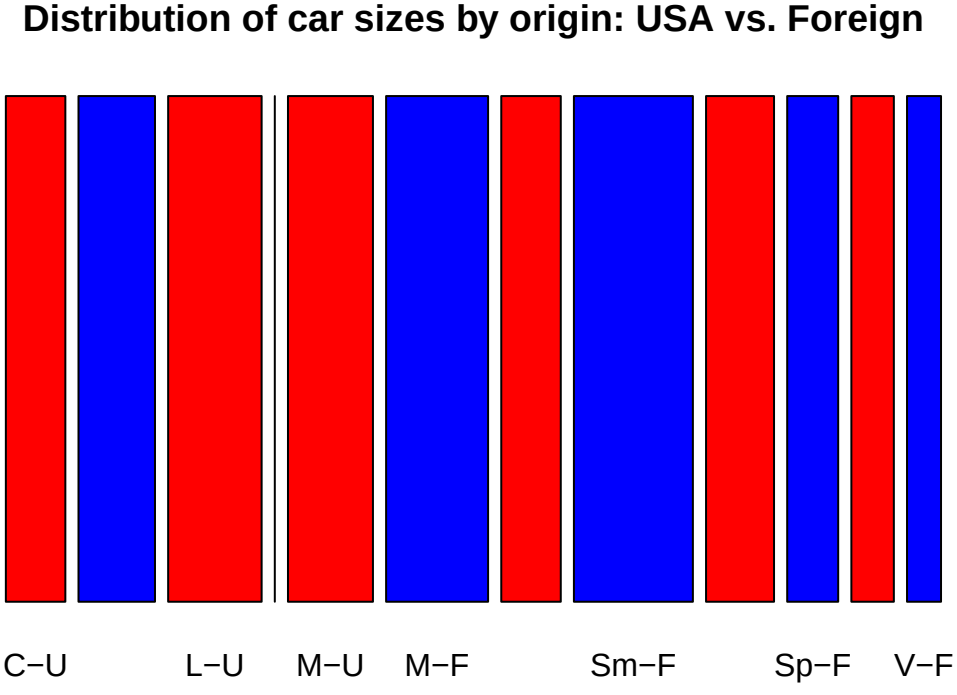
Height of bars is equal. Width of bar corresponds to proportion in that category.

```
barplot(rep(1,6), counts.usa, col='red', main="Spine Plot: USA car types", names.arg=names(counts.usa),
```



We can also use Spine Charts to compare two distributions by placing them side by side.

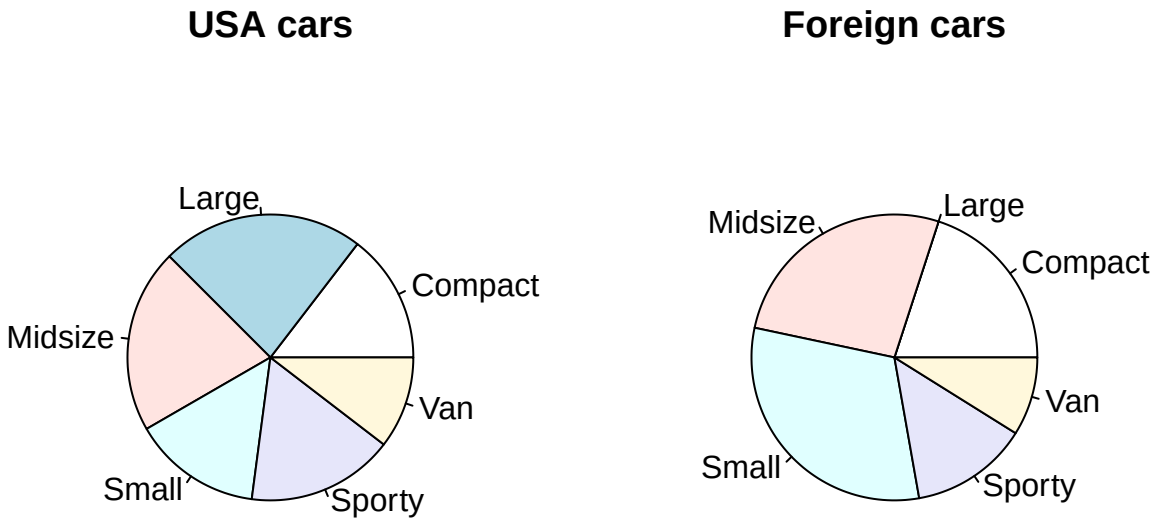
```
barplot(rep(1, 12), rbind(counts.usa, counts.foreign), beside=T, yaxt="n", col=rep(c(2,4), 6), names.arg
```



Pie Charts and its variants

If we care about overall proportions of types in a dataset, we can always look at the pie chart.

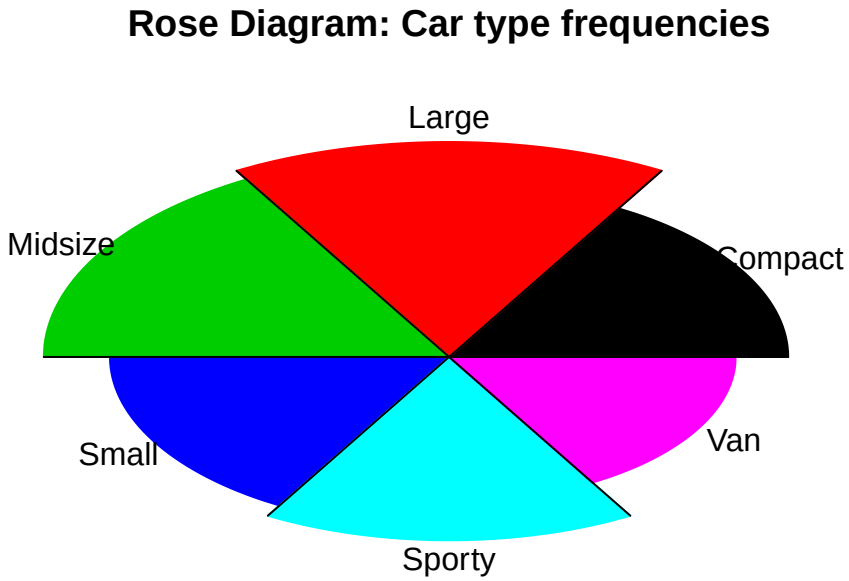
```
par(mfrow=c(1,2))
pie(counts.usa, main="USA cars")
pie(counts.foreign, main="Foreign cars")
```



Pie charts are slightly harder to compare both different categories within a group, and categories across group. Further, we lose all information about counts, and have no information about the relative sizes of one group to another.

Rose diagrams can alleviate this problem somewhat. The angles in a rose diagram are all the same. The radius is proportional to the square root of the category frequency.

```
source('Rose.R')
rose.simple(counts.usa, main.title="Rose Diagram: Car type frequencies", labels=names(counts.usa))
```



2-dimensional categorical data

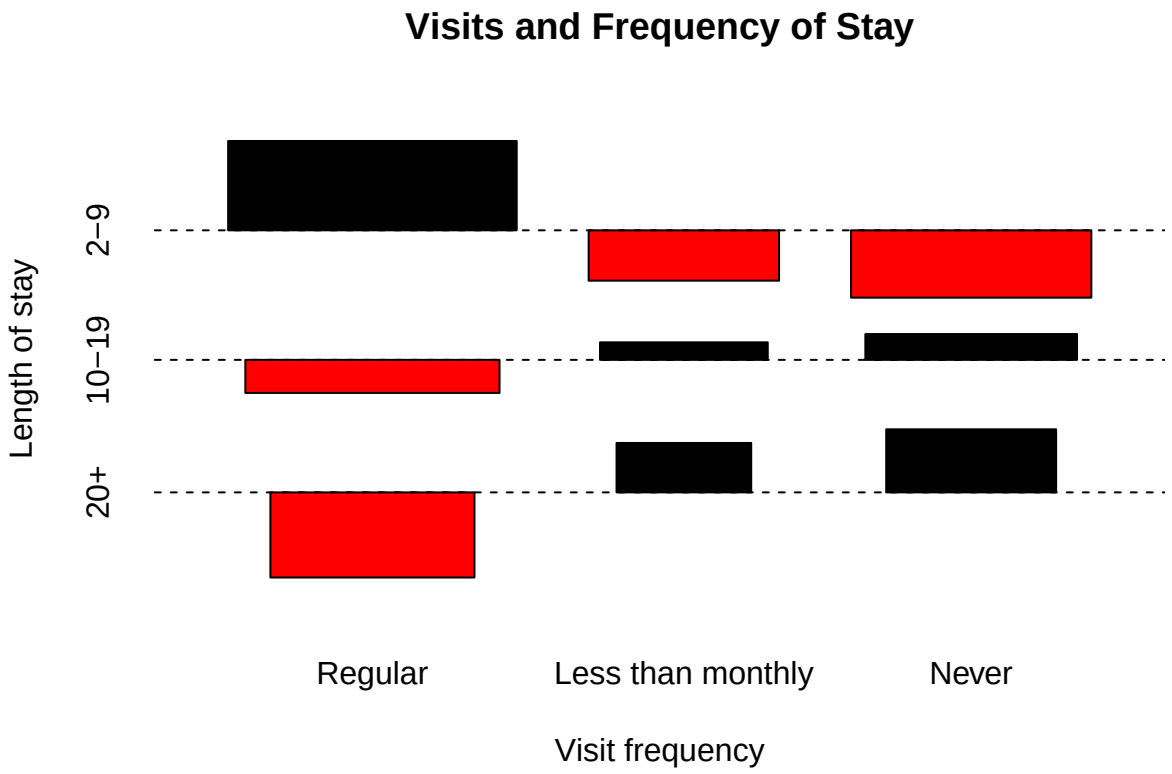
Here we are interested in the association between two variables.

```
library(graphics)
library(vcd)
data(Hospital)
data.table <- Hospital
```

Association plot

An association plot helps us see the deviation from independence.

```
assocplot(data.table, main="Visits and Frequency of Stay")
```

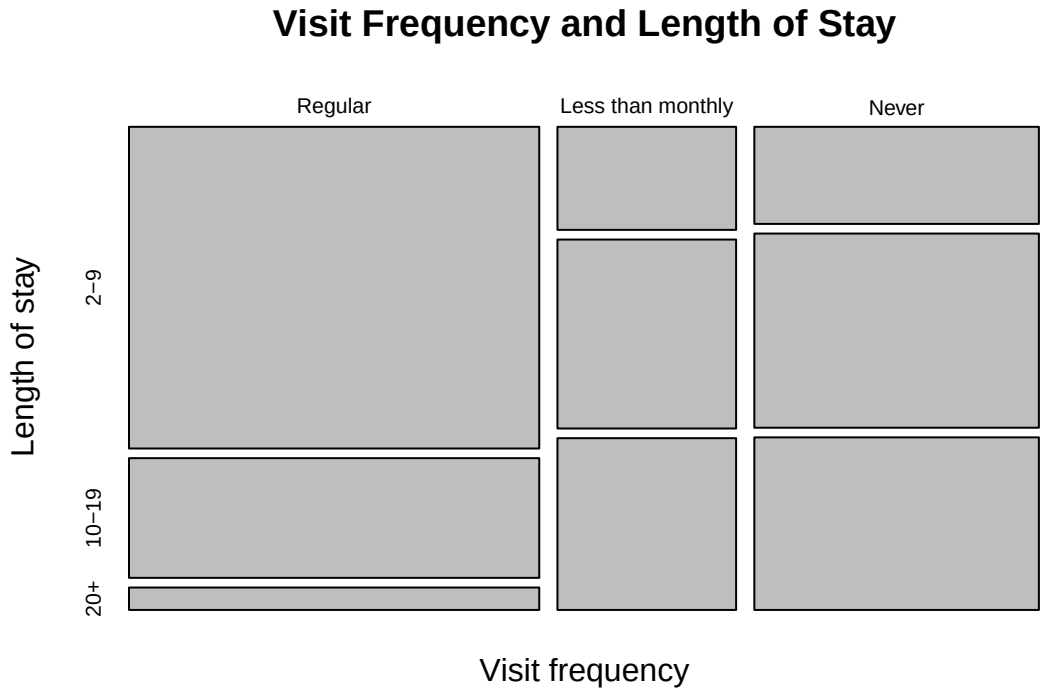


The association plot does not give you marginal or conditional distributions. However, it is helpful to display the deviation from what is expected. The area of the rectangle reflects the size of the deviation, and the direction (and color) reflects whether it is a positive or negative deviation.

Mosaic plot

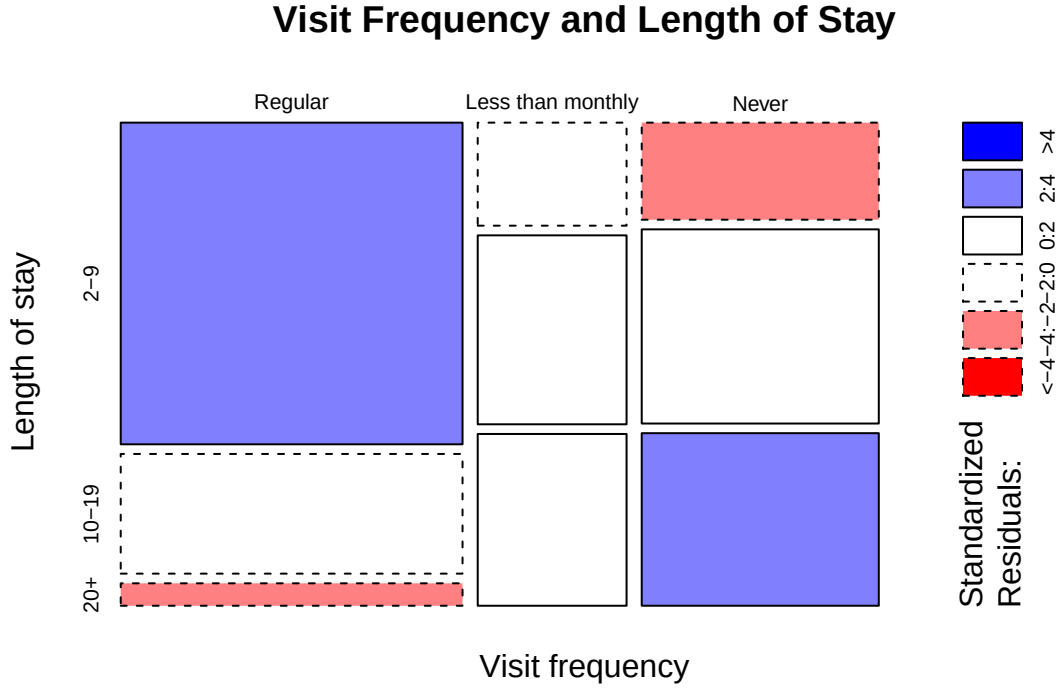
Another plot that can help us understand the distribution of two categorical variables in our dataset is a mosaic plot. The width of a bar is proportional to the percent in variable 1 of the category, and the height of a segment is proportional to the percent of variable 2 of the category, GIVEN the particular category for variable 1.

```
mosaicplot(data.table, main="Visit Frequency and Length of Stay")
```



Now, we can be crazy and color this by their difference from what was expected (this brings this more in line with the association plot in terms of its information)

```
mosaicplot(data.table, main="Visit Frequency and Length of Stay", shade=T)
```



The residuals are the Pearson residuals

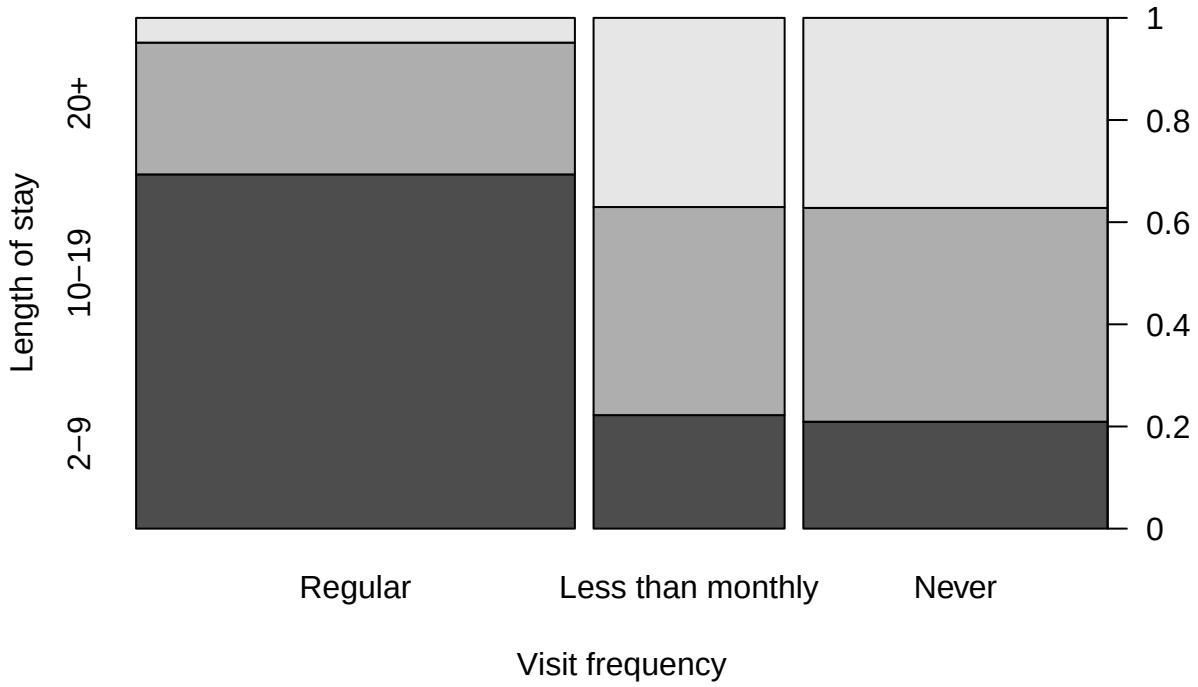
$$r_{ij} - \frac{c_{ij} - \hat{c}_{ij}}{\sqrt{\hat{c}_{ij}}}$$

which is the observed - expected divided by square root of expected (how far off from expected we are, divided by a notion of how large the expected is. This makes sense- if we expect something to be around 1, then 5 is a ways away. If we expects something to be aroudn 1001, 1005 isn't all that bad. Our thoughts on how far off we are in each scenario changes, even though the absolute difference is the same in both cases.)

Spinogram

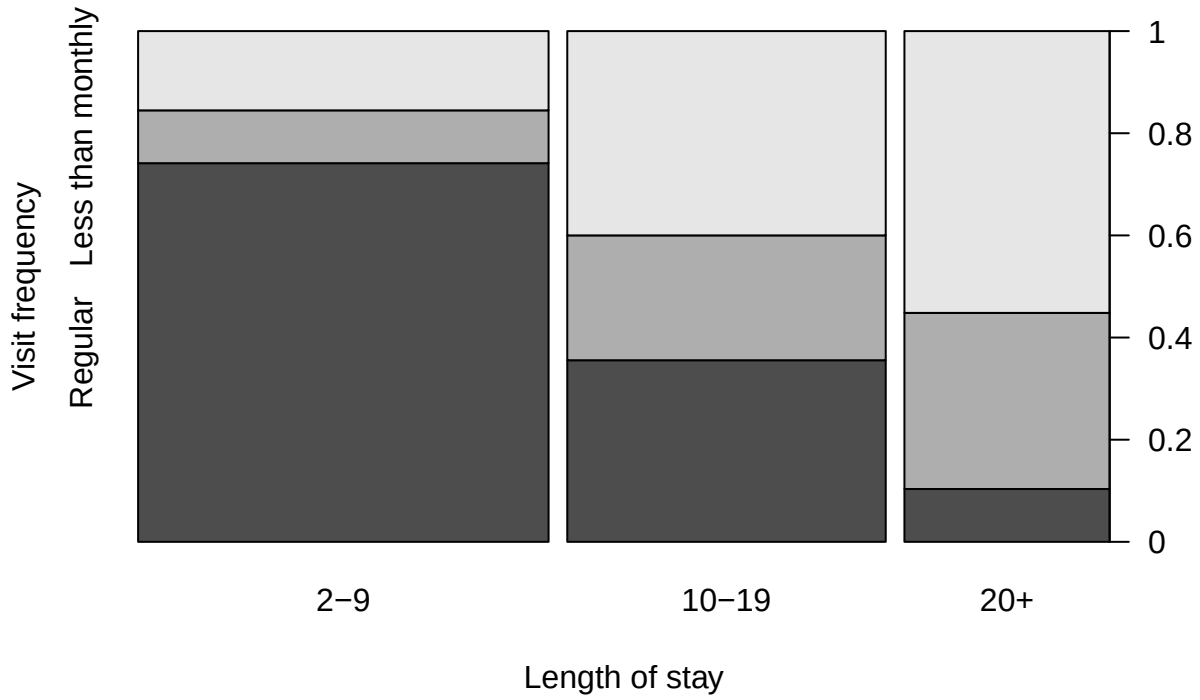
We can see the spinogram as a special case of a mosaic plot, and a generalization of a stacked bar plot. We get the marginal information of the first categorical variable (the 'x' axis) and conditional information for the second. Percentage guidelines are provided on the right hand side.

```
spine(data.table)
```



We can flip the order of the variables to get marginal information for the other variable, and conditional for the second:

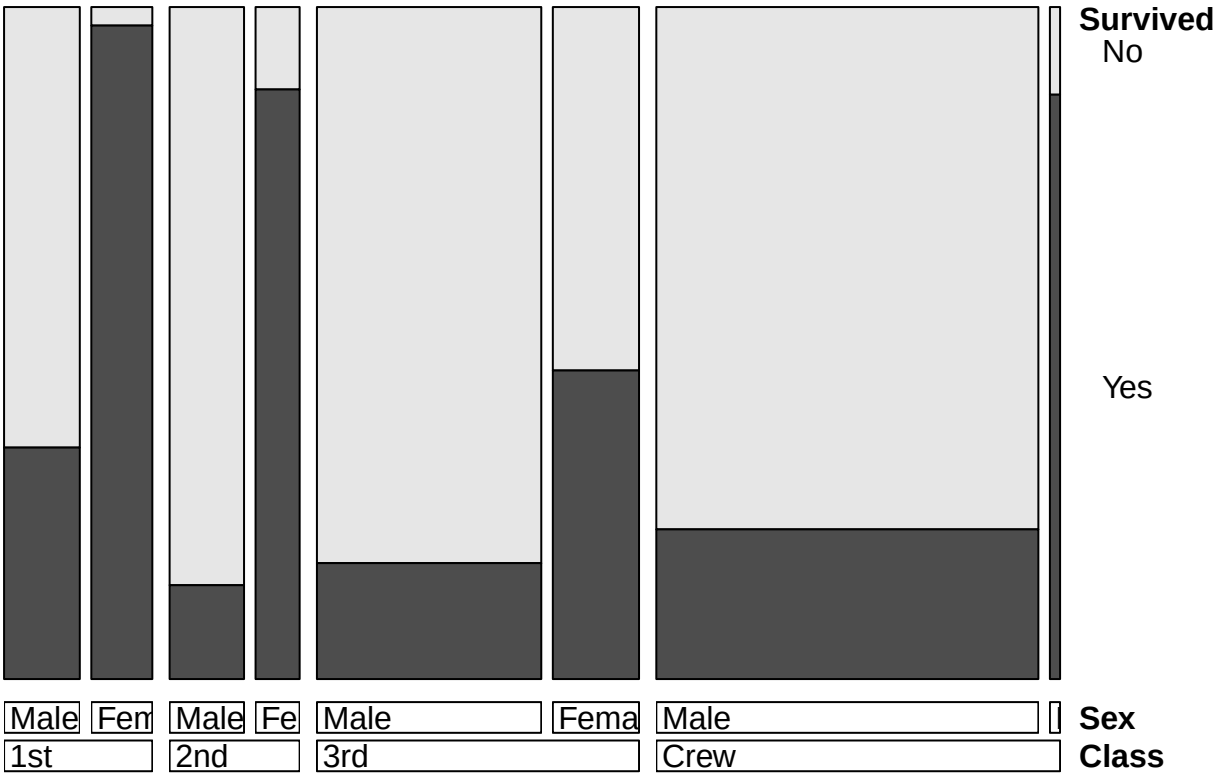
```
spine(t(data.table))
```



Double Decker

The Double Decker is similar to the spinogram but allows for more flexibility for higher dimension.

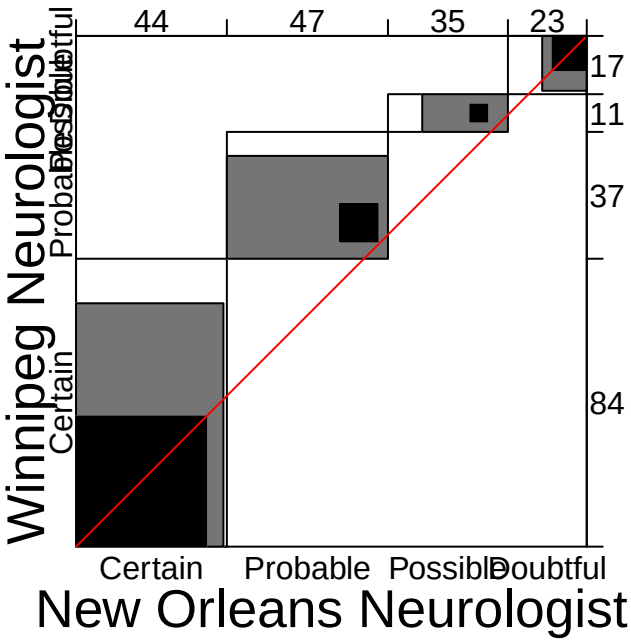
```
Titanic.3 <- apply(Titanic, c(1,2,4),sum)
doubledecker(Titanic.3)
```



Agreement Plot

Now let's say you have two sources scoring items of some sort, and you want to see how much they agree, and specifically how their agreement distributes across categories. The agreement plot is handy for this.

```
data.table <- t(MSPatients[,,1])
agreementplot(data.table)
```

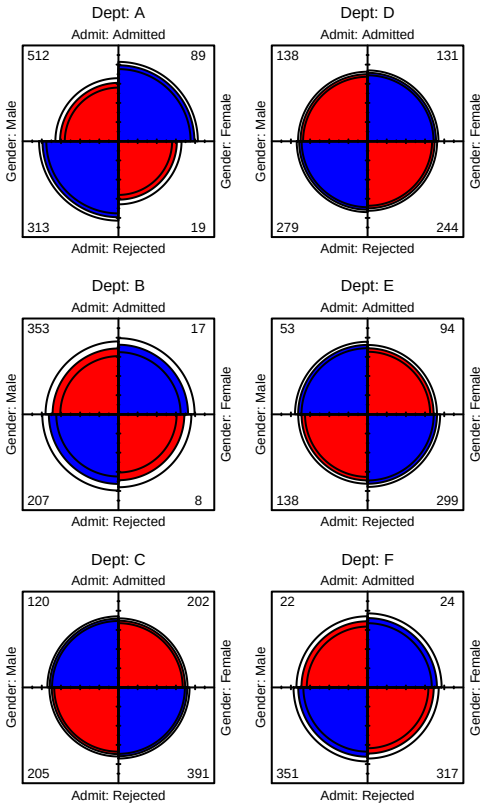


The black boxes indicate the proportion within each category that both people (or things or whatever) agree on that category. The grey boxes indicate where they almost agree on a category (so their categories are just one off), and the white boxes represent the proportion in which they have strong disagreement. The diagonal line gives you marginal homogeneity- in other words, that they place the same number of observations in each category. Departure from the horizontal line shows observer bias (in other words, if oen person almost never puts observations into category 2, this bias will show here.)

Four Fold Plots

Four fold plots display the odds ratio/association in 2x2 contingency tables. Association is apparent if not all quadrants have equally sized radii. We can test for 95% confidence in these plots by looking for overlab in the bands around the arcs of the quadrants. If the bands overlap, then we cannon reject the null hypothesis that there is no association.

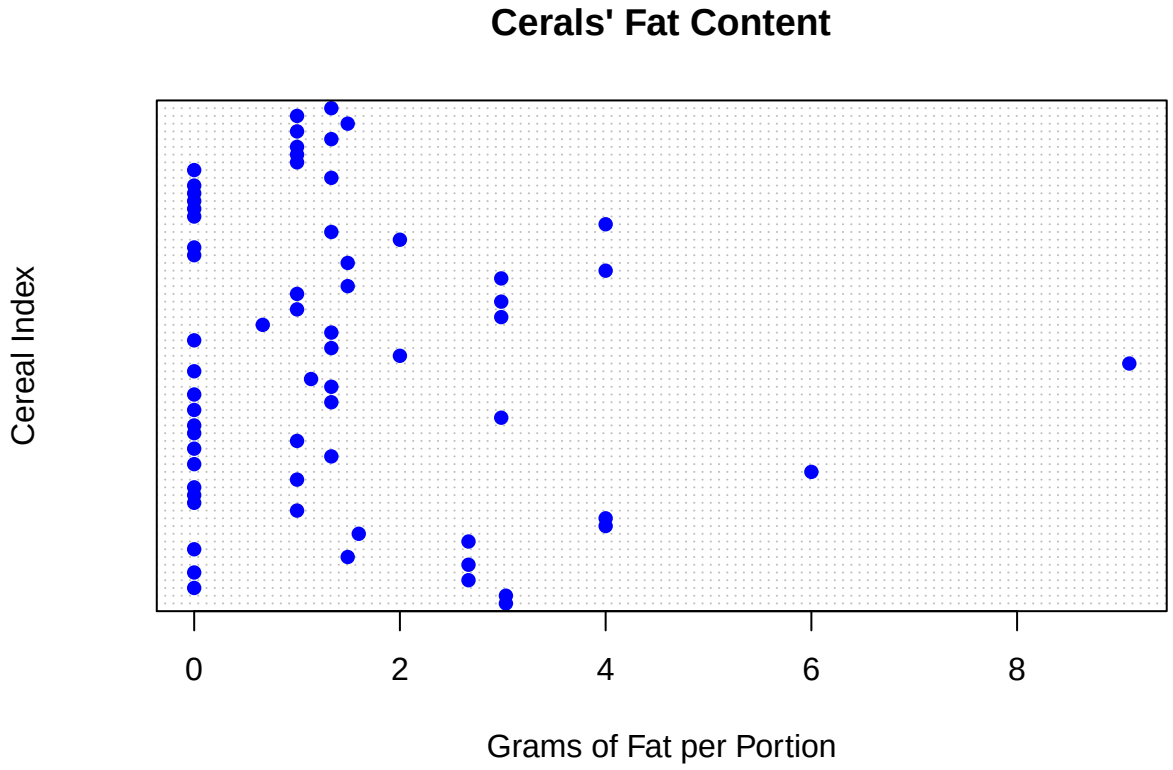
```
fourfoldplot(UCBAdmissions, col=c(2,4))
```



1-dimensional continuous data

With 1-dimensional continuous data the big question is to understand the distribution of the data. What is the shape of the distribution, are there any interesting features about the distribution.
Dot chart ———— The dot chart lets us see the values for each data point.

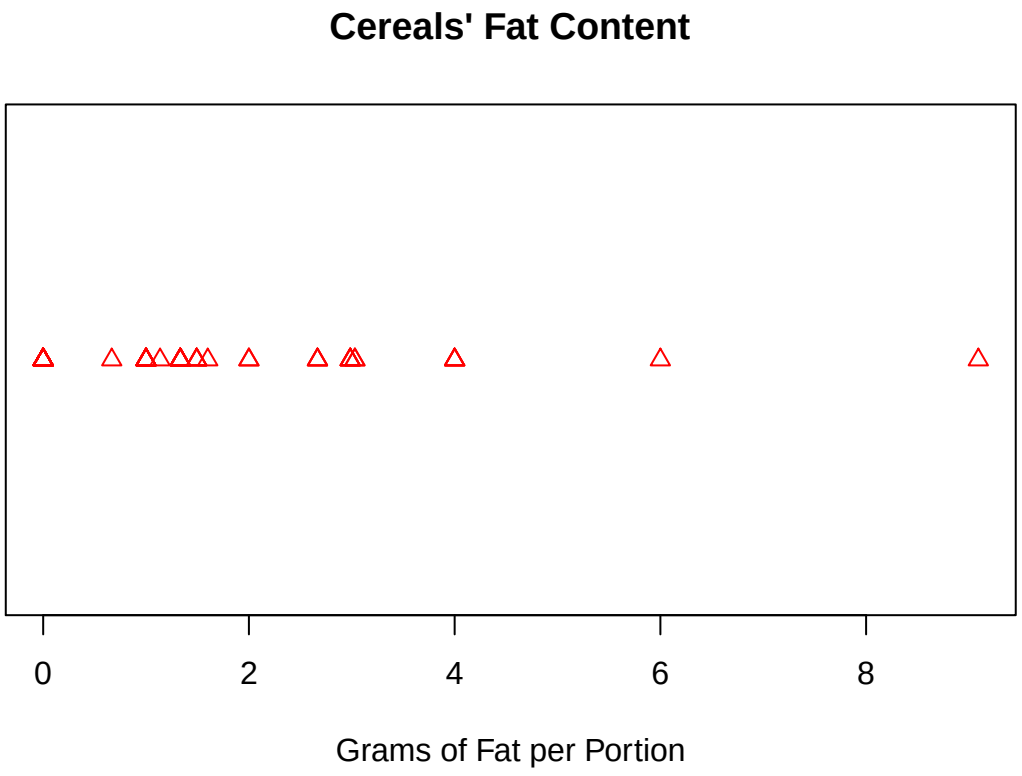
```
data(UScereal)
attach(UScereal)
dotchart(fat, pch=16, xlab="Grams of Fat per Portion", col=4, ylab="Cereal Index", main="Cerals' Fat Co
```



Strip chart

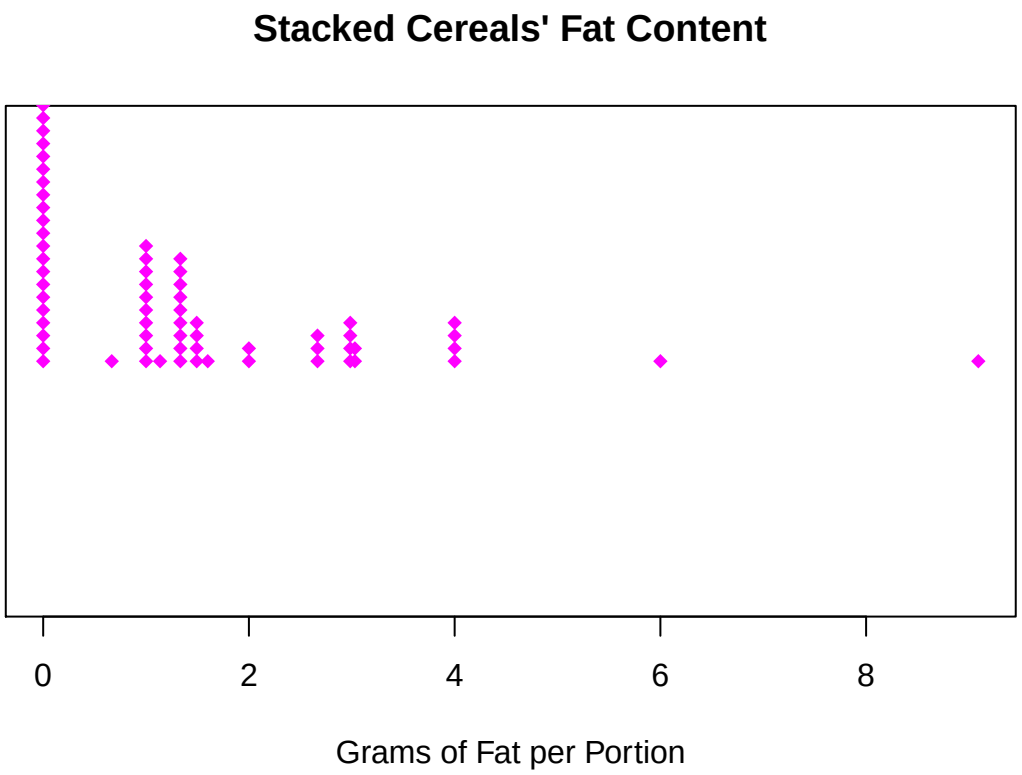
A strip chart is similar to a dotchart except that plots data at the same y-value (so we lose association between the value of the variable and the actual data point). This plots the data with almost no ink, which is great, and it is excellent for small datasets.

```
stripchart(fat, pch=2, col=2, xlab="Grams of Fat per Portion", main="Cereals' Fat Content")
```



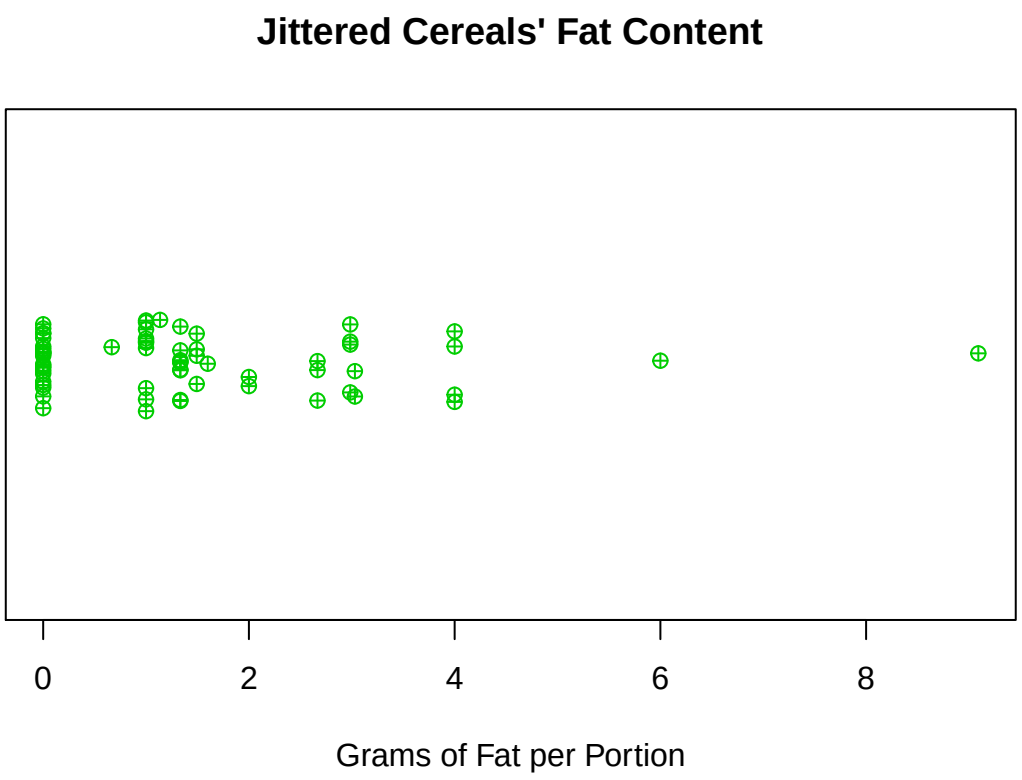
As can be seen, however, it's difficult to see similar values (especially repeated values). We have two options. First is to stack the similar values.

```
stripchart(fat, method="stack", pch=18, col=6, xlab="Grams of Fat per Portion", main="Stacked Cereals' F
```



However, this suffers if values aren't repeated, but are just extremely close together. The stack method only works if they are actually repeated. So we can use method jitter instead!

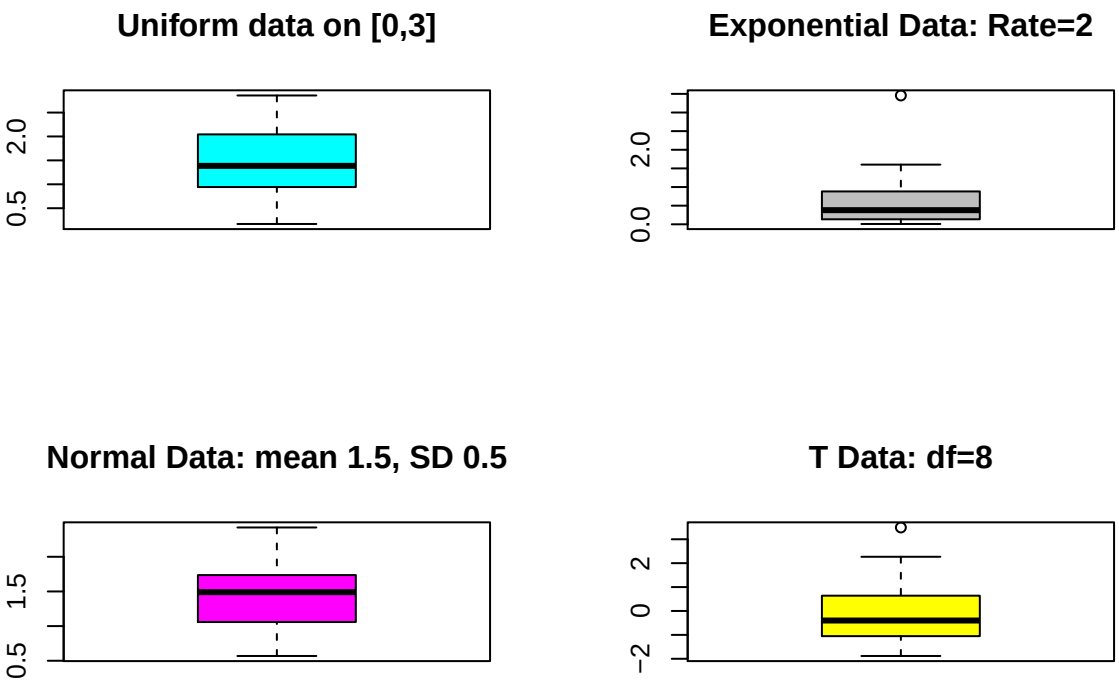
```
stripchart(fat, method="jitter", pch=10, col=3, xlab="Grams of Fat per Portion", main="Jittered Cereals
```



Boxplot

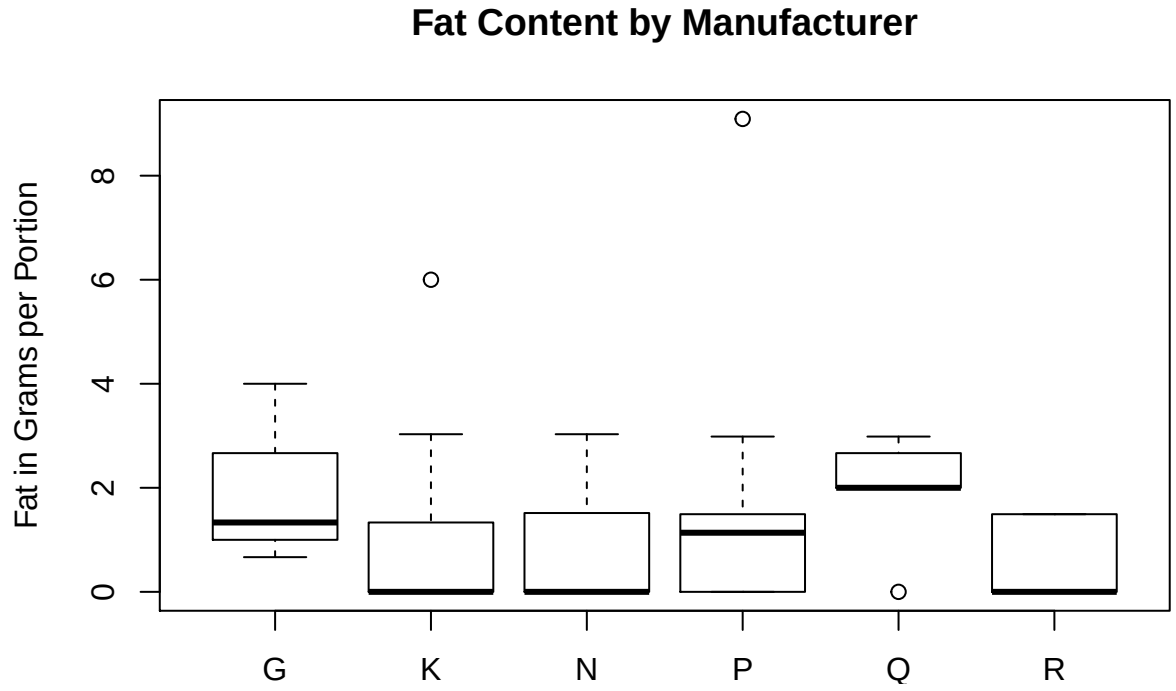
The boxplot can help us identify features of our distribution like symmetry, skew, etc. The middle line of a boxplot is the median, the edges of the box are the 25% and 75% quantiles (so the distance between the edges is the interquartile range). The 'whiskers' extend to the maximum and minimum value, UNLESS the max and the min fall outside of 1.5 times the interquartile range beyond the 75th (or 25th) quantile. Then the whiskers denote this value (75th quantile + 1.5*IQR, for example) and data that lies outside of this range is plotted as a point.

```
par(mfrow=c(2,2))
x <- runif(50, 0, 3); boxplot(x, col=5, main="Uniform data on [0,3]")
x <- rexp(50, 2); boxplot(x, col=8, main="Exponential Data: Rate=2")
x <- rnorm(50, 1.5, 0.5); boxplot(x, col=6, main="Normal Data: mean 1.5, SD 0.5")
x <- rt(50,8); boxplot(x, col=7, main="T Data: df=8")
```



In order to really compare distributions using boxplots, you really should ensure that the y range is the same. One easy way to do this is to comine them into the same graphics window.

```
boxplot(fat~mfr, ylab="Fat in Grams per Portion")
title("Fat Content by Manufacturer")
```

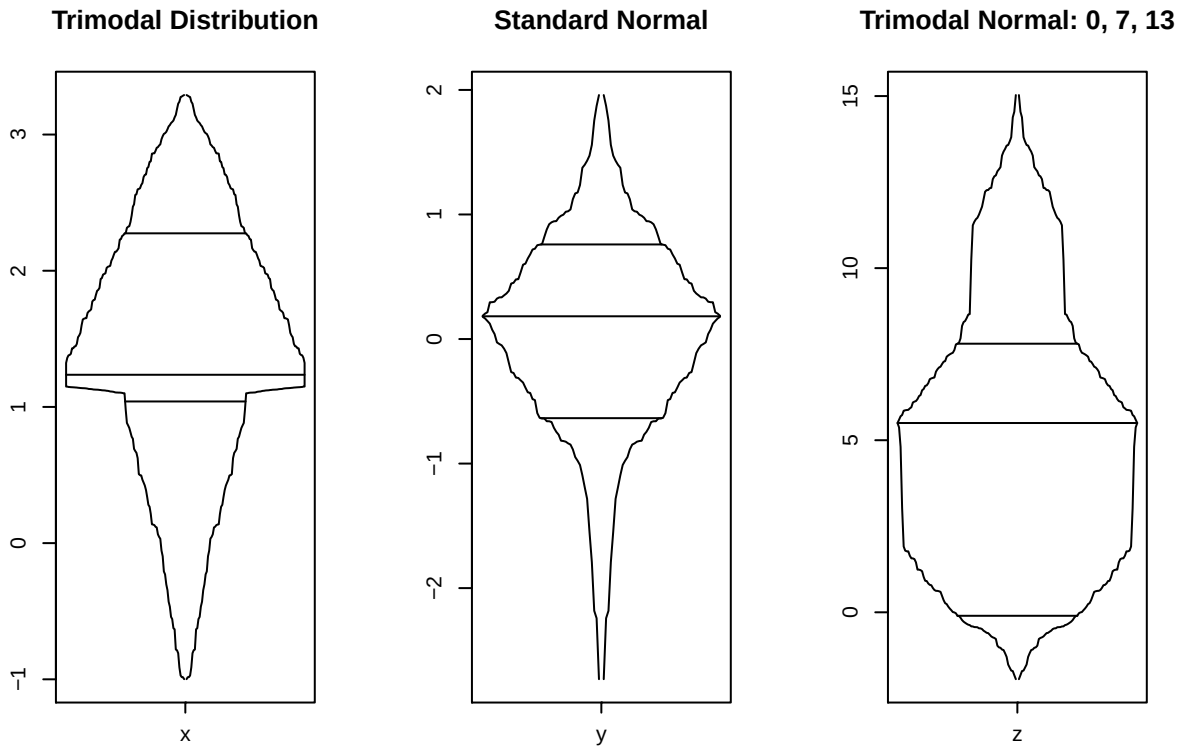


One potential downside of a boxplot is that it doesn't give a very fine-grained view of the distribution. It's very, well, boxy. There are several tools that can help us get an even better view of the distribution.

Box-Percentile Plot

The box-percentile plot combines a box plot with the cumulative distribution function. The width of the box is proportional to the probability of seeing that value or a value that is more extreme.

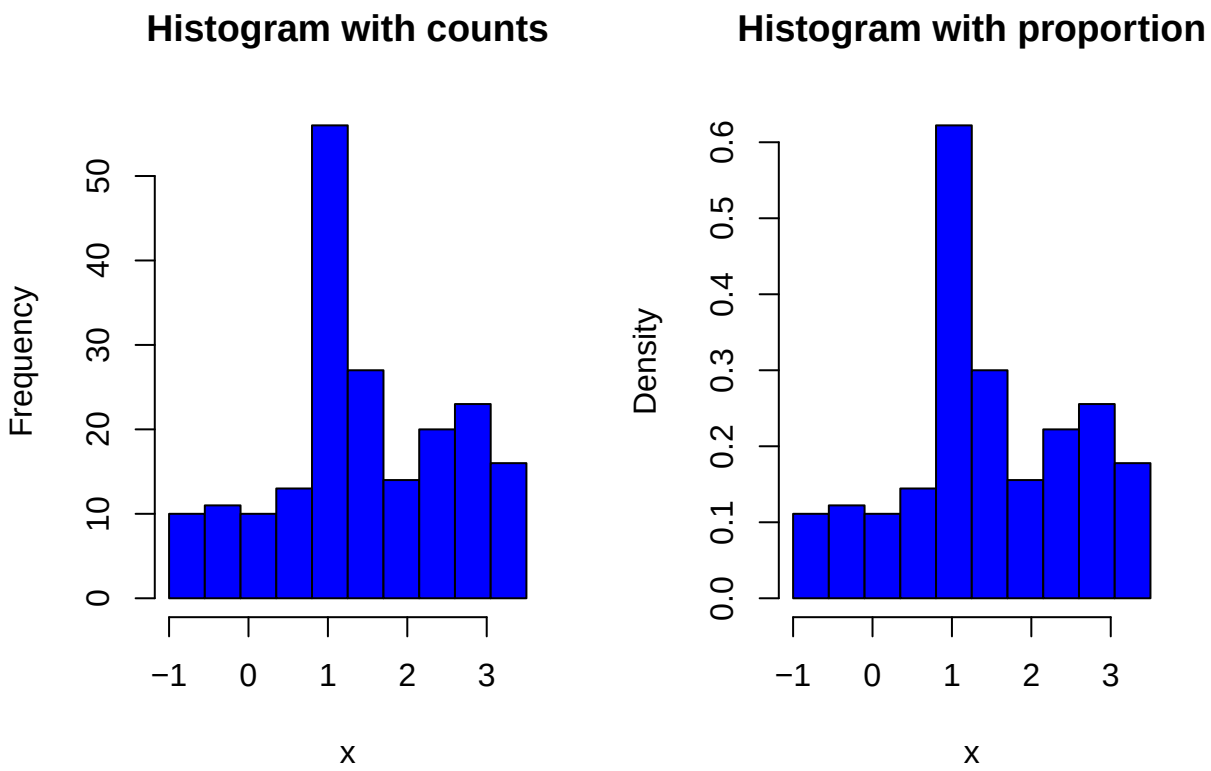
```
par(mfrow=c(1,3))
x <- c(runif(50, -1, 1), runif(50, 1.1, 1.15), runif(100, 1.3, 3.3))
y <- rnorm(100, 0, 1)
z <- c(rnorm(75, 0, 1), rnorm(50, 7, 1), rnorm(30, 13, 1))
library(Hmisc)
bpplot(x, main="Trimodal Distribution")
bpplot(y, main="Standard Normal")
bpplot(z, main="Trimodal Normal: 0, 7, 13")
```



The Histogram

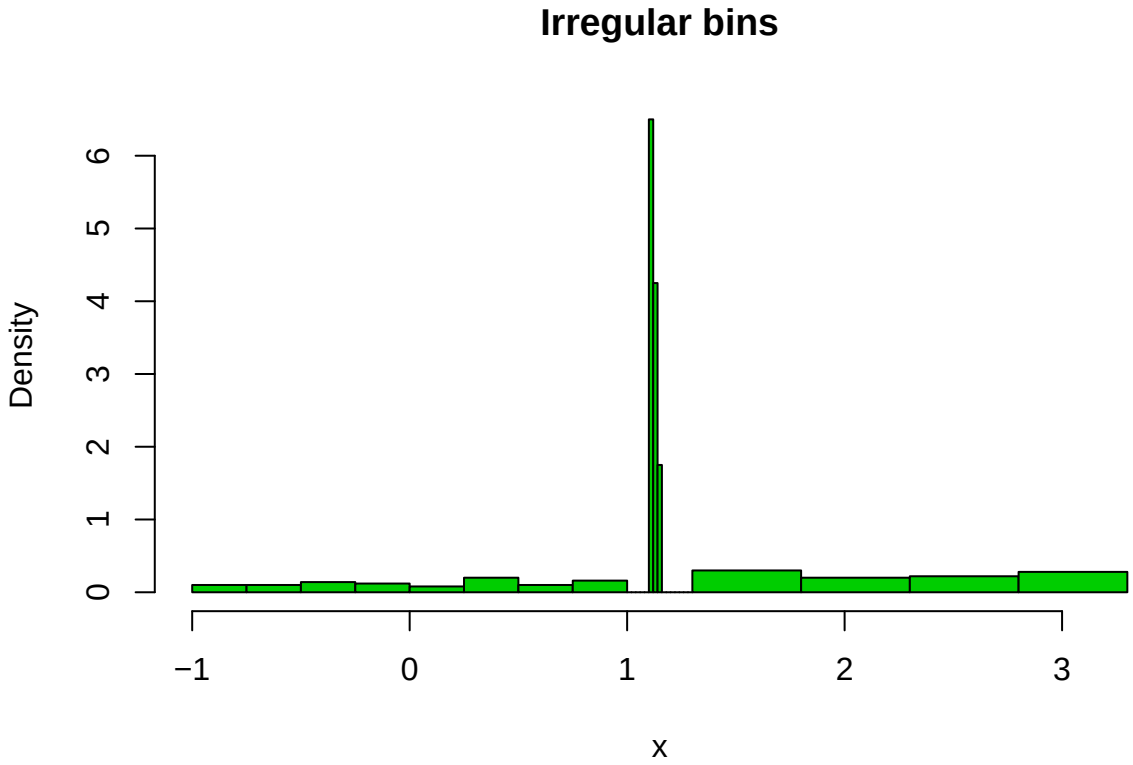
Perhaps the most well known of the plots, the histogram attempts to give some sense of the density of a series of observations. It does so by defining windows over the range of observed observations, and counts the number of observations that occur in those windows. The heights of the bars in a histogram correspond to either the count of observations, or the proportion of observations, in each window. Histograms tell us min/max, skewness, symmetry, tail behavior, modality. Quantiles are much harder to obtain from a histogram.

```
x <- c(runif(50, -1, 1), runif(50, 1.1, 1.15), runif(100, 1.3, 3.3))
par(mfrow=c(1,2))
hist(x, breaks=seq(-1, 3.5, length=11), col=4, main="Histogram with counts")
hist(x, breaks=seq(-1, 3.5, length=11), col=4, main="Histogram with proportion", prob=T)
```



We can also use unequal bin widths. In this case, the area of the rectangle is proportional to the probability of seeing observations in the bin (this is true of regular histograms as well, but our eyes default to using heights)

```
bins <- c(seq(-1,1,by=.25), seq(1.02,1.30, by=.02), seq(1.8, 3.3, by=.5))
hist(x, breaks=bins, col=3, main="Irregular bins")
```

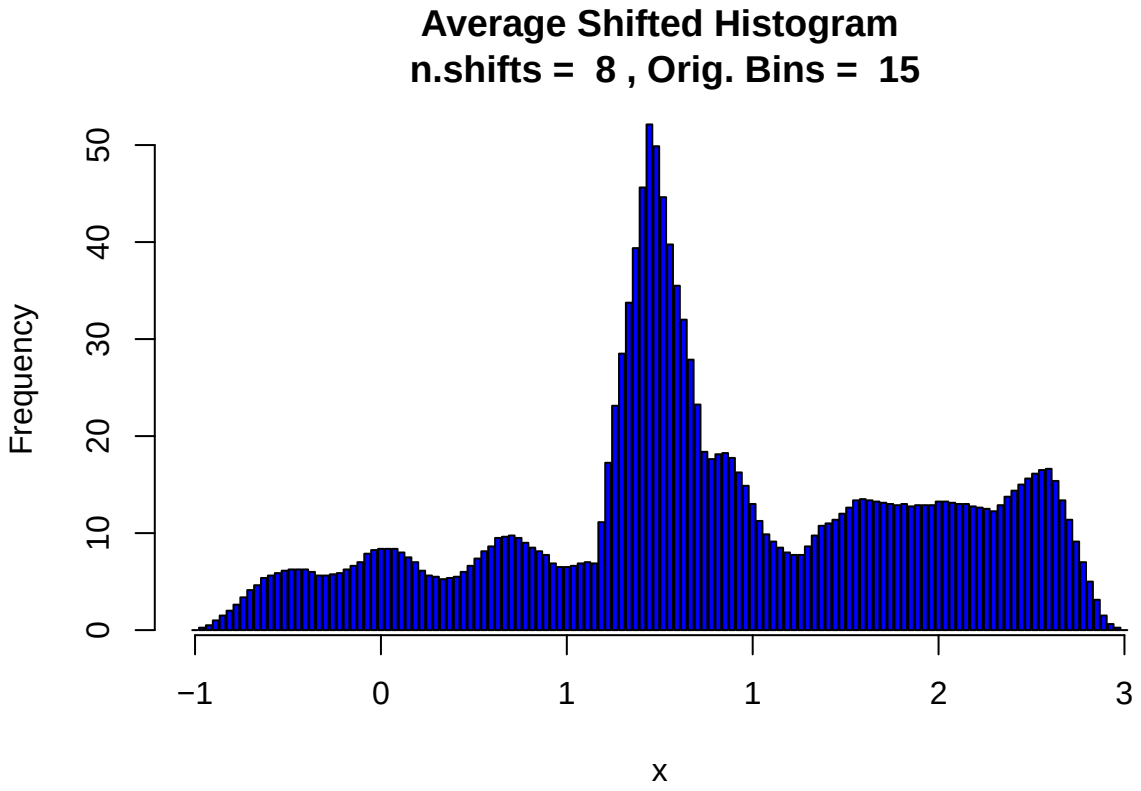


There are two major concerns with histograms. First is choice of bin width. Second is bin location. Bins that are too wide won't capture the nuances of a distribution, and bins that are too narrow will aspects of a distribution that aren't really there (extra modes, 'wiggles' etc).

Average Time Shiften Histogram

One solution to this is the Average Time Shifted Histogram. This is essentially averages of multiple histograms that are shifted versions of each other. In ashcode.R, we can use ashcode1D(x, m, nbins), where x is the data, m is the number of shits, and nbins are the number of bins in our histograms.

```
source('ashcode.R')
ashcode1D(x, 8, 15)
```



Density Estimation

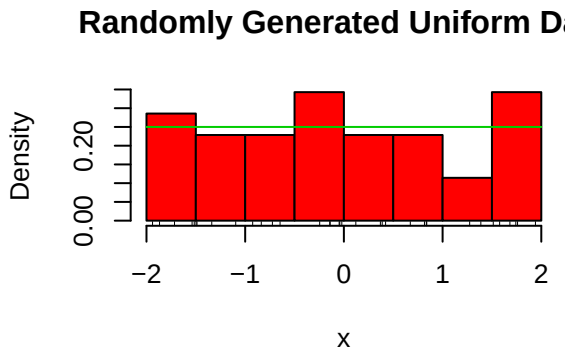
Another approach to visualizing the density of a 1-dimensional continuous variable is to attempt to estimate the density function $f(x)$. We do so using Kernel Density Estimation. The idea behind KDE is to estimate

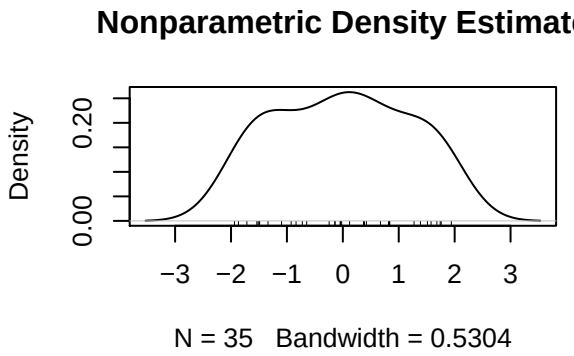
$$\hat{f}(x) = \frac{1}{nh}\sum_{j=1}^{n} K(\frac{x - x_j}{h})$$

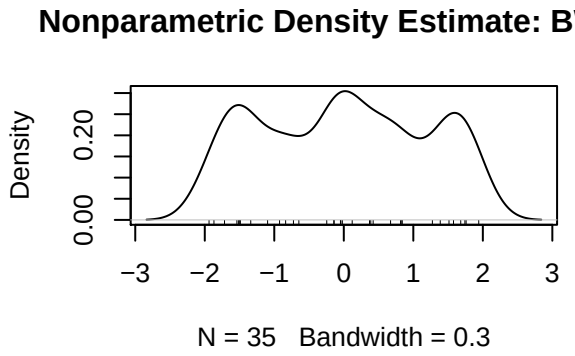
where n is the number of observations, h is the bandwitdth, and K is the chosen Kernel. Let's see the KDE in action. For our bandwith, large h smooths out the curve. A larger number of points are contributing larger values to the sum of the $K()'s$. Smaller h values means that only the points close to the x of choice are really contributing anything (so x values where you see a lot of observations will have high density, and x values where there aren't many observations in the near vicinity will have density close to zero). Small h leads to really wiggly density estimates.

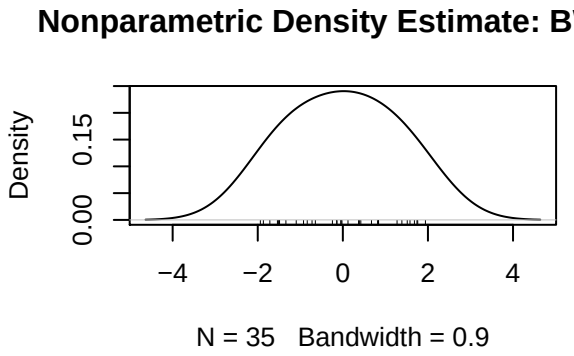
In addition to changing the bandwitdth, we can change the kernel, K, that we are using (gaussian, triangular, rectangular, etc).

```
x <- runif(35, -2, 2)
par(mfrow=c(2,2))
hist(x, col=2, main="Randomly Generated Uniform Data", prob=T)
curve(dunif(x, -2, 2), col=3, add=T); rug(x)
dens.est <- density(x)
plot(dens.est, main="Nonparametric Density Estimate"); rug(x)
dens.est.3 <- density(x, bw=.3)
plot(dens.est.3, main="Nonparametric Density Estimate: BW=.3"); rug(x)
dens.est.9 <- density(x, bw=.9)
plot(dens.est.9, main="Nonparametric Density Estimate: BW=.9"); rug(x)
```

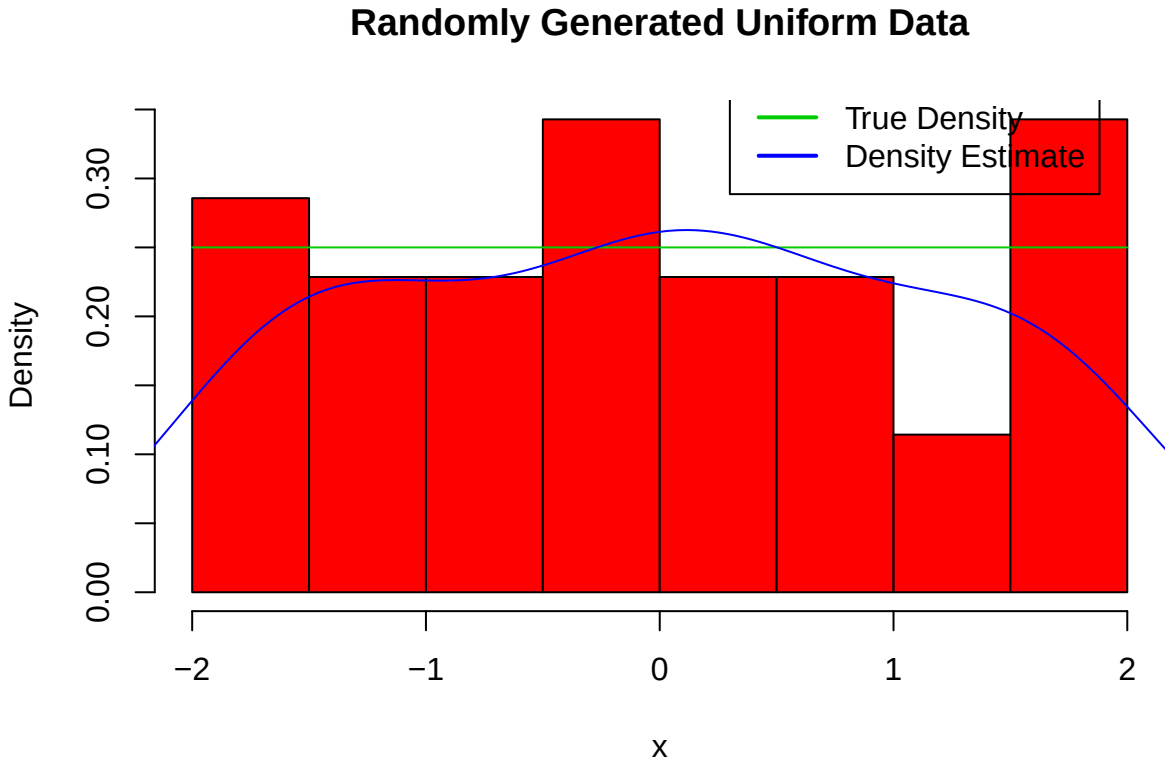








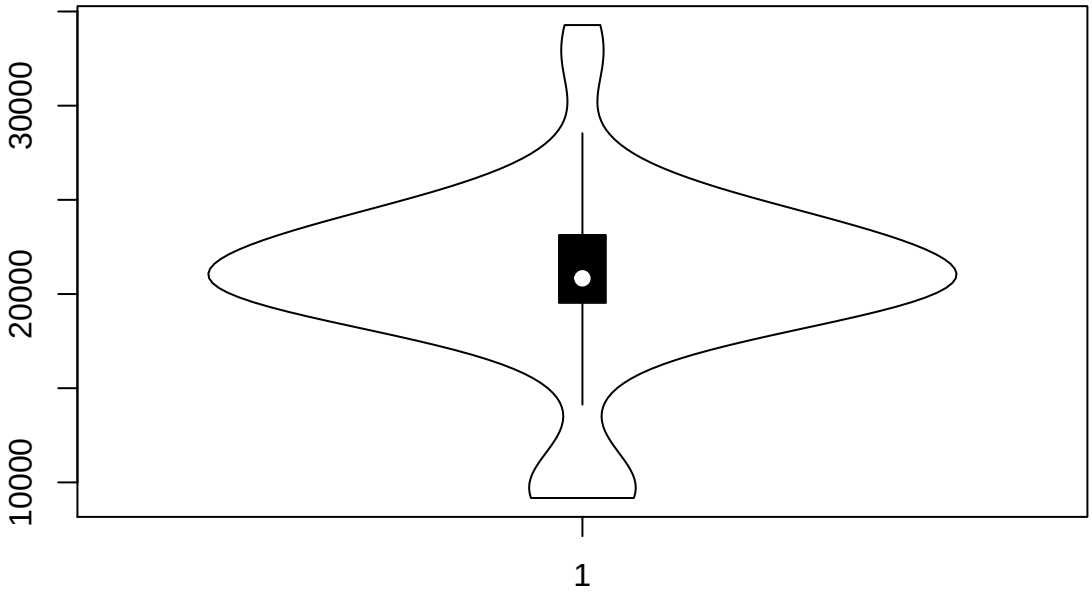

```
par(mfrow=c(1,1))
hist(x, col=2, main="Randomly Generated Uniform Data", prob=T)
curve(dunif(x, -2, 2), col=3, add=T)
lines(dens.est, col=4)
legend(.3, 0.4, c("Histogram", "True Density", "Density Estimate"), col=c(2,3,4), lwd=2)
```



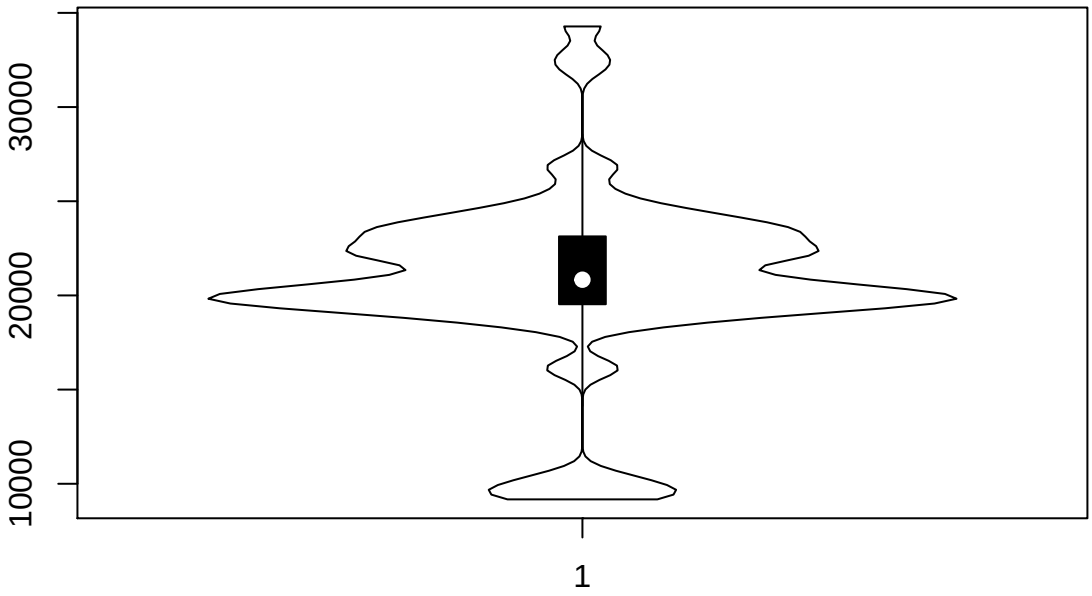
The Violin Plot

Just as the box-percentile plot combined box plots and continuous distribution functions, the violin plot combines the boxplot and the kernel density estimate. We can change the bandwith h.

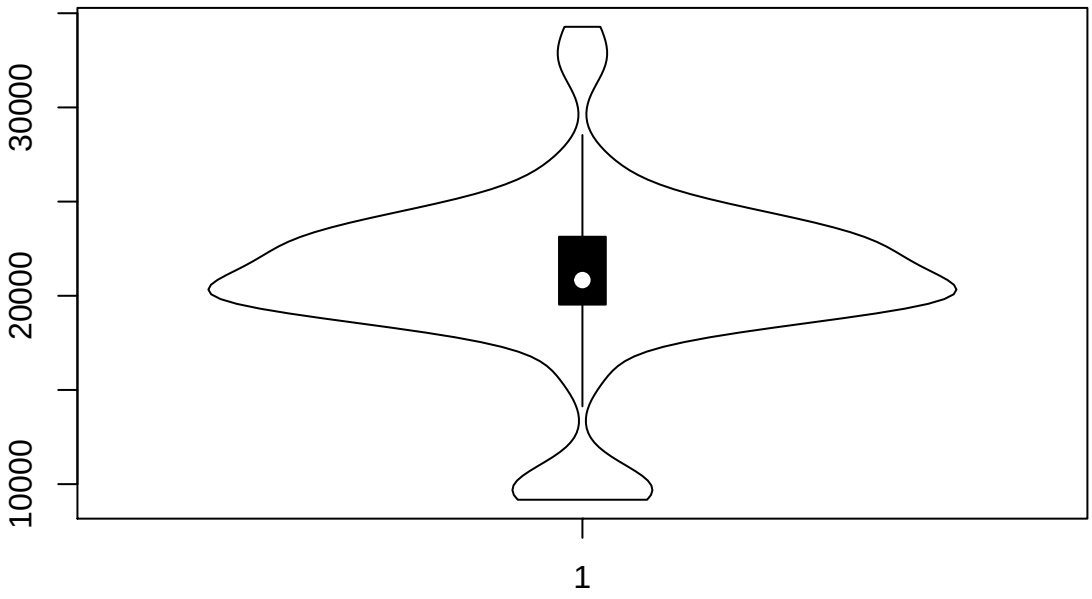
```
library(vioplot); library(sm)
vioplot(galaxies, col="white")
```



```
vioplot(galaxies, h=500, col="white")
```



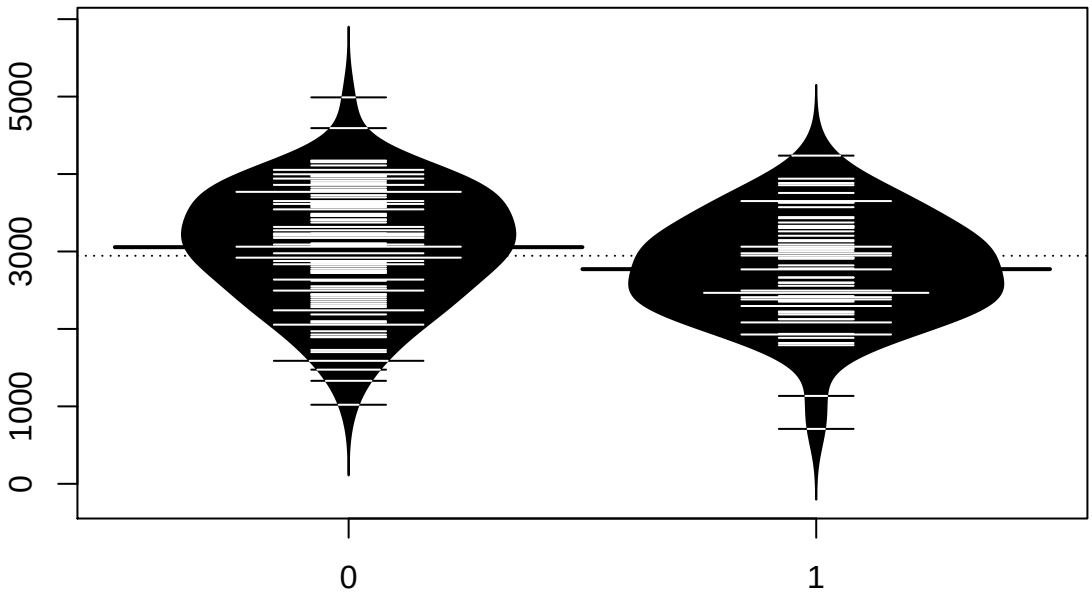

```
vioplot(galaxies, h=1250, col="white")
```



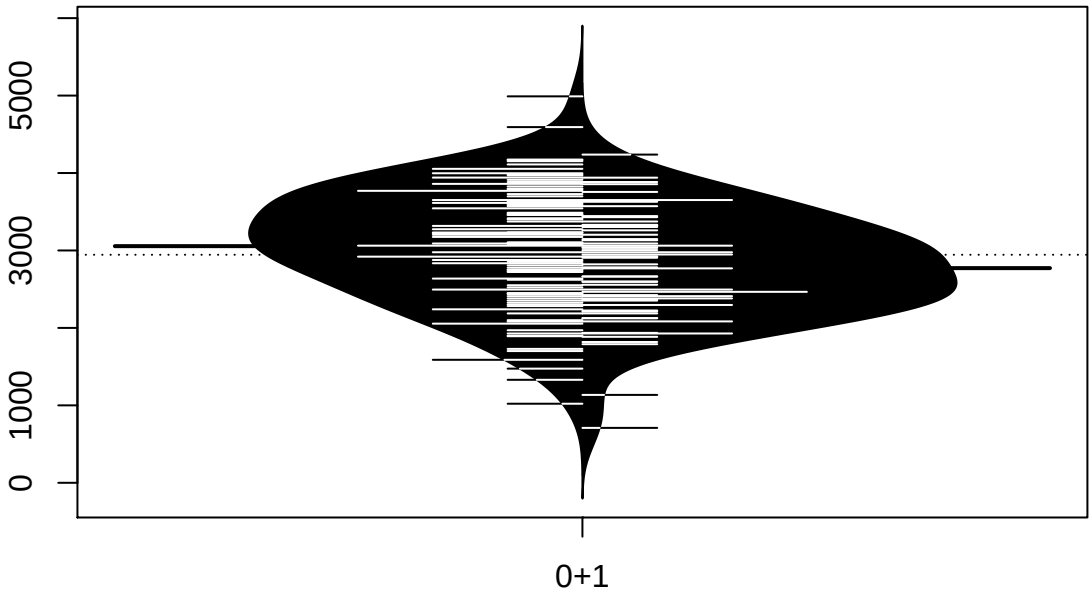
Note that changing the bandwidth h doesn't change the boxplot (because the boxplot doesn't estimate the density function)

Bean plot

```
library(MASS); attach(birthwt)
library(beanplot)
beanplot(bwt~smoke)
```



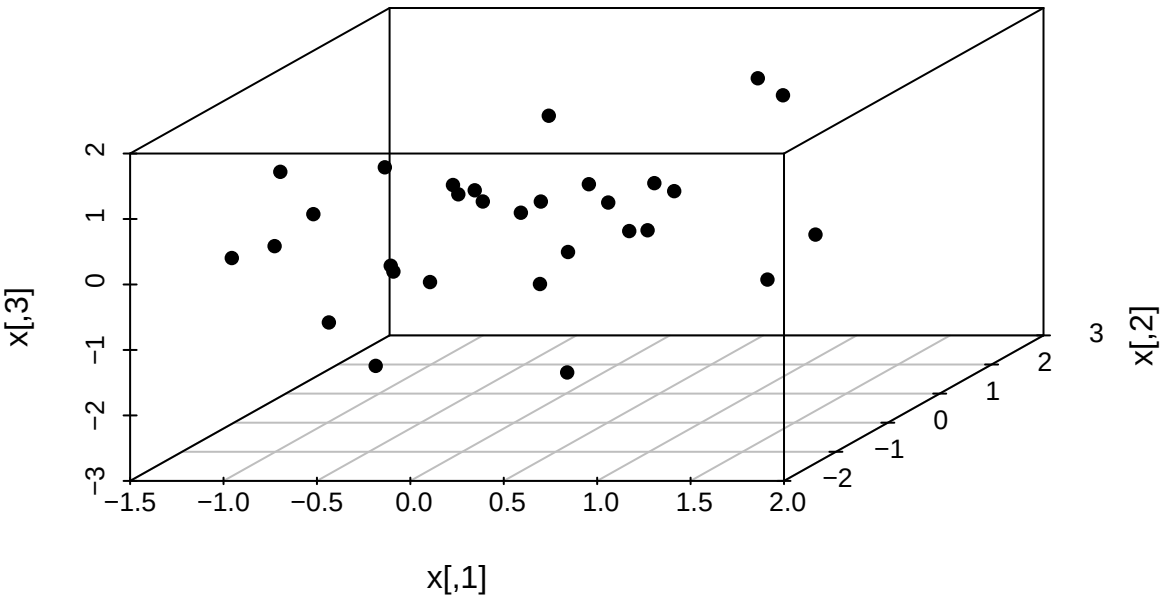

```
beanplot(bwt~smoke, side="both")
```



Higher Dimensions

If we have 3 dimensions that we would like to visualize, one strategy is to plot them in a 3-d plot. This gives us a great idea as to how dimensions of data relate to one another. The library 'scatterplot3d' allows us to do this. One downside is that this plot is fixed (though there is some wiggle room in the parameters), so if you aren't looking at your plotted data at the right 'angle', then you may miss some important features of your data.

```
library(scatterplot3d)
x <- cbind(rnorm(30, 0, 1), rnorm(30, 0, 1), rnorm(30, 0, 1))
s3 <- scatterplot3d(x, pch=16)
```



One possibility is to plot your data in a frame that allows you to dynamically rotate the frame, so you can see your data at whatever angle you chose. The library 'rgl' can do this (note, it won't show up here- you'll have to try this on your own system).

```
library(rgl)
open3d()
plot3d(rnorm(10), rnorm(10), rnorm(10), col=1, size=6, pch=16)
```

Lastly, we can increase the dimensionality of our data by changing characteristics of our points according to a 4th or even 5th dimension. For instance, three of our data dimensions can correspond to x, y, and z location on our plot, the 4th dimension can correspond to point size, and the 5th dimension can correspond to the color of our plot, as below. It's in the package 'rgl' and, like plot3d, can rotate! Like plot3d, it won't show up on this pdf, however.

```
spheres3d(rnorm(10), rnorm(10), rnorm(10), radius=runif(10), color=rainbow(10))
```

```
spheres3d(rnorm(10), rnorm(10), rnorm(10), color=rainbow(10))
```

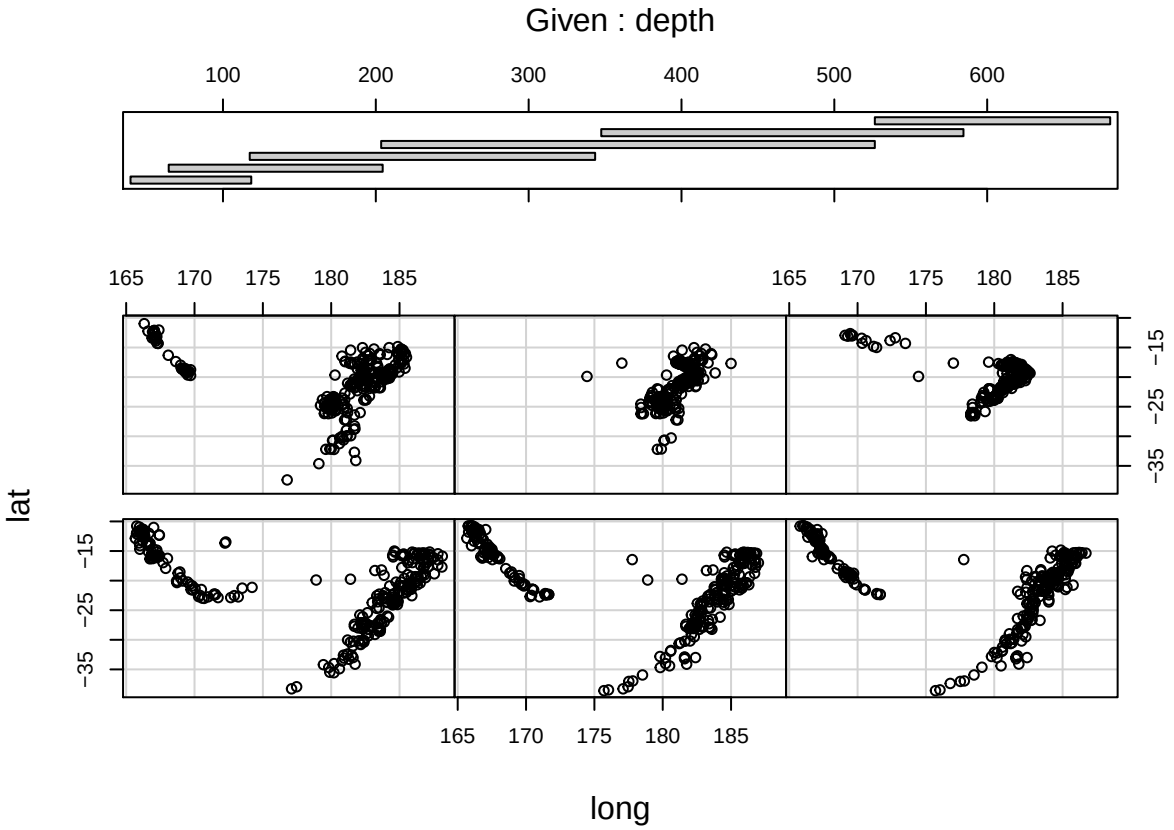
Here we are plotting a 2-d kernel density estimate in 3d.

```
library(MASS)
de.loc <- kde2d(quakes$lat, quakes$long)
open3d(); persp3d(de.loc$x, de.loc$y, de.loc$z)
```

Conditioning

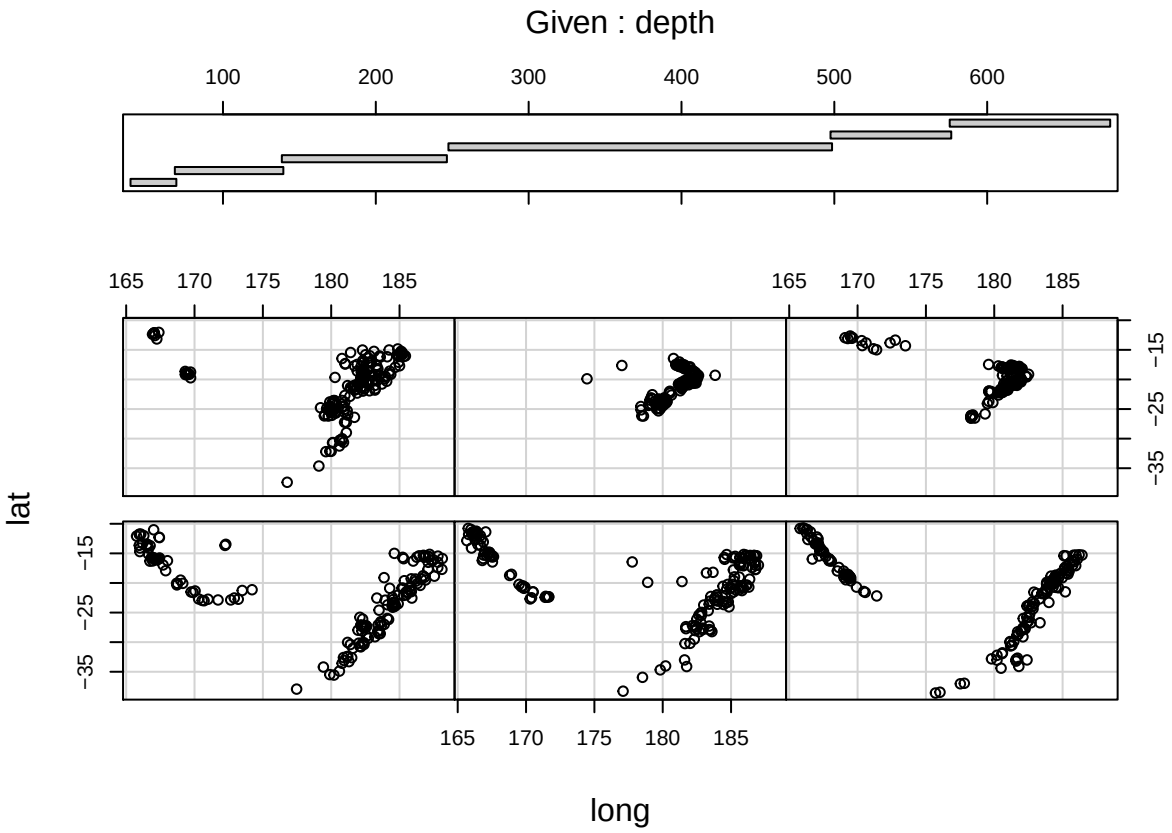
Another way to visualize more than 2-dimensional data is to condition on the 3rd (or 4th) variable. We can do this with a conditioning plot. If our conditioning variable (say Z) is continuous, 'coplot' divides this variable into intervals (the default is 6 OVERLAPPING intervals, based on quantiles), and creates a scatterplot of Y~X for each interval. So for instance, Z (the third variable) is divided into 6 intervals. For the first interval, a scatterplot of Y~X would be created for points whose Z value falls into the first interval. The same is true for all the other intervals.

```
attach(quakes)
coplot(lat~long|depth)
```



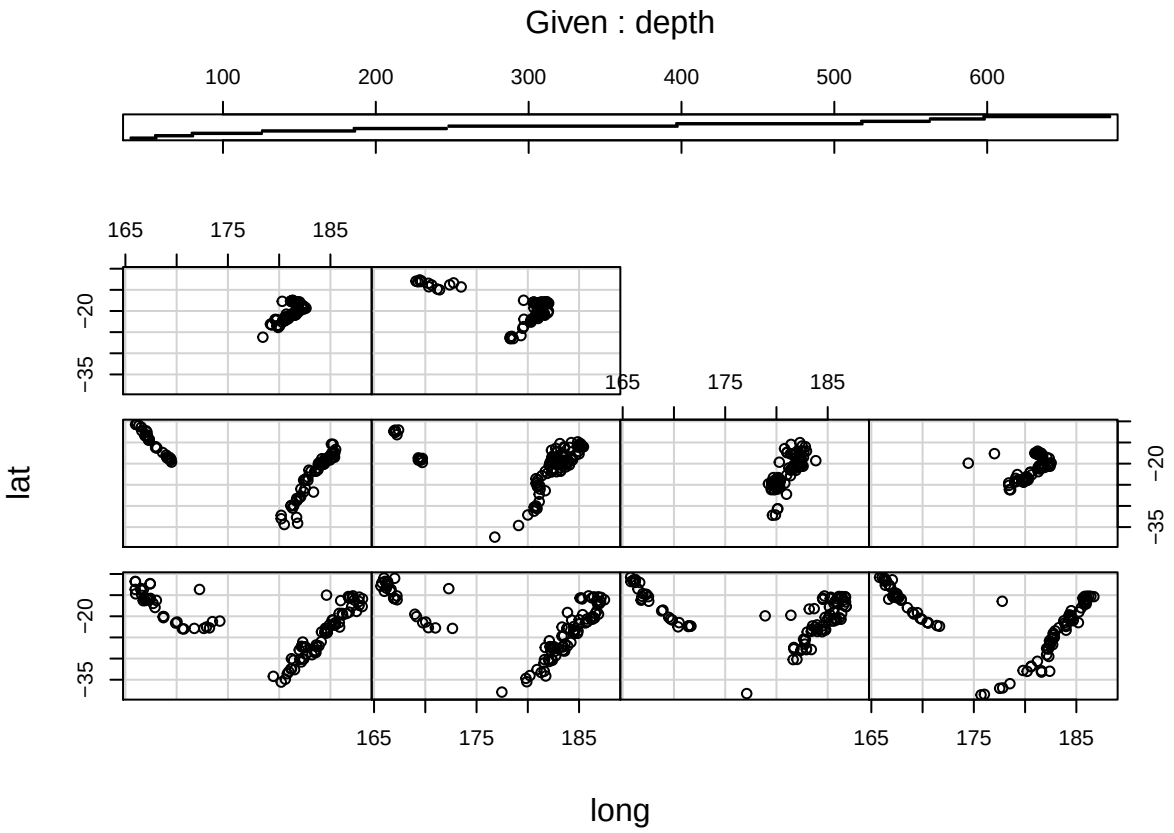
We can specify that we want no overlap in the third variable.

```
coplot(lat~long|depth, overlap=0)
```



And we can specify that, instead of just 6 bins, we want, say, 10 bins.

```
coplot(lat~long|depth, overlap=0, number=10)
```

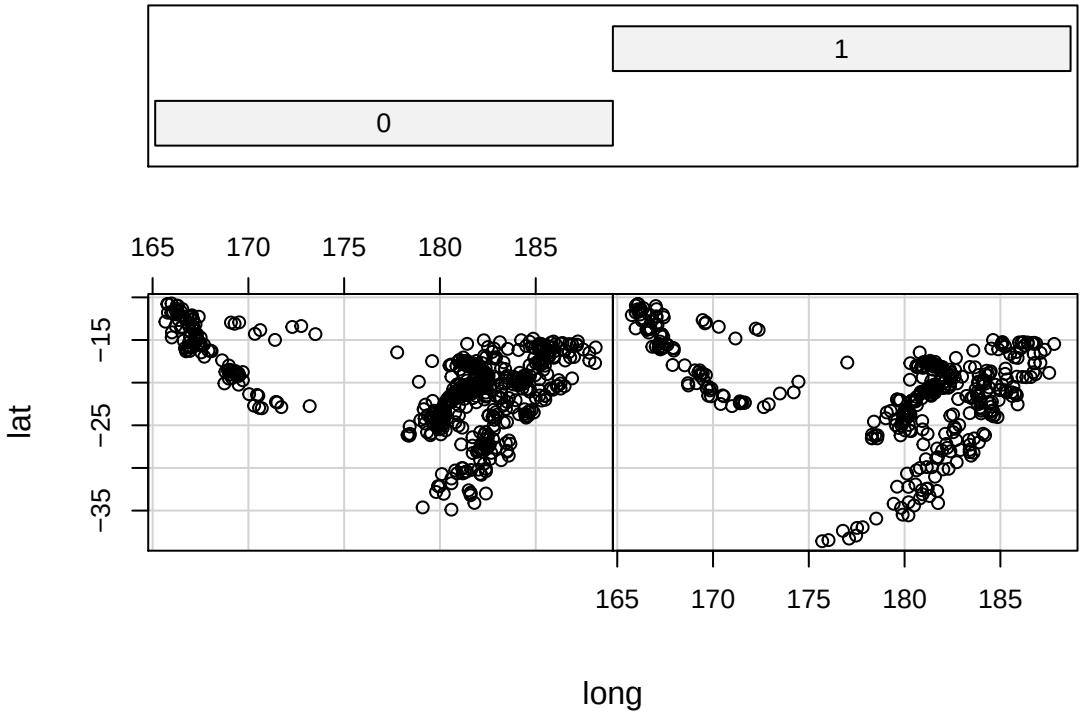


We can condition on a 4th variable as well. For some reason this plot isn't compiling, but it should be obvious if you run it on your own machine.

```
coplot(lat~long|mag*stations)
```

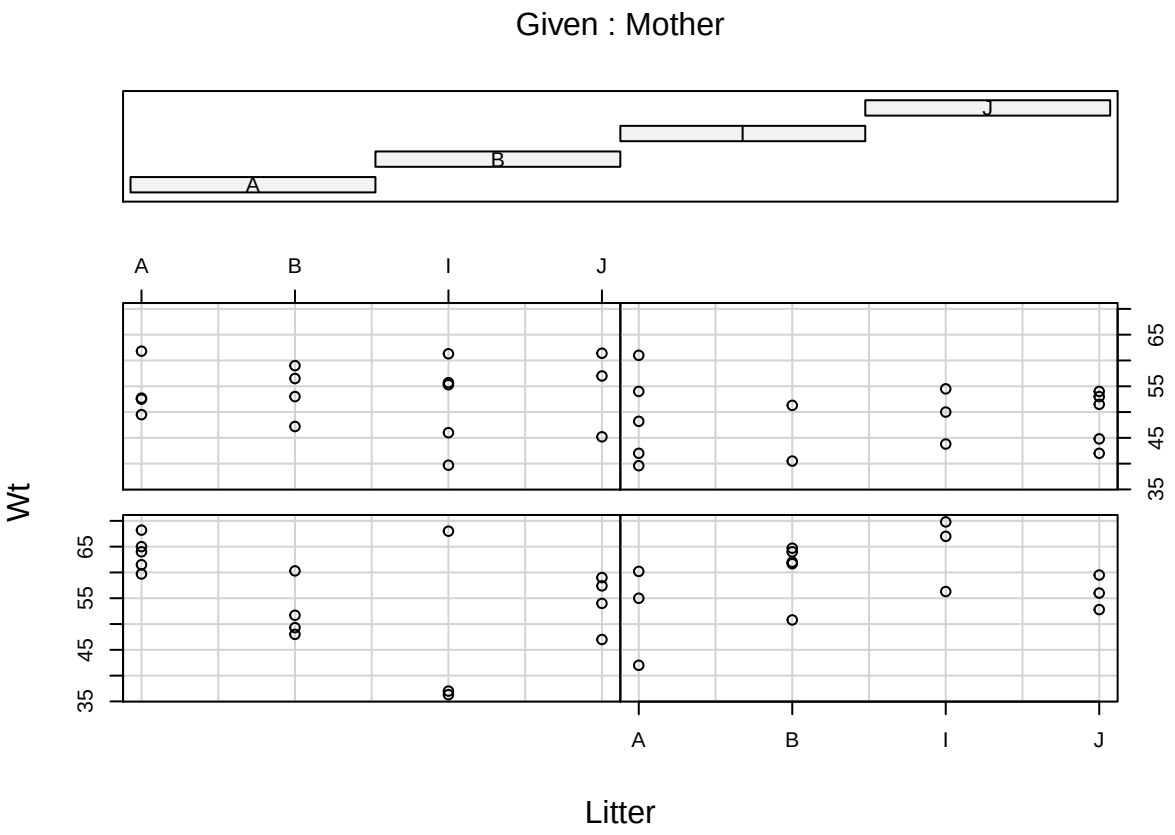
We can condition on a categorical variable

```
stat30 <- ifelse(stations>30, 1, 0)
coplot(lat~long|as.factor(stat30))
```



And, in the formula Y~X, Y and X need not both be continuous:

```
attach(genotype)
coplot(Wt~Litter|Mother)
```

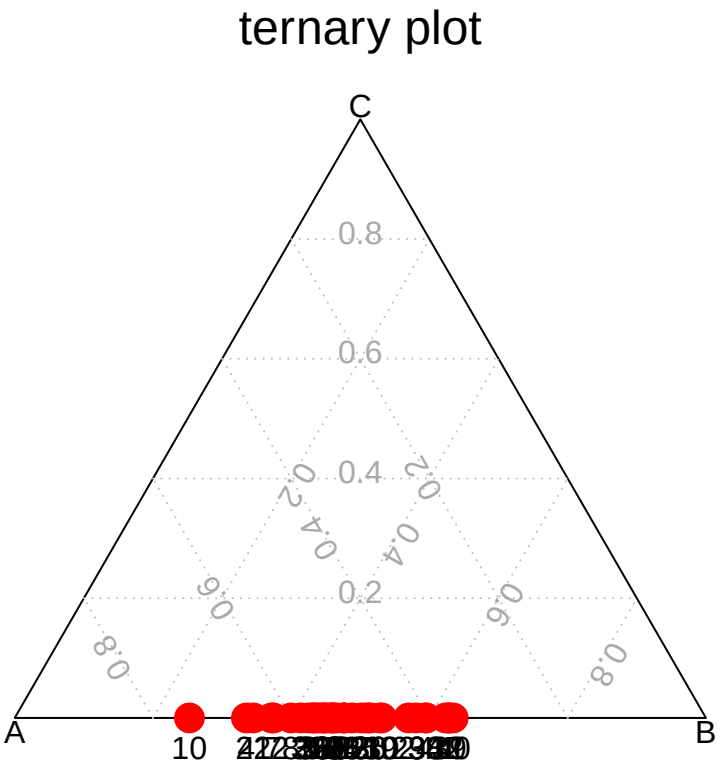


Ternary Plot

A ternary plot is helpful for visualizing composition data. Lifeboats on the Titanic- For each boat, what proportion of people in that boat were men, women, and crew? How much time does each student spend on homework in three subjects (as a proportion of total time spent on homework)? Each student is going to have a slightly different balance of time spent in each subject matter.

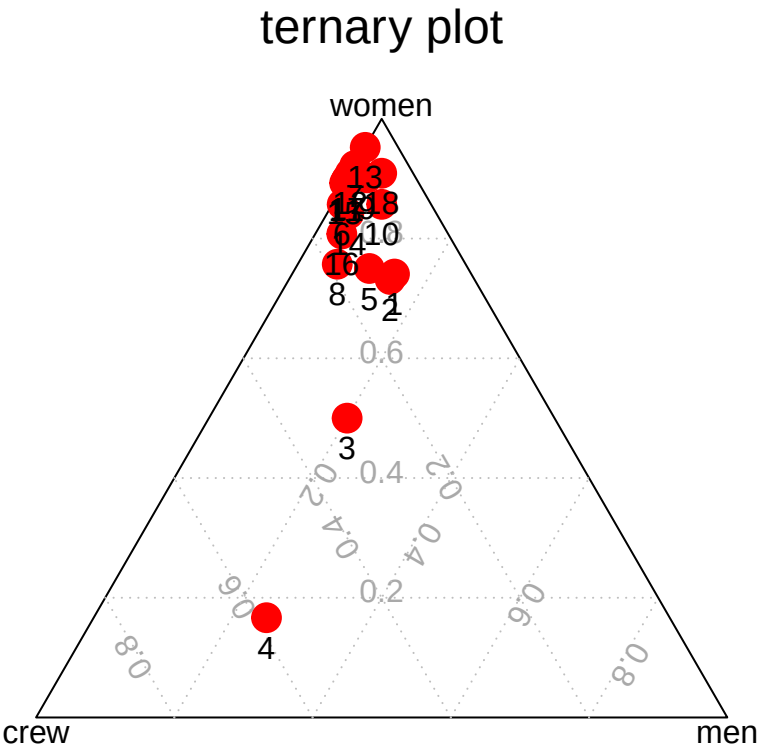
For starters, let's look at fake data, and to build intuition, let's say that the third variable for each observation is always zero. What do we expect the ternary plot to look like?

```
library(vcd)
# Create a fake dataset.  Note the third variable is always zero.
test <- cbind(rnorm(40, 4, 1), rnorm(40, 4, 1), rep(0, 40))
colnames(test) <- c("A", "B", "C")
#Row sums- we want each observation to add to 1.
test.sum <- apply(test, 1, sum)
#And divide each row by its row sum so they add to 1.
test <- test/test.sum

ternaryplot(test, id=seq(1,40))
```



And now let's have a look at the titanic data.

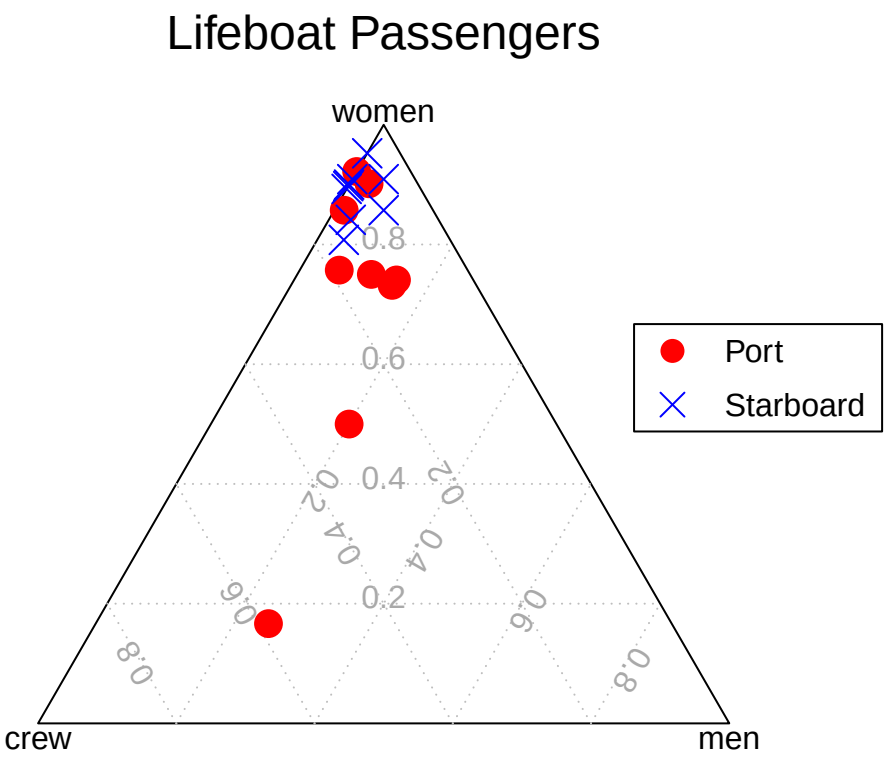
```
ternaryplot(Lifeboats[,4:6], id=seq(1,18))
```



Using similar techniques as before, we can increase the dimensionality of our plotting capabilities by playing around with the characteristics of the points we are plotting. We can add information to our titanic plot by

changing the points to reflect the side of the boat the the lifeboats were on.

```
attach(Lifeboats)
pch.vec <- ifelse(side=="Port", 16, 4)
col.vec <- ifelse(side=="Port", 2, 4)
ternaryplot(Lifeboats[,4:6], col=col.vec, pch=pch.vec, main="Lifeboat Passengers")
grid_legend(0.8, 0.5, pch=c(16, 4), col=c(2,4), labels=c("Port", "Starboard"))
```



Visualizing several variables at once

Pairs plots

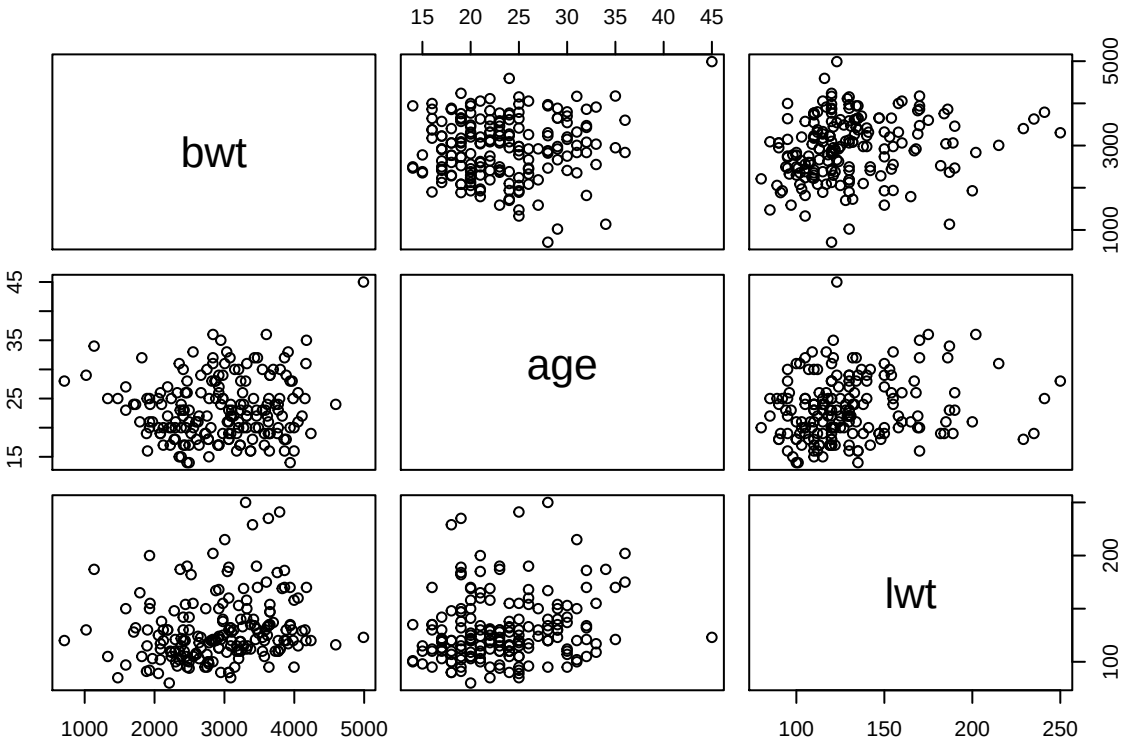
Are variables correlated? Test for significance:

$$t^* = \frac{r\sqrt{n-2}}{\sqrt{1-r^2}} \sim t_{n-2}\left(1-\frac{\alpha}{2}\right)$$

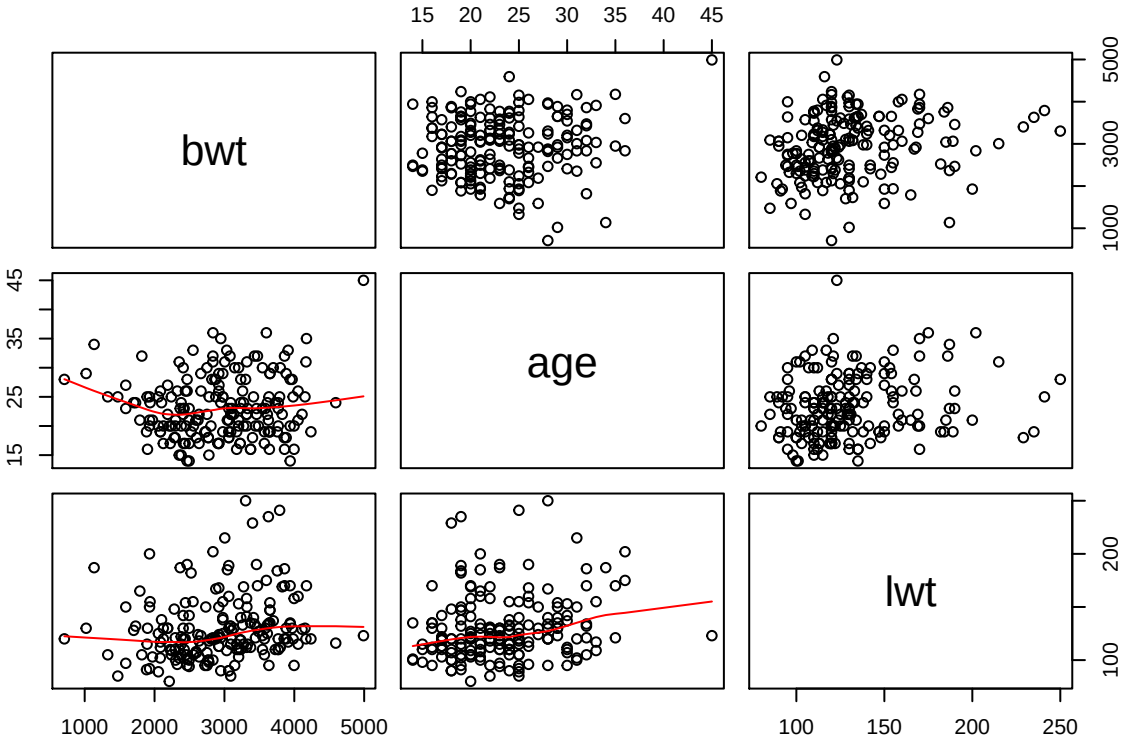
```
library(MASS); attach(birthwt)
```

```
## The following objects are masked from birthwt (pos = 8):
##
##     age, bwt, ftv, ht, low, lwt, ptl, race, smoke, ui
```

```
vars <- cbind(bwt, age, lwt)
pairs(vars)
```




```
pairs(vars, lower.panel=panel.smooth)
```



Visualizing density

-Curve over 2-d space -Height of curve indicates frequency/density level

Kernel estimation

Two options

-dims are independent

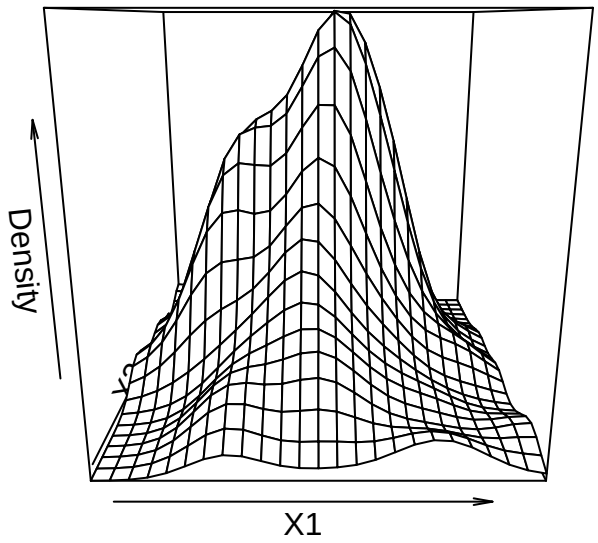
$$\hat{f}(x) = \frac{1}{nh_1h_2}\sum_{i=1}^{n} K\left(\frac{\bar{x}_1 - x_{i1}}{h_1}\right) K\left(\frac{\bar{x}_2 - x_{i2}}{h_2}\right)$$

-dims are dependent

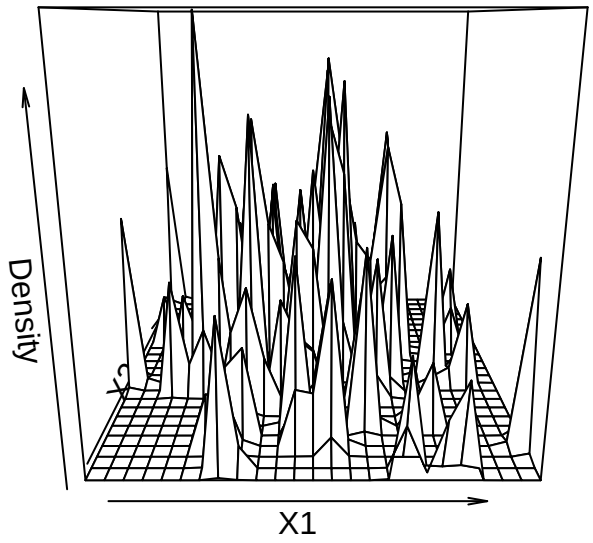
$$\hat{f}(x) = \frac{1}{n|H|}\sum_{i=1}^{n} K\left(H^{-1}(\bar{X} - X_i)\right)$$

H is a bandwidth matrix.

```
x1 <- rnorm(100, 0, 1); x2 <- rnorm(100, 0, 1)
d.e2 <- kde2d(x1, x2)
persp(d.e2, xlab="X1", ylab="X2", zlab="Density")
```



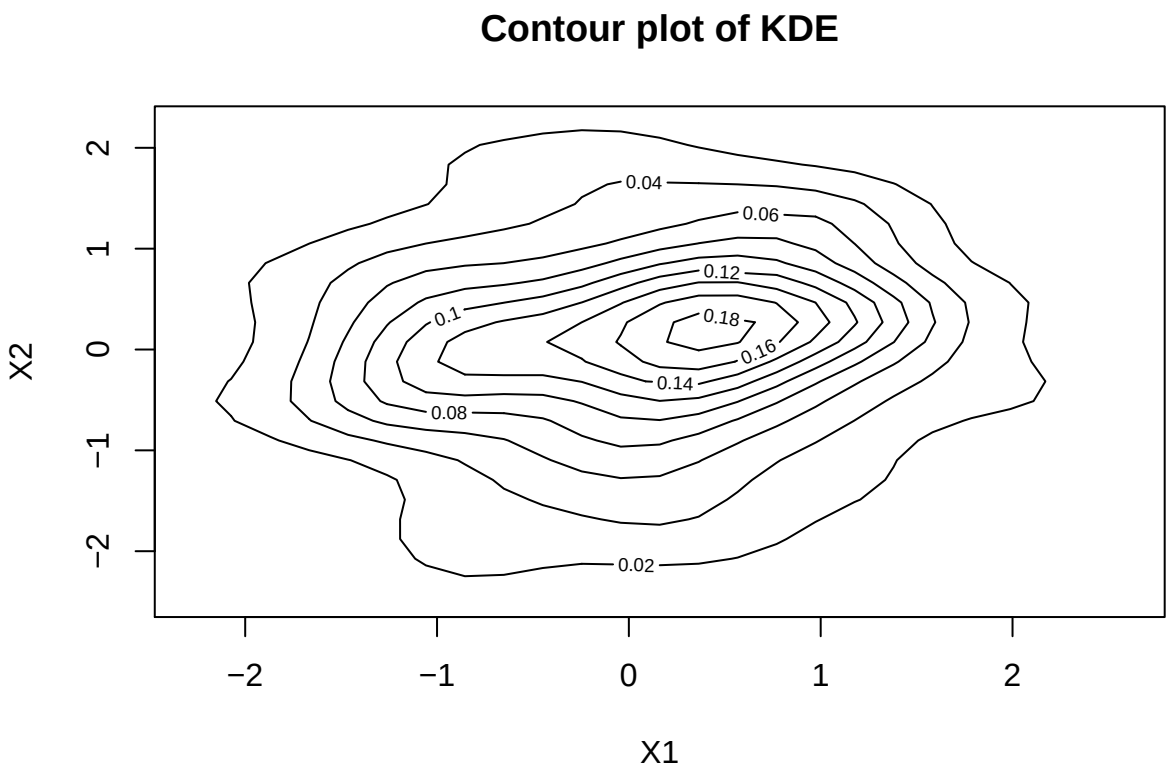

```
d.e2b <- kde2d(x1, x2, h=c(.3, .3))
persp(d.e2b, xlab="X1", ylab="X2", zlab="Density")
```



Level Sets

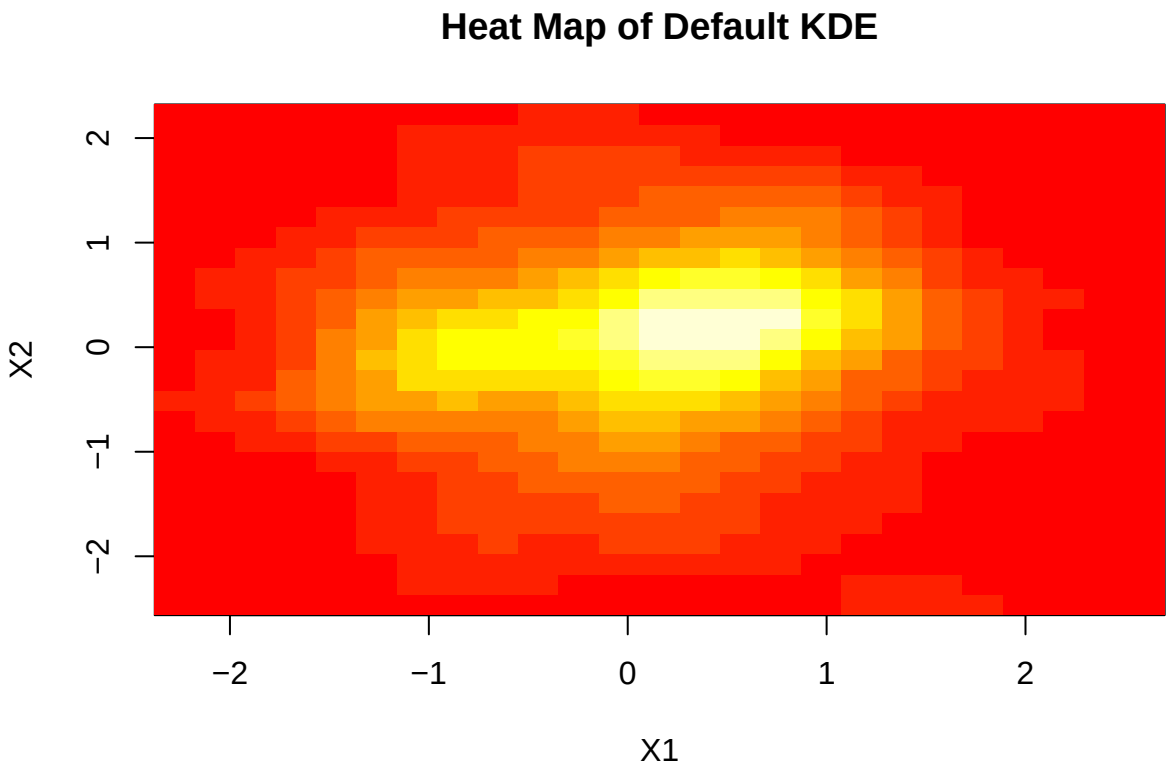
$$L(\lambda, f(x)) = \{x : f(x) > \lambda\}$$

```
contour(d.e2, xlab="X1", ylab="X2", main="Contour plot of KDE")
```



Heat map

```
image(d.e2, xlab="X1", ylab="X2", main="Heat Map of Default KDE")
```

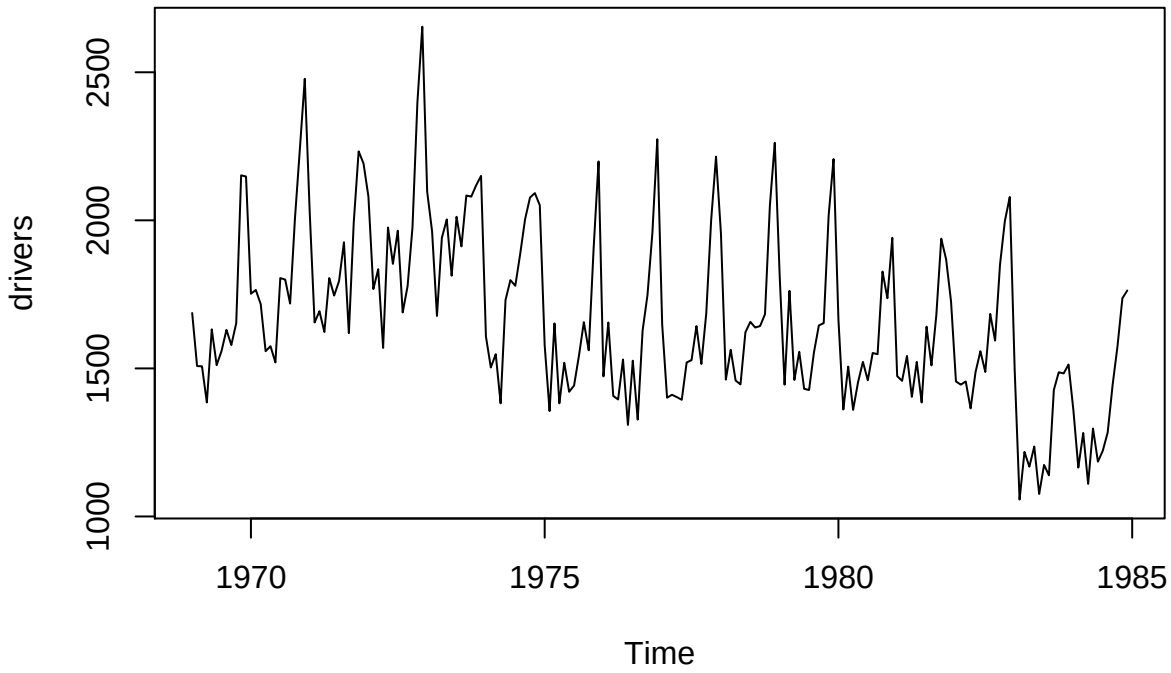


Time Series

Time series are a special kind of data. Each observation has both a value and a time associated with it. If we collect data about driver death and casualties each month over a span of several years, it's not just a collection of IID data- it matters WHEN we collected each data point. We are interested in trends over time- are values in our dataset generally increasing over time? Decreasing? Following some curve? Is there seasonality (cycles) in our data?

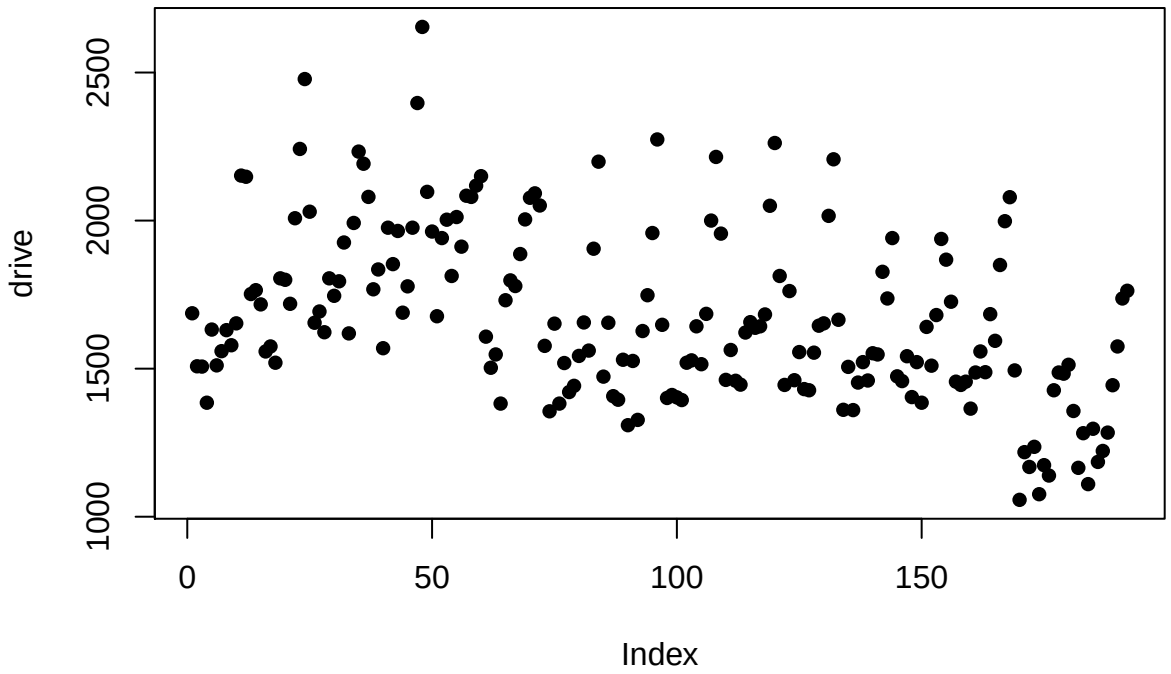
Here is a plot of driver deaths in the UK. UKDriverDeaths is a special object in R called a 'ts' (it's a time series...).

```
#drivers is a time series object
drivers <- UKDriverDeaths
plot(drivers)
```



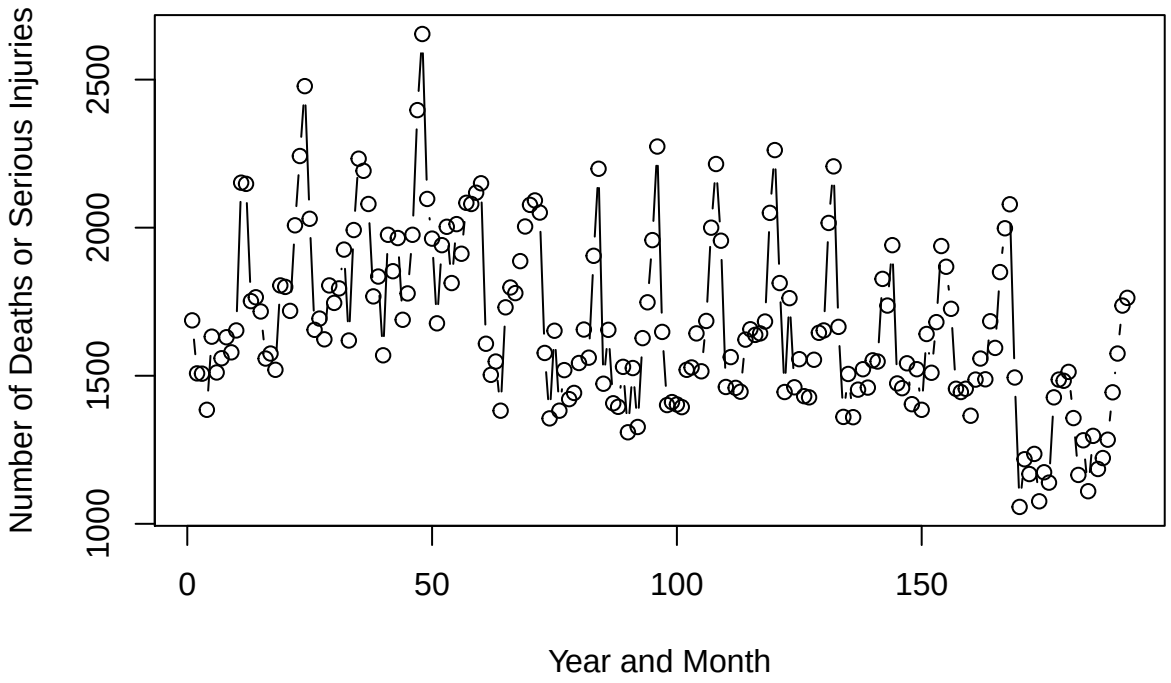
But having this special object isn't necessary. We could just put our data into a regular old vector and plot it as well.

```
drive <- as.vector(drivers)
plot(drive, pch=16)
```



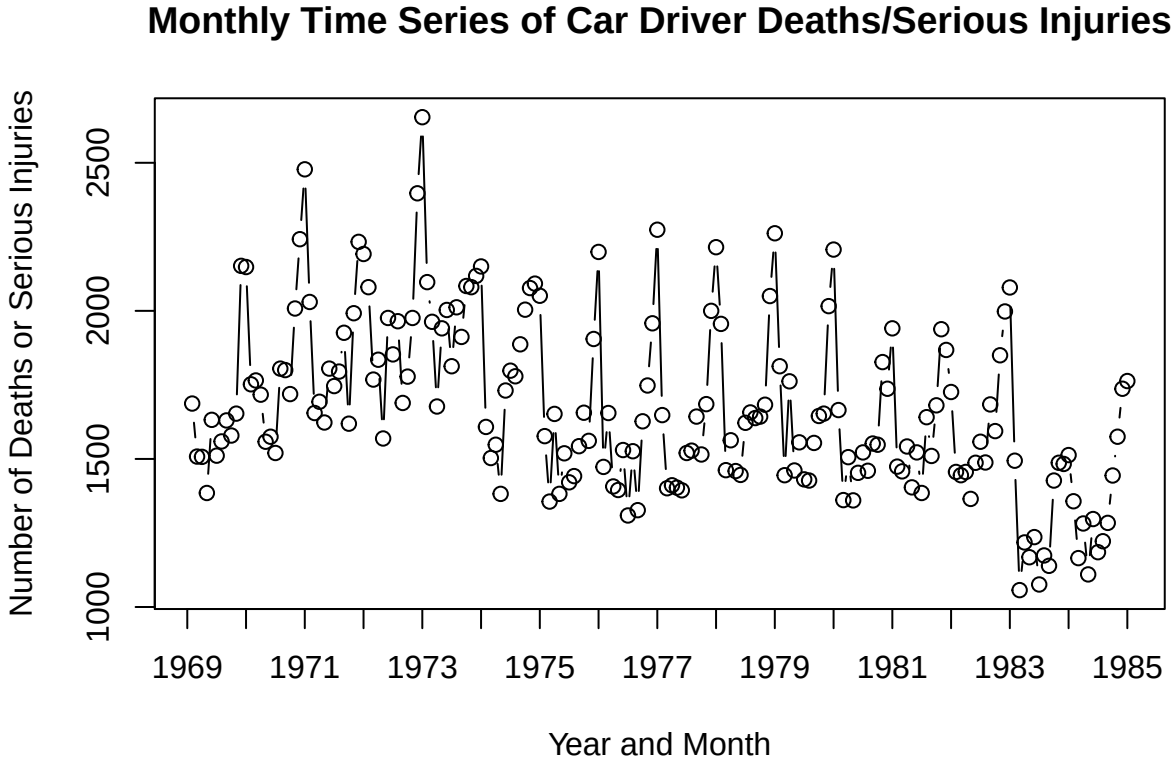
Of course, we want to make the progression from one point to the next more apparent with lines:

```
plot(drive, type="b", ylab="Number of Deaths or Serious Injuries", xlab="Year and Month")
```



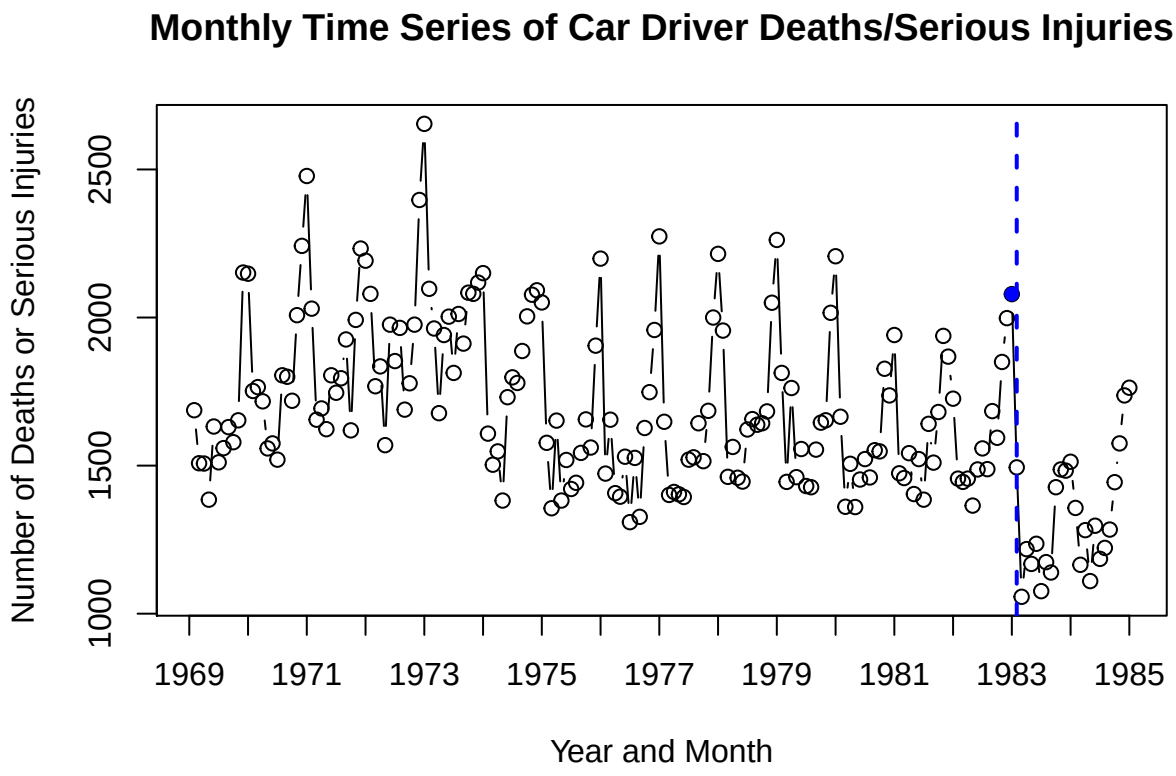
And we can modify our x-axis to reflect time.

```
plot(drive, type="b", ylab="Number of Deaths or Serious Injuries", xlab="Year and Month", xaxt="n")
axis(1, at=seq(0, 192, by=12), labels=c("1969", "1970", "1971", "1972", "1973", "1974", "1975", "1976",
title("Monthly Time Series of Car Driver Deaths/Serious Injuries")
```



In this time series, a seatbelt law went into effect on January 31, 1983. We can see a change in the time series starting in February of the same year.

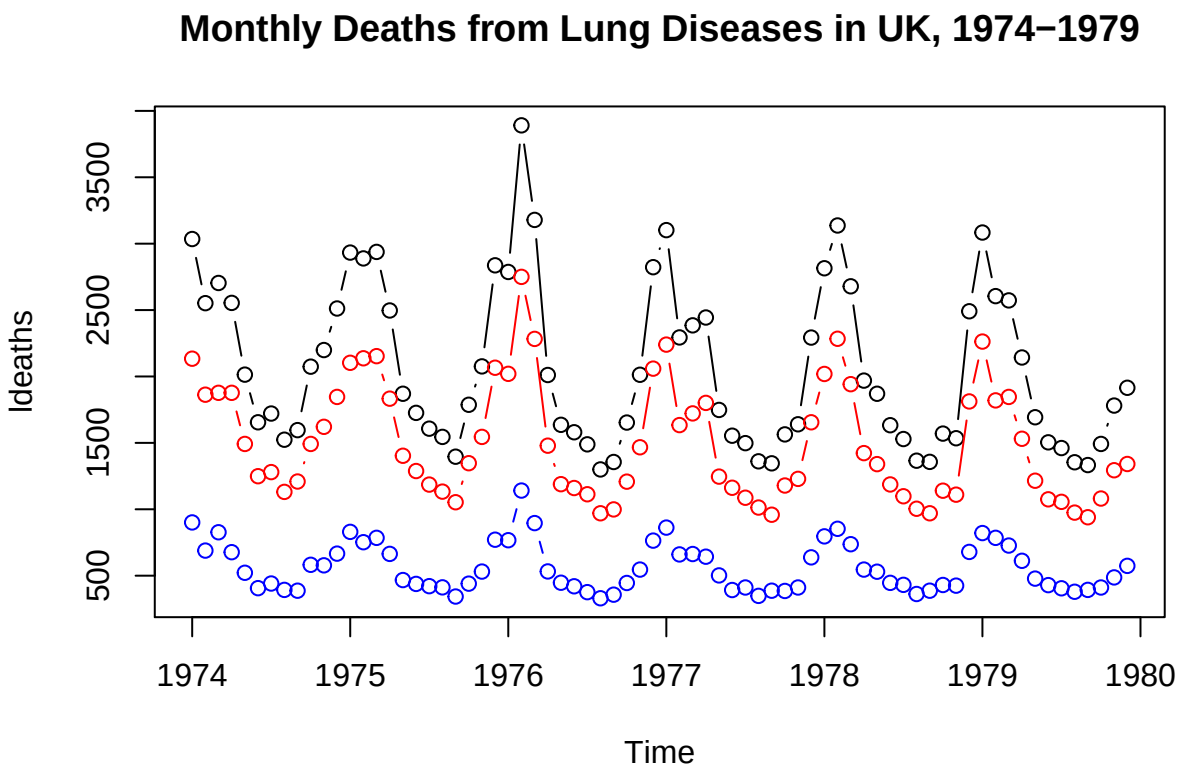
```
#Mandatory seatbelt law, Jan 31 1983
plot(drive, type="b", ylab="Number of Deaths or Serious Injuries", xlab="Year and Month", xaxt="n")
axis(1, at=seq(0, 192, by=12), labels=c("1969", "1970", "1971", "1972", "1973", "1974", "1975", "1976",
title("Monthly Time Series of Car Driver Deaths/Serious Injuries")
seatbelt <- 12*14+1; abline(v=seatbelt, col=4, lty=2, lwd=2)
#text(locator(1), "Mandatory Seatbelt Law", col=4)
points(seatbelt-1, drive[seatbelt-1], col=4, pch=16)
```




```
#text(locator(1), "Mandatory Seatbelt Law: Jan 31, 1983", col=4)
```

We can also condition on additional information. For instance, take monthly deaths from lung disease in the UK. We can consier the overall trend (the black line in the plot below), but maybe we believe that death from lung disease has a different trend for men and women. How are the trends different? How are they similar?

```
all.deaths <- c(ldeaths, mdeaths, fdeaths)
plot(ldeaths, type="b", ylim=c(min(all.deaths), max(all.deaths)))
lines(mdeaths, type="b", col=2)
lines(fdeaths, type="b", col=4)
title("Monthly Deaths from Lung Diseases in UK, 1974-1979")
```




```
#legend(locator(1), c("All", "Male", "Female"), col=c(1, 2, 4), lwd=1)
```

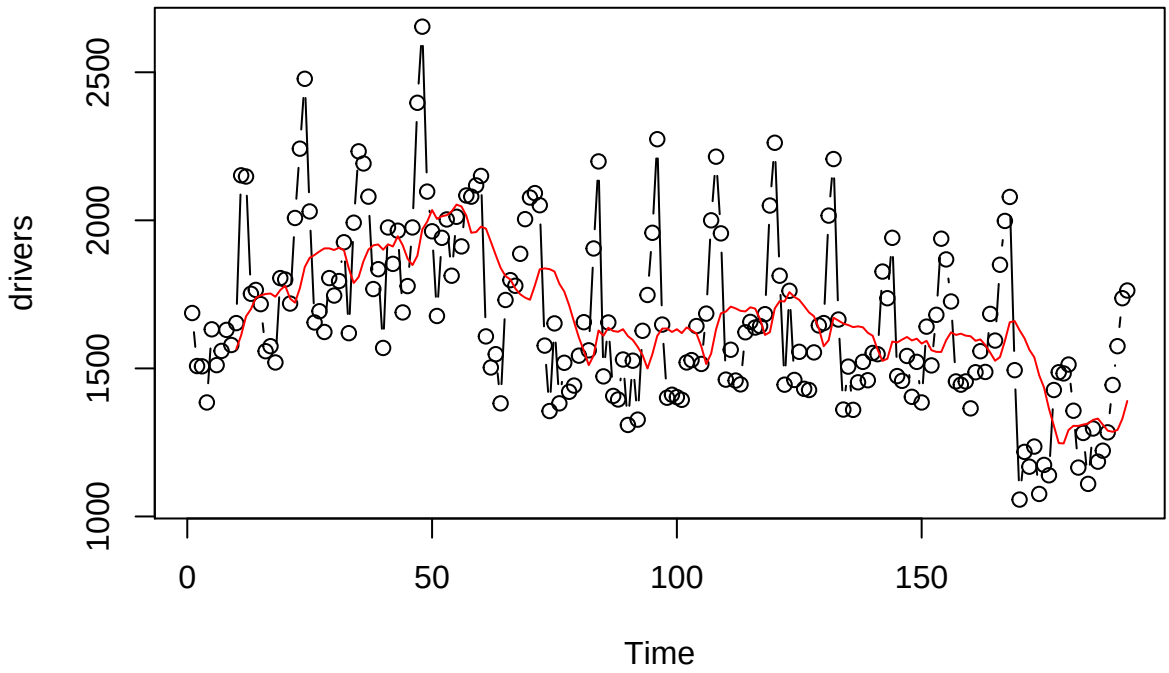
A caveat here is that the time points must line up! If women deaths were sampled once a month for a year, and men's deaths were sampled once every two months for a year, you must make sure to stretch the plot for men over the same span of time as the plot for women.

Likewise, if women were sampled once a month for a year, and men were sampled once every other month for 2 years, even though they have the same number of samples, it wouldn't make sense for those lines to be the same size. The women's line would have to be half as long.

Visualizing trend over time

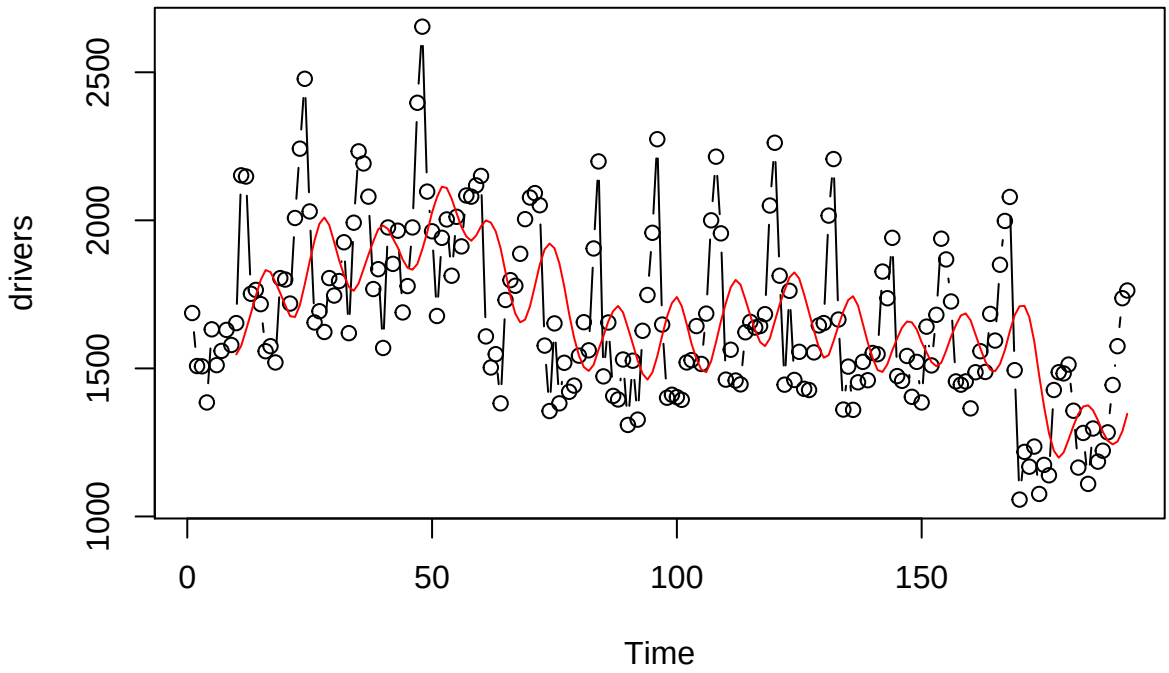
Of course, it would be helpful if we could visualize the actual trend over time. We can do so by plotting the moving average. To build our intuition, we can think about the simple moving average. The simple moving average at time t for a given window w is simply the average of most recent w values (including the one at time t). That's it! Here it is visually:

```
plot(drive, type="b", ylab="drivers", xlab="Time")
n <- length(drive)
MA <- rep(NA, n)
window <- 10
for(i in window:n){
  MA[i] <- mean(drive[(i-window+1):i])
}
lines(MA, col=2)
```



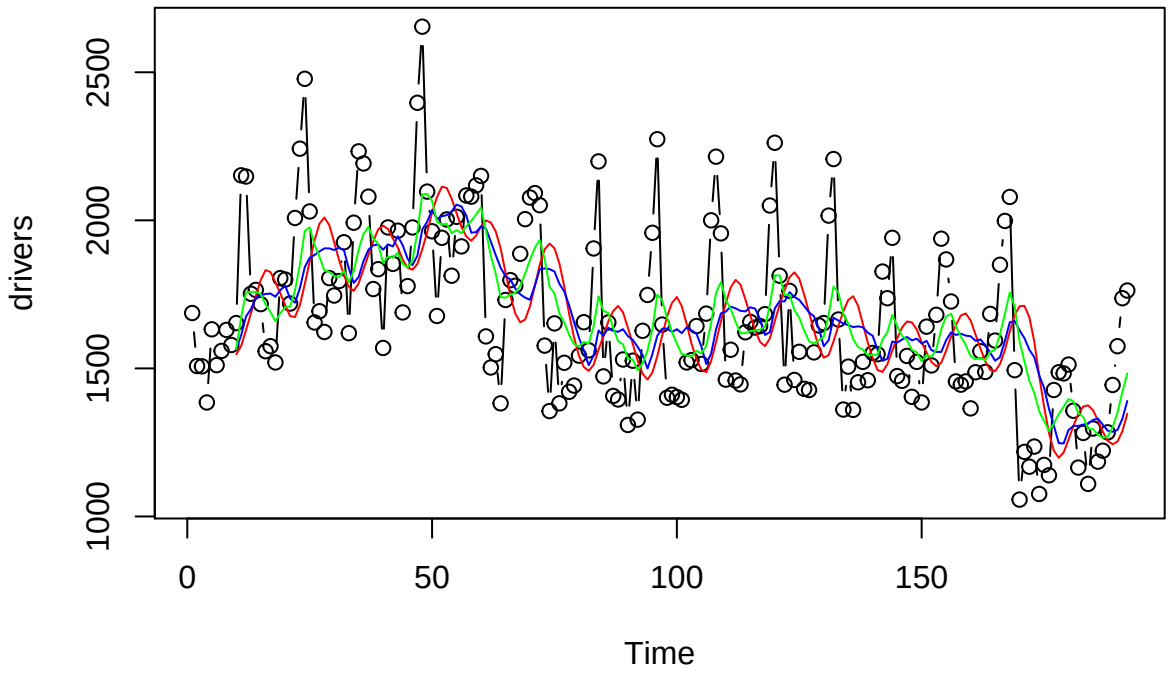
We can think of a plain, boring, average of n points as simply a weighted sum of data points, where each point is weighted by the same value, $\frac{1}{n}$. But we don't have to weight each point the same! We can create a moving average that weights the most recent w points differently. Here's an example. Think about why we may want to do this.

```
plot(drive, type="b", ylab="drivers", xlab="Time")
weight <- c(1:5, 5:1)/sum(c(1:5, 5:1)) # this is the weightings we will use
MA.wt <- rep(NA, n)
window <- 10
# Now go through the entire vector in a for loop
for(i in window:n){
  MA.wt[i] <- weight%*%drive[(i-window+1):i]
}
lines(MA.wt, col=2)
```



'TTR' is a library that makes fitting simple moving averages and exponential moving averages easily. Exponential moving averages places a high weight on the most recent value.

```
library(TTR)
MA.wt2 <- SMA(drive, n=10) # simple moving average

MA.wt3 <- EMA(drive, n=10) #exponential moving average.

plot(drive, type="b", ylab="drivers", xlab="Time")
lines(MA.wt, col=2)
lines(MA.wt2, col=4)
lines(MA.wt3, col='green')
```

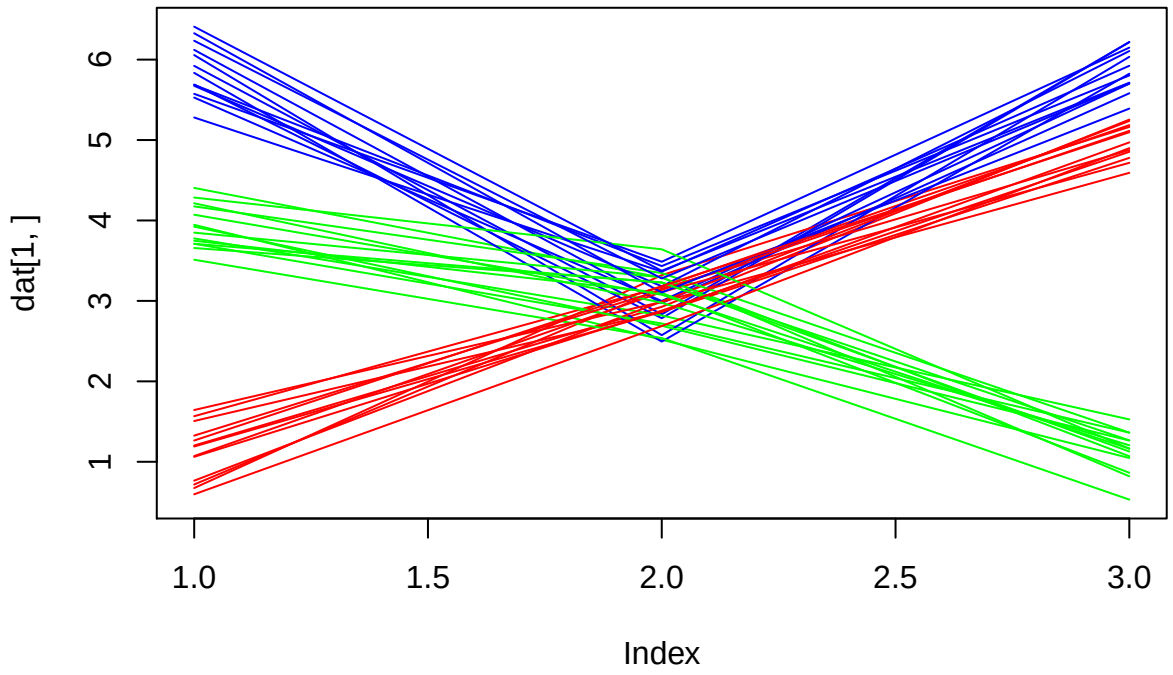


Spaghetti Plots

So far we have focused on a single time series that don't have replications (can't go back in time and observe a new draw from a time series without the help of Dr Who and maybe a Dalak or two). But with medical data, say, we often have multiple time series. Think patients in a weight loss trial. Some patients are given no drug, some are given the drug at low doses, others are given the drug at higher doses. Are there trends in the weight loss of patients over time? Are those trends different given the different dosages of drugs? We can use spaghetti plots to visualize this data.

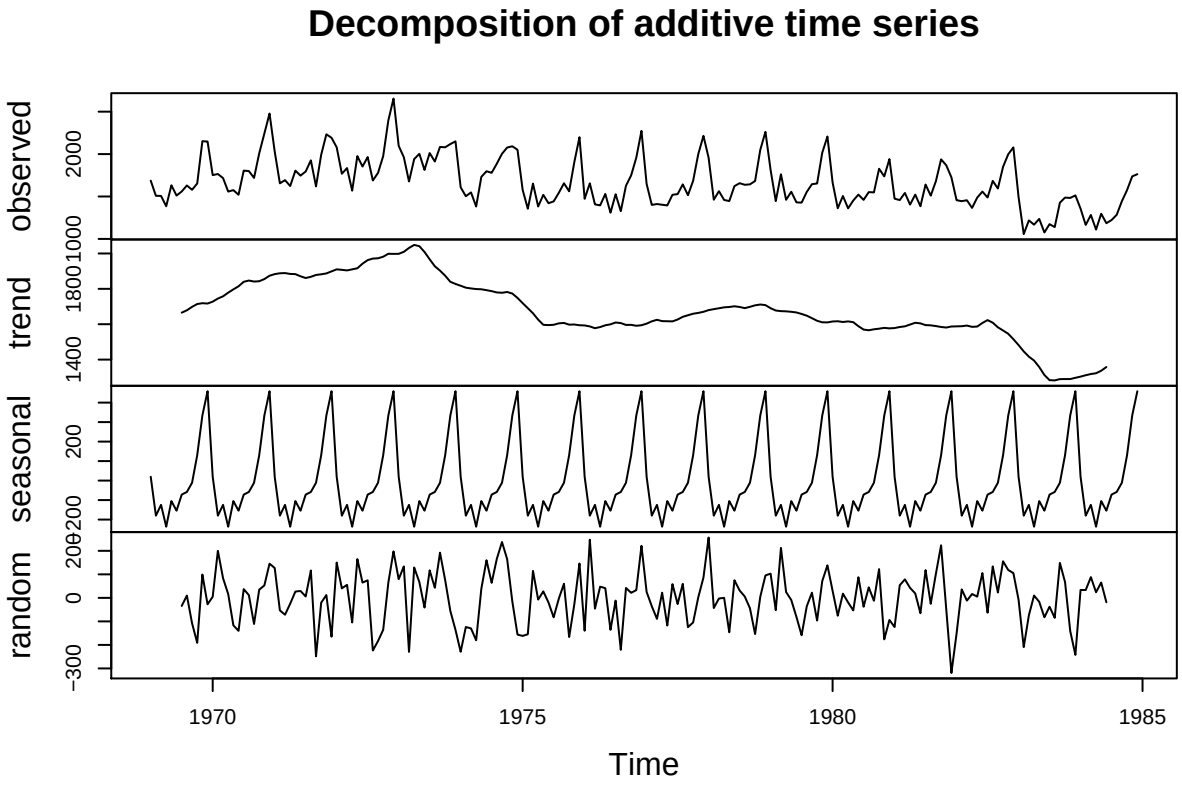
The main thing here is we first have to plot patient 1, setting the ylim of our plots so that all patients' y limits are accommodated (the first patient may not have a large range in their values, but our plot must accommodate for all possible patient ranges). Then we loop over the data one by one adding a line for each patient (and we can color them given the patient category)

```
#This is fake data, so let's make really distinctive trends
trendmat <- matrix(c(1, 3, 5, 4, 3, 1, 6, 3, 6), nrow=3, byrow=T)

#Assigne each patient to a category
subj <- sample(3, 40, replace=T)

# fake data- each patient has a trend plus some noise.
dat <- trendmat[subj, ] + matrix(rnorm(40*3, sd=.3), nrow=40)
color.use <- c("red", "green", "blue") #these are the colors we will use

#plot the first patient, but set the ylim to accomodate all patients
plot(dat[1,], type="l", col=color.use[subj[1]], ylim=c(min(dat), max(dat)))
for(i in 2:40){
  lines(dat[i,], col=color.use[subj[i]])
}
```

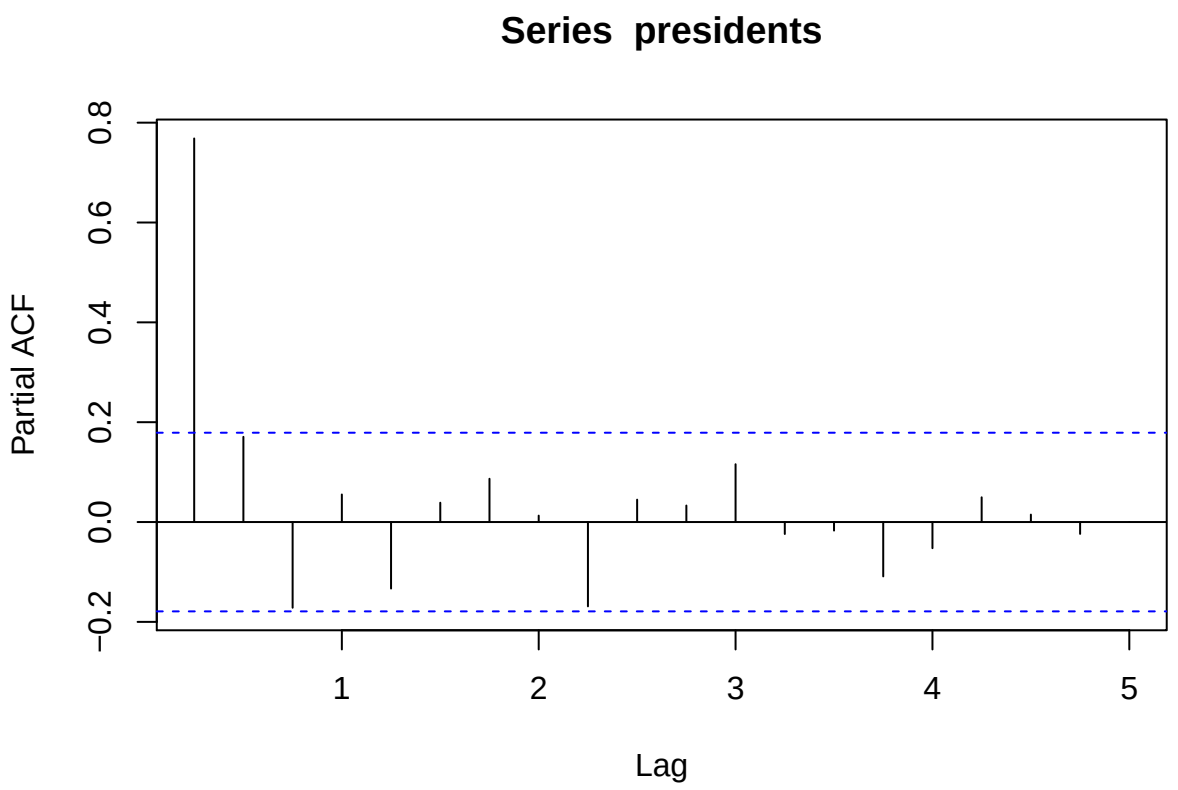


Time Series diagnostics

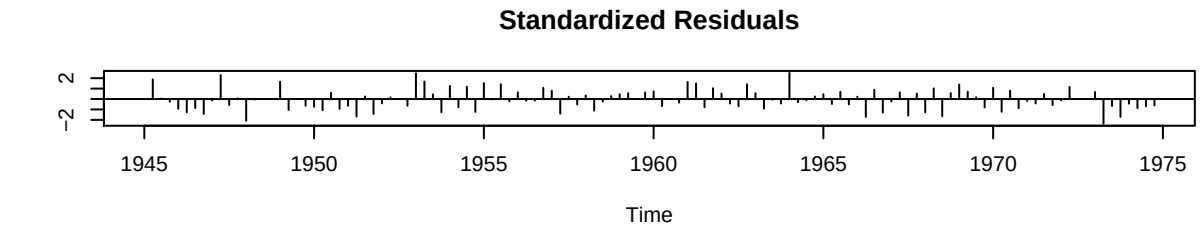
```
dri <- decompose(UKDriverDeaths)
plot(dri)
```

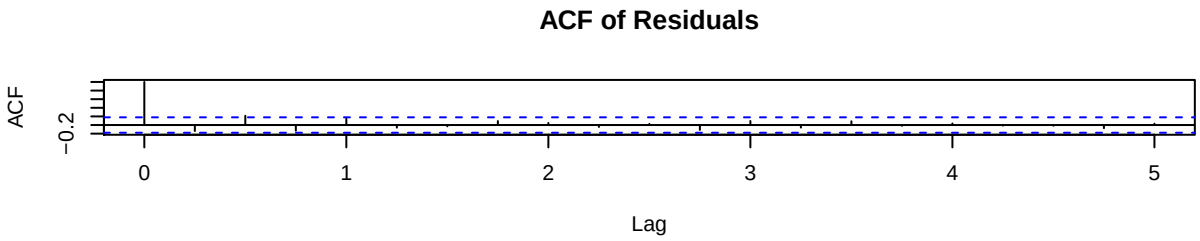


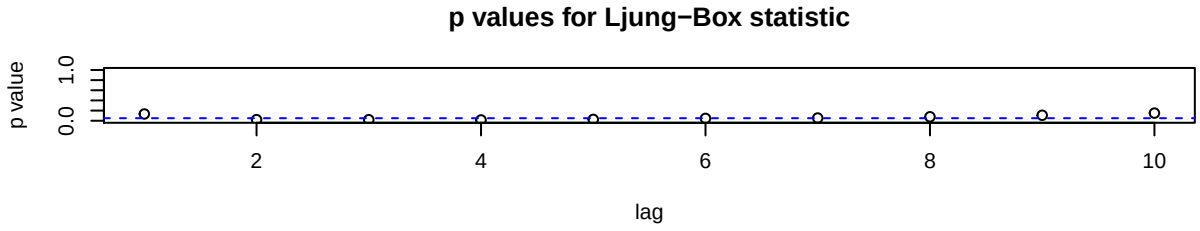
```
pacf(presidents, na.action=na.pass)
```



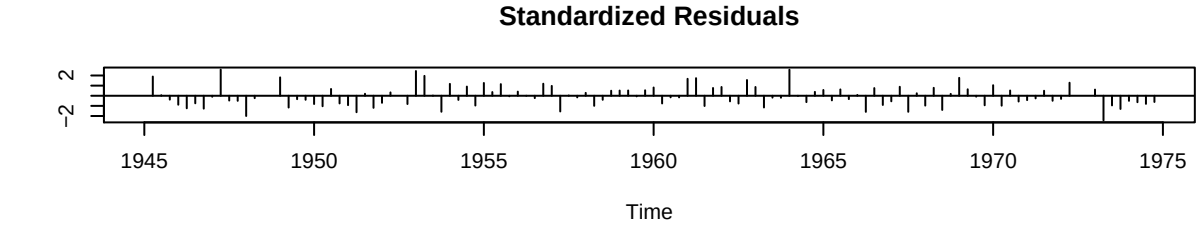

```
f1 <- arima(presidents, order=c(1,0,0))
tsdiag(f1)
```

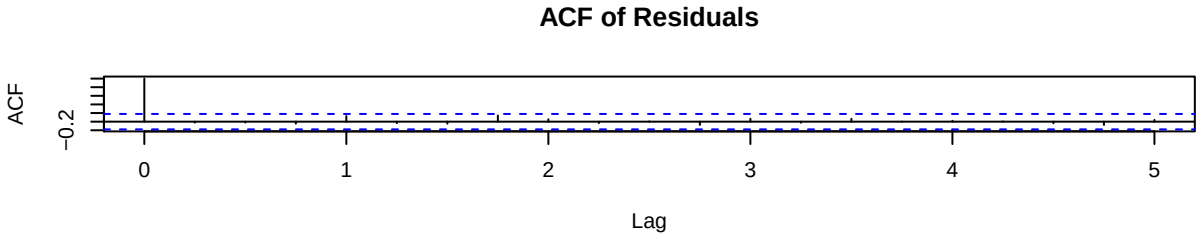


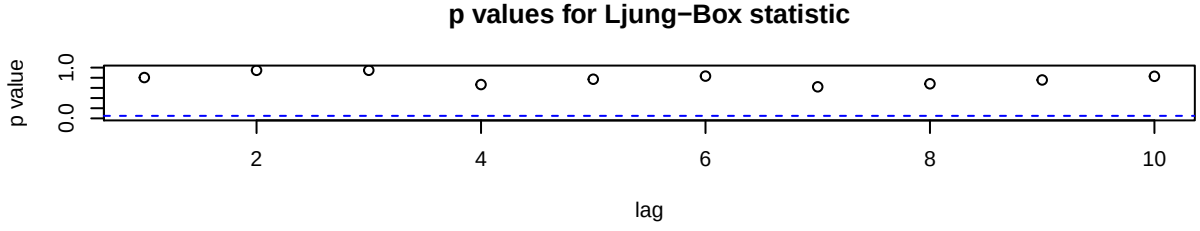





```
f2 <- arima(presidents, order=c(3,0,0))
tsdiag(f2)
```

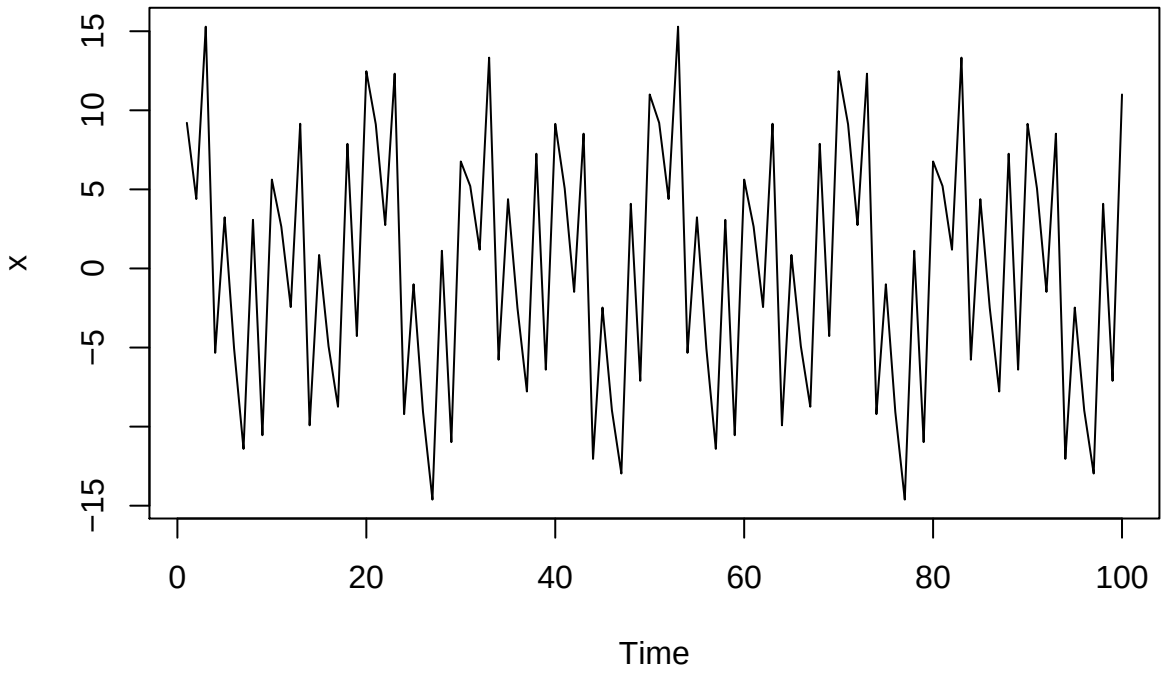




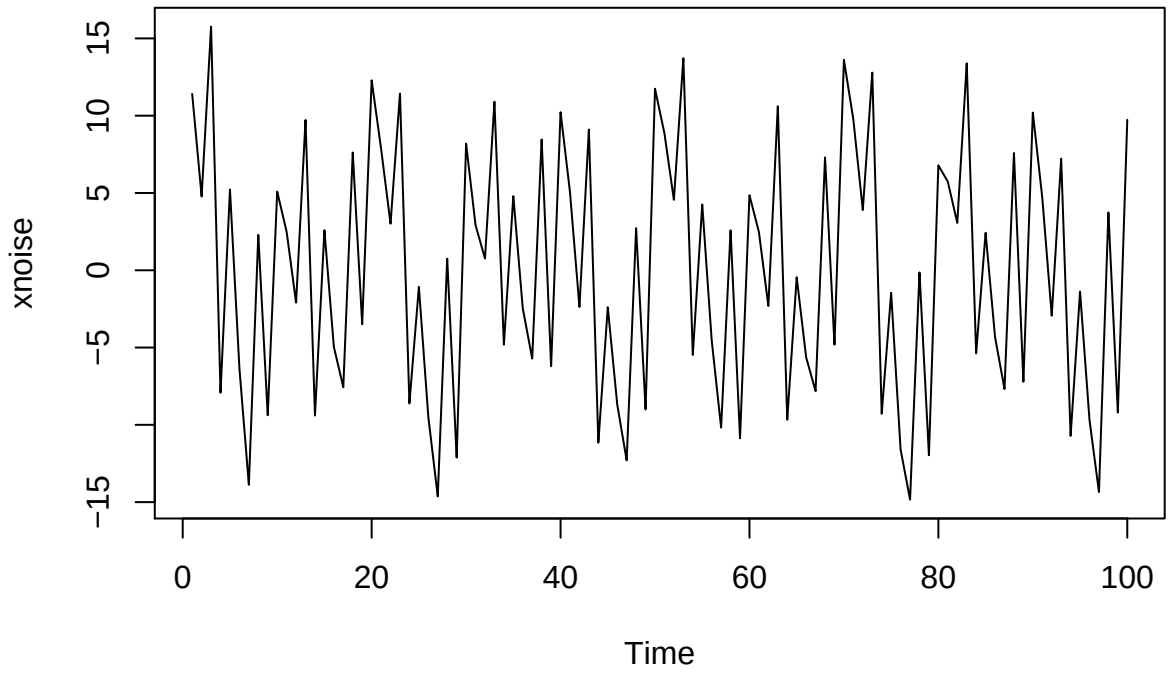


Some Spectral Basics (and I mean basics)

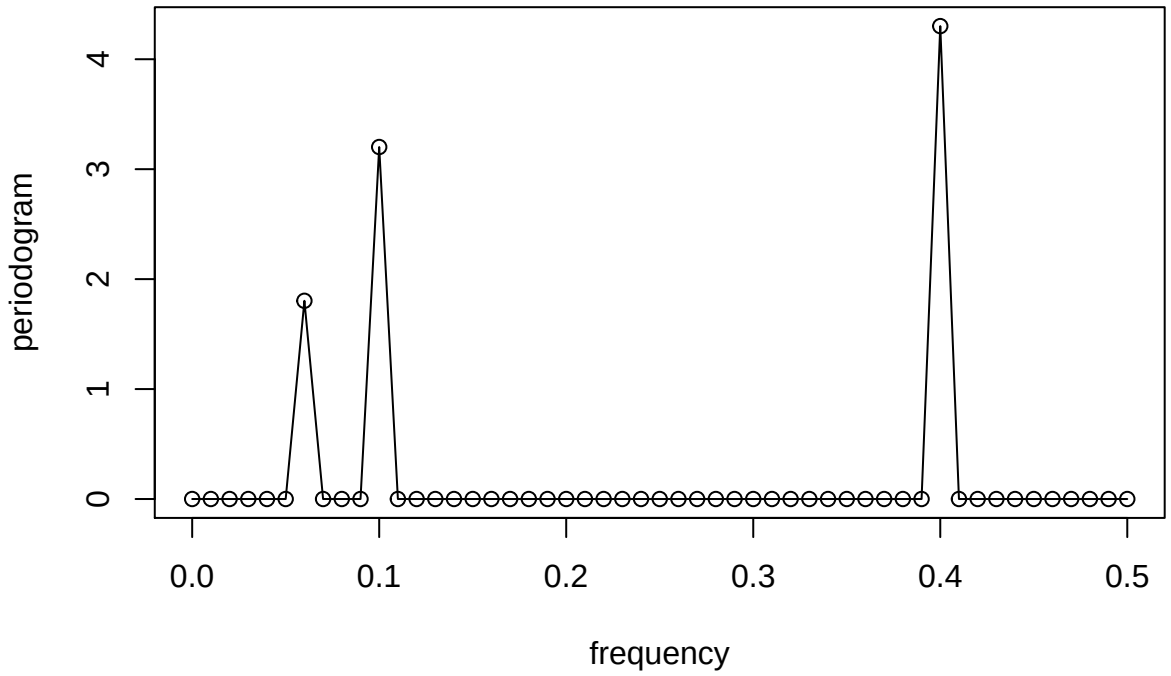
```
t=1:100
x1=2*cos(2*pi*t*6/100)+3*sin(2*pi*t*6/100)
x2=4*cos(2*pi*t*10/100)+5*sin(2*pi*t*10/100)
x3=5*cos(2*pi*t*40/100)+7*sin(2*pi*t*40/100)
x=x1+x2+x3
plot.ts(x)
```



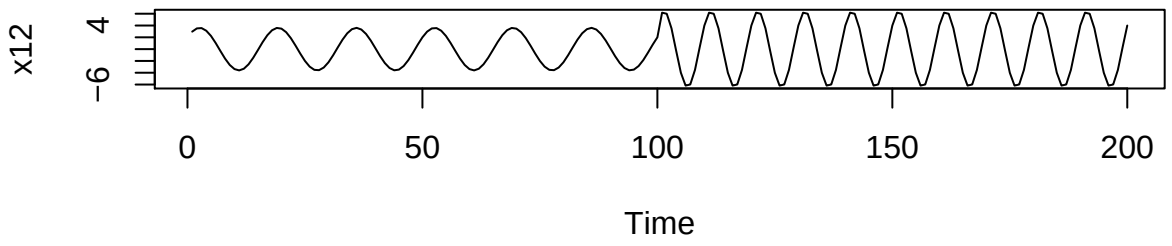

```
xnoise=x+rnorm(100)
plot.ts(xnoise)
```

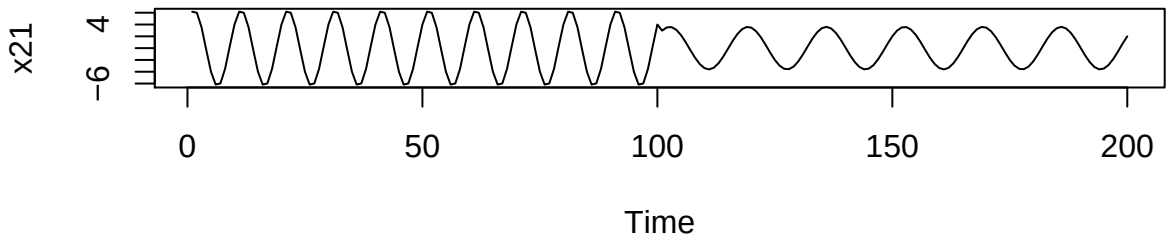



```
P=abs(fft(x)/100)
f=0:50/100
plot(f, P[1:51], type="o", xlab="frequency", ylab="periodogram")
```

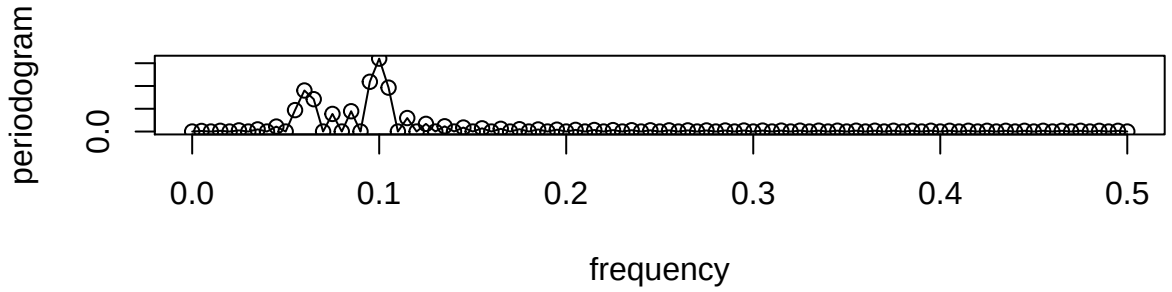


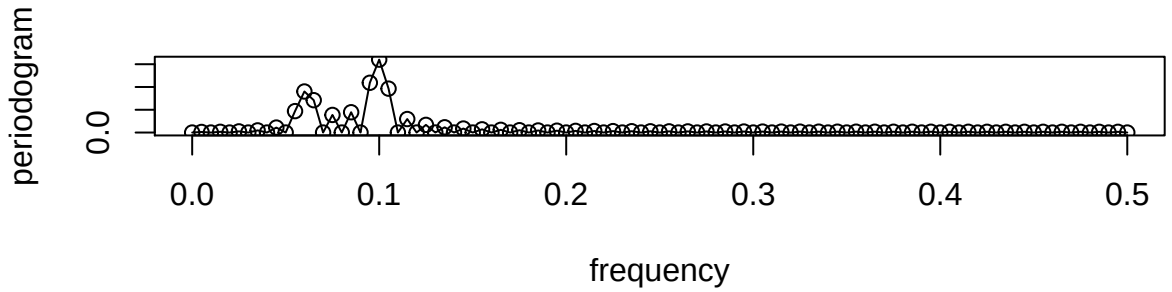

```
x12=c(x1, x2)
x21=c(x2, x1)
par(mfrow=c(2,1))
plot.ts(x12)
plot.ts(x21)
```






```
par(mfrow=c(2,1))
P=abs(fft(x12)/200)
f=0:100/200
plot(f, P[1:101], type="o", xlab="frequency", ylab="periodogram")

P2=abs(fft(x21)/200)
plot(f, P2[1:101], type="o", xlab="frequency", ylab="periodogram")
```



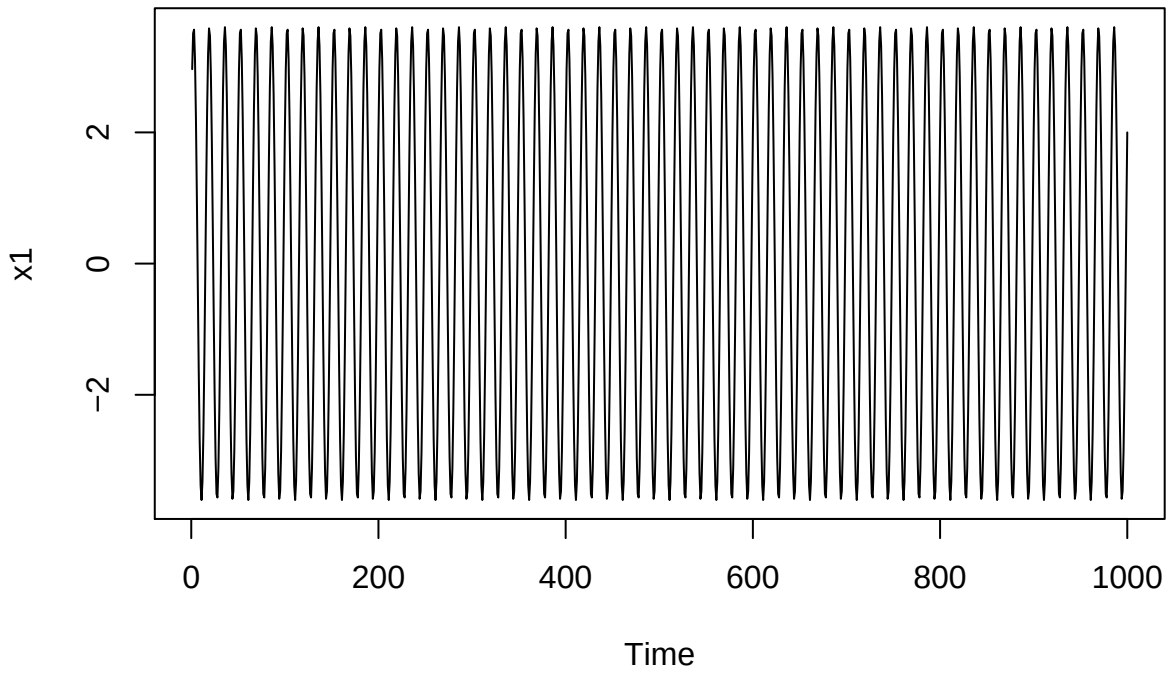


```
library(signal)
```

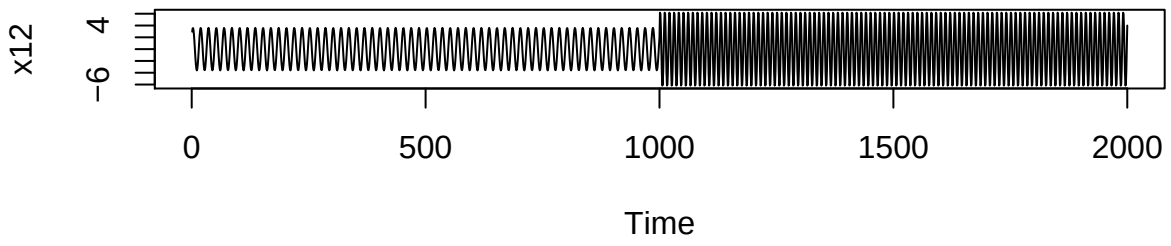
```
## Warning: package 'signal' was built under R version 3.1.2
```

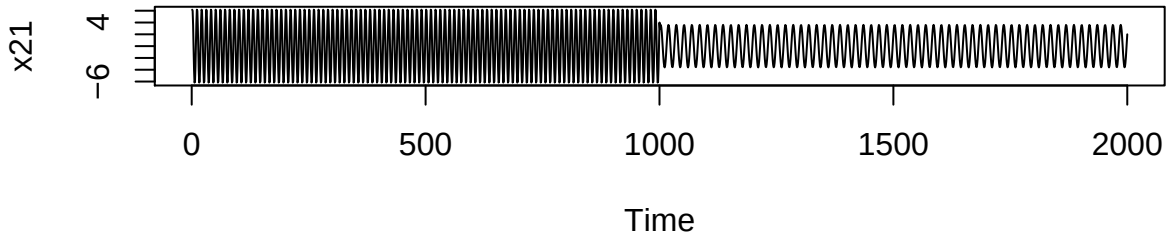
```
##
## Attaching package: 'signal'
##
## The following objects are masked from 'package:stats':
##
##     filter, poly
```

```
t=1:1000
x1=2*cos(2*pi*t*6/100)+3*sin(2*pi*t*6/100)
x2=4*cos(2*pi*t*10/100)+5*sin(2*pi*t*10/100)
x3=6*cos(2*pi*t*40/100)+7*sin(2*pi*t*40/100)
x=x1+x2+x3
plot.ts(x1)
```

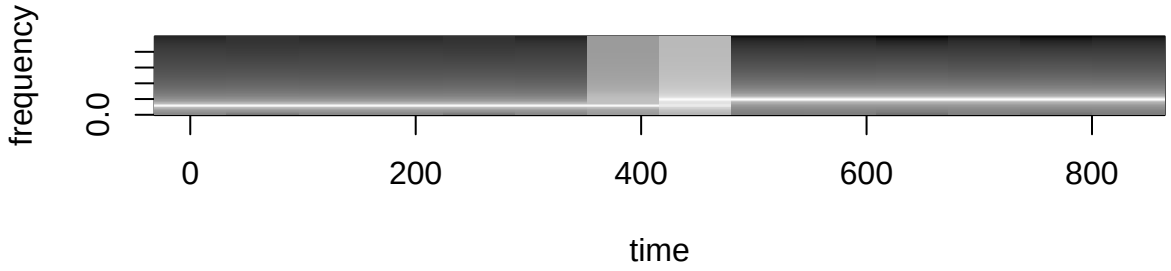


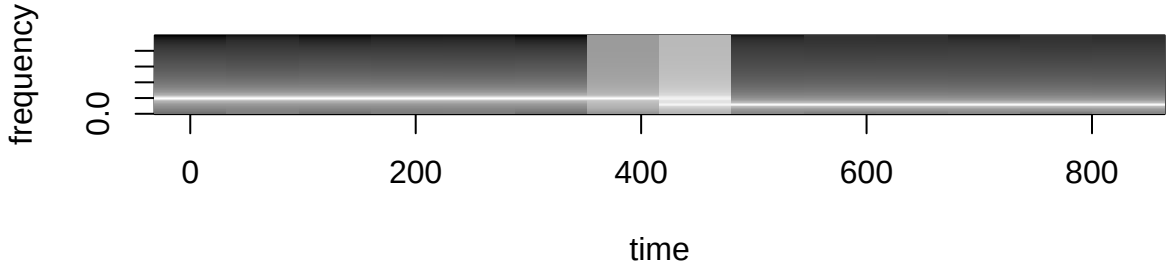

```
x12=c(x1, x2)
x21=c(x2, x1)
par(mfrow=c(2,1))
plot.ts(x12)
plot.ts(x21)
```






```
par(mfrow=c(2,1))
specgram(x12)
specgram(x21)
```

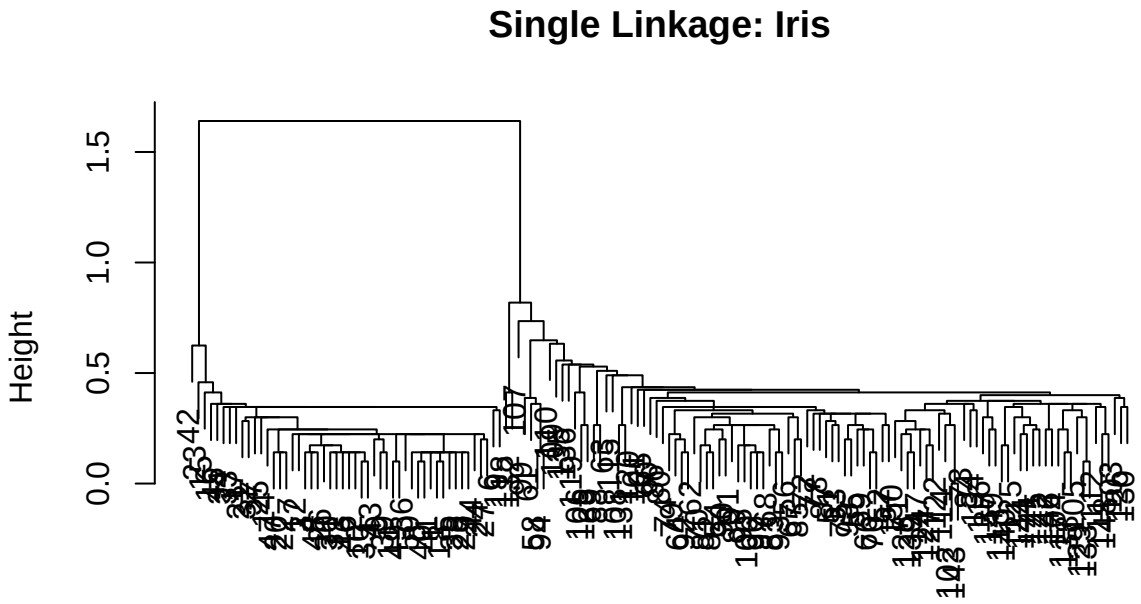




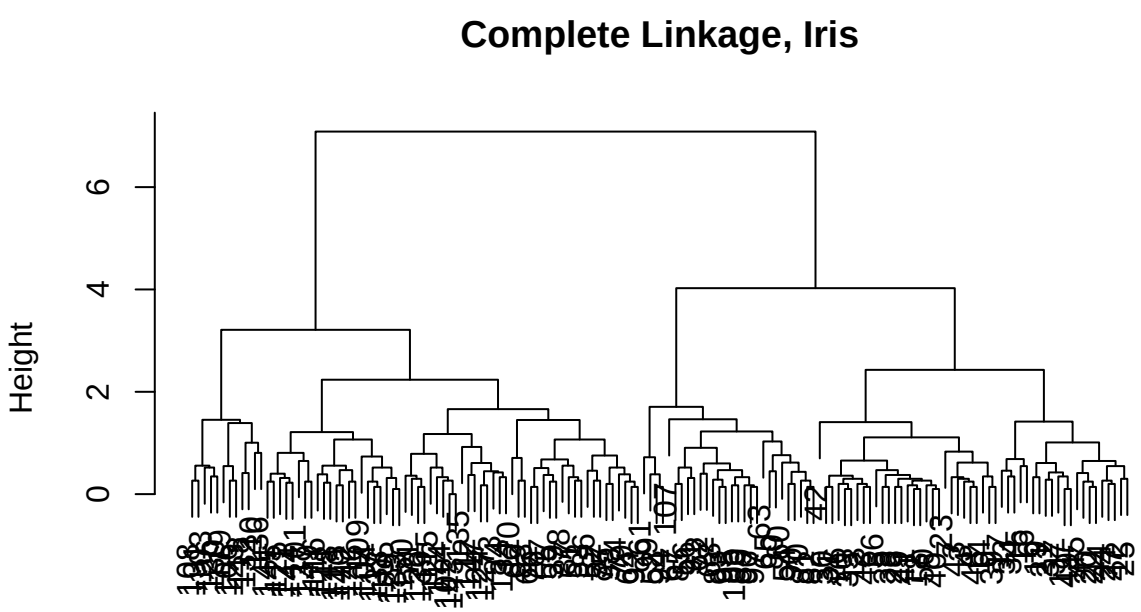
===

```
library(MASS); data <- iris[,1:4]

species <- c(rep(1, 50), rep(2, 50), rep(3, 50))
```

```
hc.s <- hclust(dist(data), method="single")
plot(hc.s, xlab="", sub="", main="Single Linkage: Iris")
```

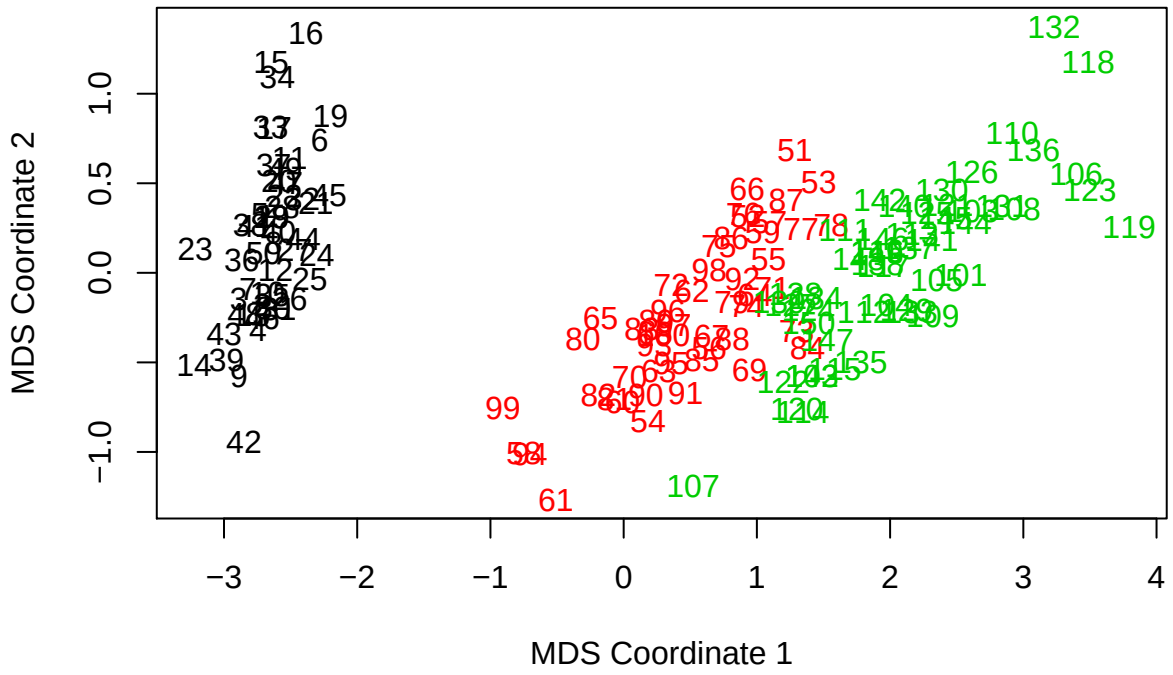



```
hc.c <- hclust(dist(data), method="complete")
plot(hc.c, xlab="", sub="", main="Complete Linkage, Iris")
```

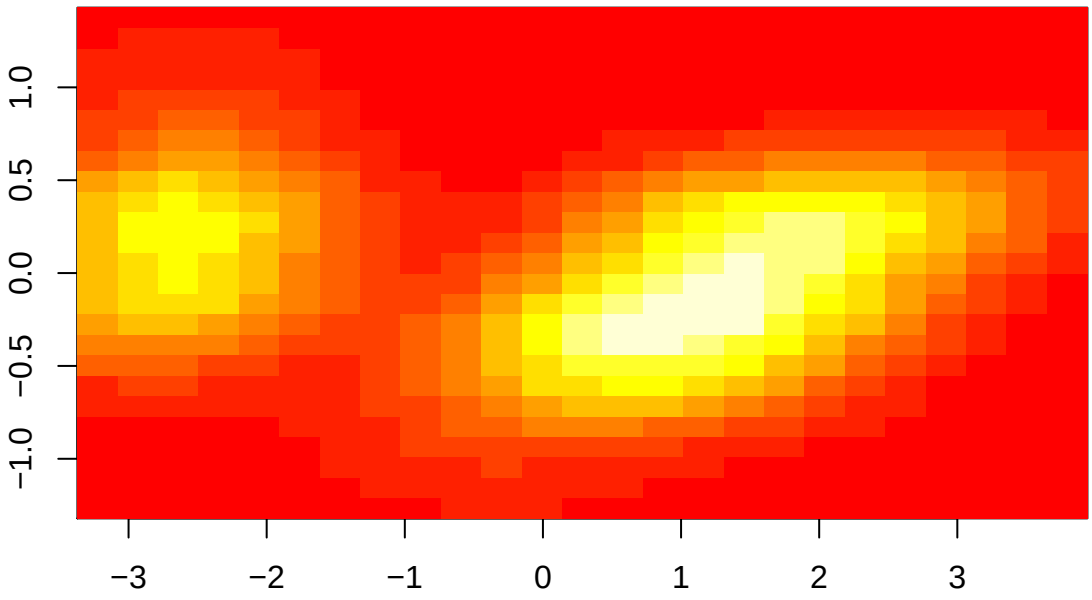



```
height.4 <- cutree(hc.c, h=4)
no.clus.4 <- cutree(hc.c, k=4)
```

```
ir.mds <- cmdscale(dist(data), k=2)
plot(ir.mds, type="n", xlab="MDS Coordinate 1", ylab="MDS Coordinate 2")
text(ir.mds, labels=seq(1,150), col=species)
```

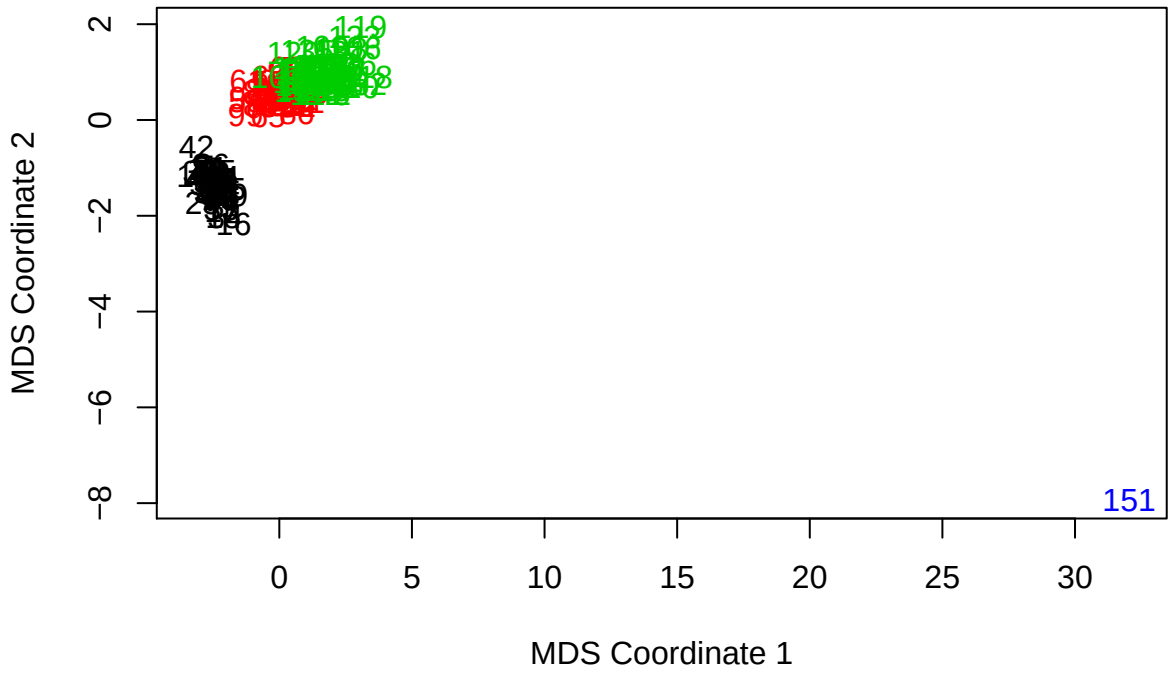



```
kde <- kde2d(ir.mds[,1], ir.mds[,2]); image(kde)
```




```
dist.orig <- dist(data)
dist.mds <- dist(ir.mds)
stress <- sum((dist.orig-dist.mds)^2)/sum(dist.orig^2)
```

```
data2 <- rbind(data, c(20, 20, 20, 20))
ir.mds <- cmdscale(dist(data2), k=2)
plot(ir.mds, type="n", xlab="MDS Coordinate 1", ylab="MDS Coordinate 2")
text(ir.mds, labels=seq(1, 151), col=c(species, 4))
```

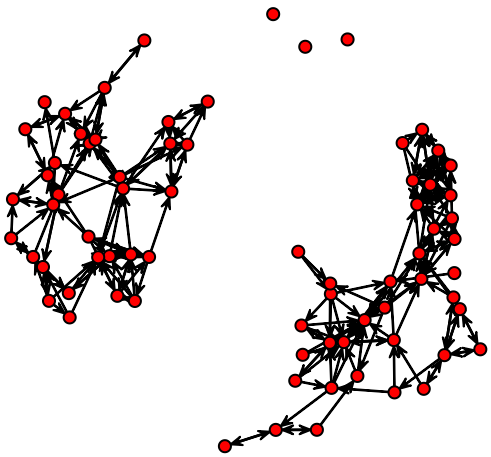



```
library(sna)
```

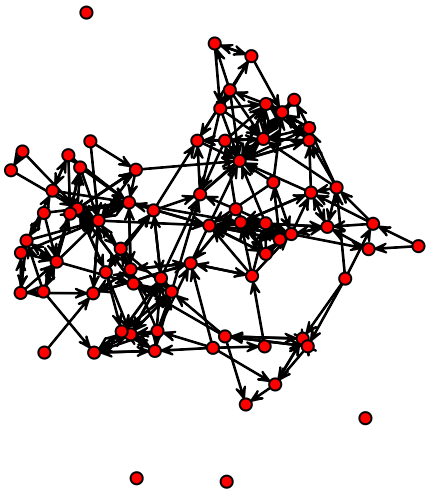
```
## Warning: package 'sna' was built under R version 3.1.2
```

```
## sna: Tools for Social Network Analysis
## Version 2.3-2 created on 2014-01-13.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##  For citation information, type citation("sna").
##  Type help(package="sna") to get started.
```

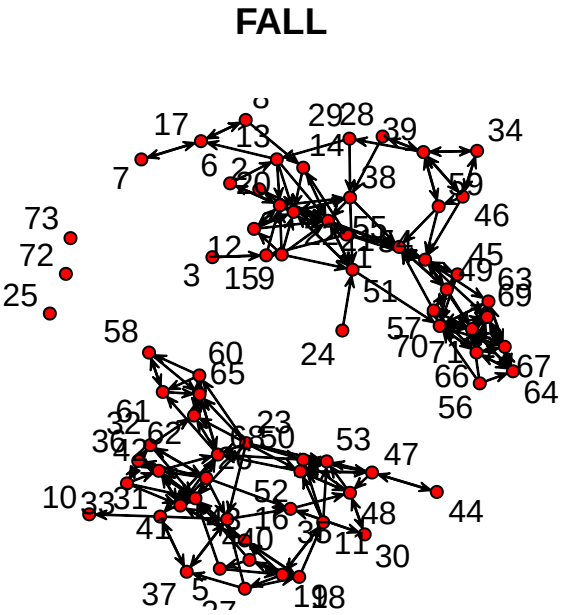
```
data(coleman)
gplot(coleman[1,,])
```



```
gplot(coleman[2,,])
```



```
gplot(coleman[1,,], label=seq(1, 73), main="FALL")
```



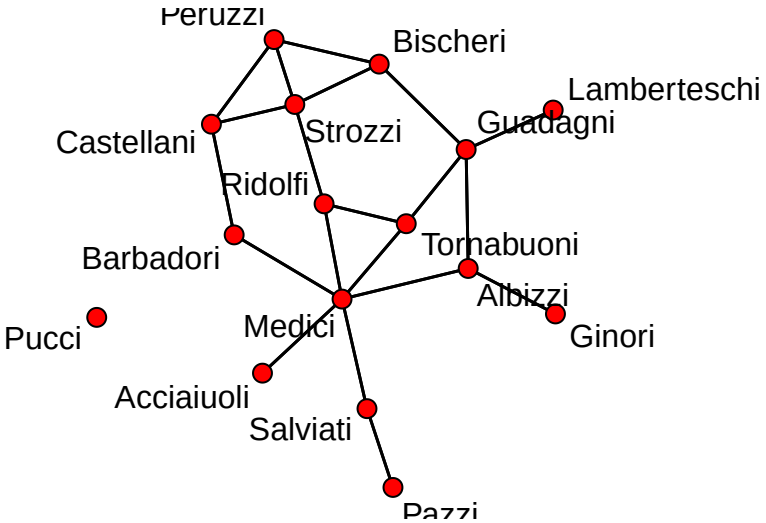

```
library(network)
```

```
## Warning: package 'network' was built under R version 3.1.2
```

```
## network: Classes for Relational Data
## Version 1.11.3 created on 2014-12-05.
## copyright (c) 2005, Carter T. Butts, University of California-Irvine
##                     Mark S. Handcock, University of California -- Los Angeles
##                     David R. Hunter, Penn State University
##                     Martina Morris, University of Washington
##                     Skye Bender-deMoll, University of Washington
##  For citation information, type citation("network").
##  Type help("network-package") to get started.
##
```

```
##
## Attaching package: 'network'
##
## The following object is masked from 'package:sna':
##
##     %c%
##
## The following object is masked from 'package:Hmisc':
##
##     is.discrete
```

```
data(flo)
gplot(flo, label=rownames(flo), gmode="graph") #gmode="graph"
```



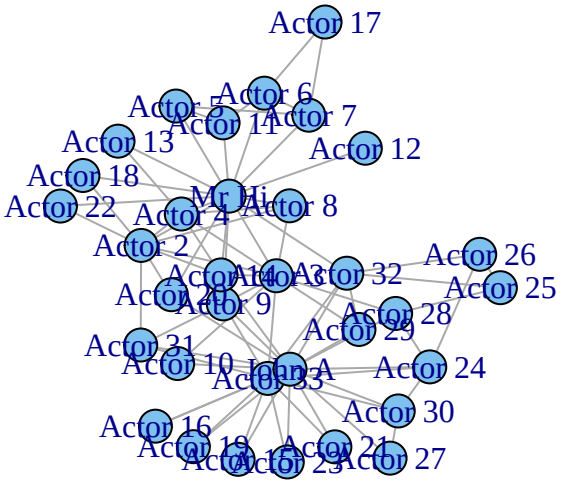

```
library(igraph); library(igraphdata)
```

```
## Warning: package 'igraph' was built under R version 3.1.2
```

```
##
## Attaching package: 'igraph'
##
## The following objects are masked from 'package:network':
##
##     %c%, %s%, add.edges, add.vertices, delete.edges,
##     delete.vertices, get.edge.attribute, get.edges,
##     get.vertex.attribute, is.bipartite, is.directed,
##     list.edge.attributes, list.vertex.attributes,
##     set.edge.attribute, set.vertex.attribute
##
## The following objects are masked from 'package:sna':
##
##     %c%, betweenness, bonpow, closeness, degree, dyad.census,
##     evcent, hierarchy, is.connected, neighborhood, triad.census
```

```
## Warning: package 'igraphdata' was built under R version 3.1.2
```

```
data(karate)
plot(karate)
```




```
vcount(karate)
```

```
## [1] 34
```

```
ecount(karate)
```

```
## [1] 78
```

```
neighbors(karate, 1, mode="in")
```

```
##  [1]  2  3  4  5  6  7  8  9 11 12 13 14 18 20 22 32
```

```
neighbors(karate, 34, mode="out")
```

```
##  [1]  9 10 14 15 16 19 20 21 23 24 27 28 29 30 31 32 33
```

```
are.connected(karate, 1, 34) #edge, not path
```

```
## [1] FALSE
```

```
shortest.paths(karate, 1, 34)
```

```
##       John A
## Mr Hi      3
```

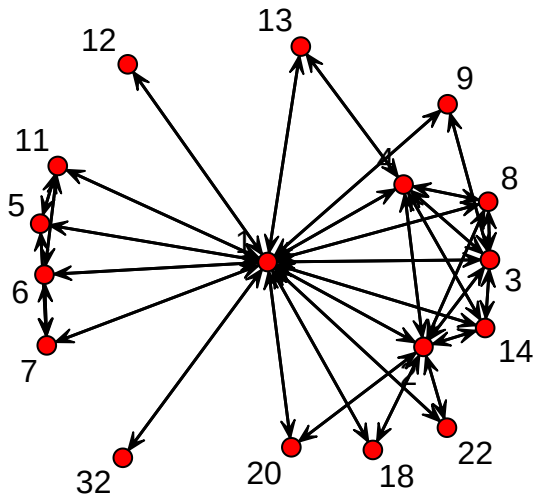
```
get.shortest.paths(karate, 1, 34)
```

```
## $vpath
## $vpath[[1]]
## [1]  1 20 34
##
##
## $epath
## NULL
##
## $predecessors
## NULL
##
## $inbound_edges
## NULL
```

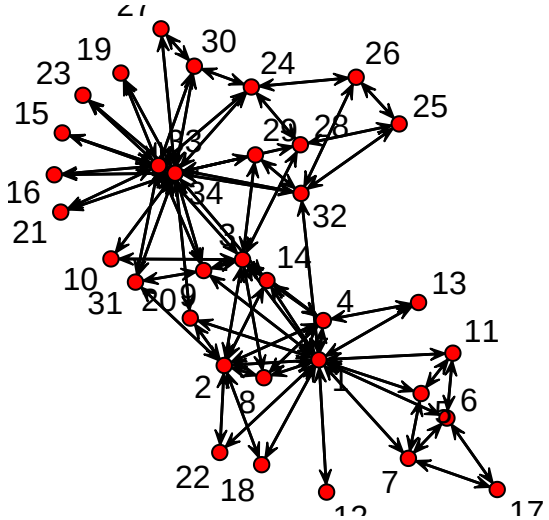
```
no.clusters(karate)
```

```
## [1] 1
```

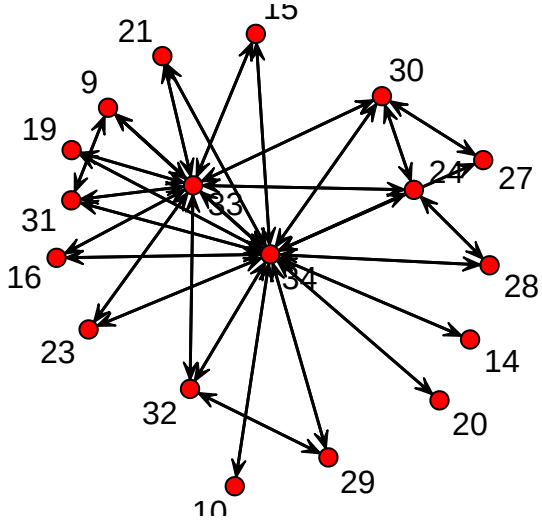
```
ego <- 1
karate.adj <- get.adjacency(karate)
alters <- which(karate.adj[ego,]==1)
gplot(as.matrix(karate.adj[c(ego, alters), c(ego, alters)]), label=c(ego, alters))
```




```
karate.adj <- get.adjacency(karate)
karate.adj <- as.matrix(karate.adj)
gplot(karate.adj, label=seq(1,34))
```



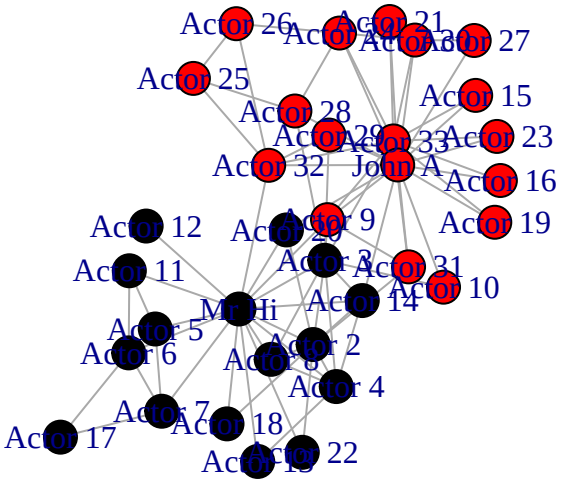

```
ego <- 34
alters <- which(karate.adj[ego,]==1)
gplot(as.matrix(karate.adj[c(ego, alters), c(ego, alters)]), label=c(ego, alters))
```




```
V(karate)$Faction
```

```
##  [1] 1 1 1 1 1 1 1 1 2 2 1 1 1 1 2 2 1 1 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2
```

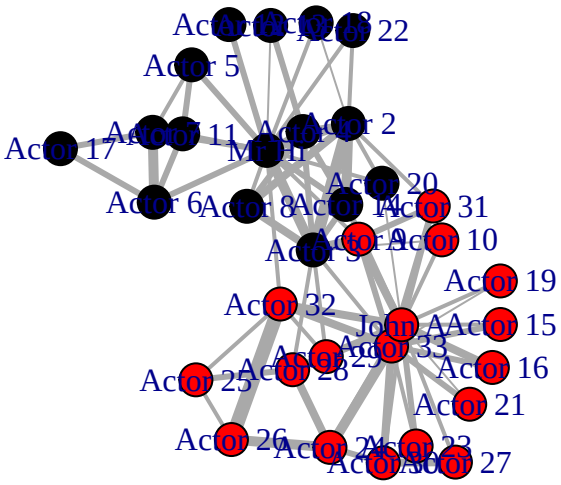
```
V(karate)$color <- V(karate)$Faction
plot(karate)
```



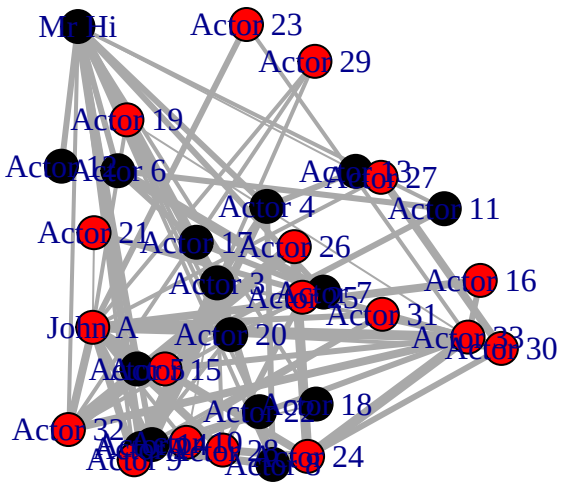

```
E(karate)$weight
```

```
##  [1] 4 5 3 3 3 3 2 2 2 3 1 3 2 2 2 2 6 3 4 5 1 2 2 2 3 4 5 1 3 2 2 2 3 3 3
## [36] 2 3 5 3 3 3 3 3 4 2 3 3 2 3 4 1 2 1 3 1 2 3 5 4 3 5 4 2 3 2 7 4 2 4 2
## [71] 2 4 2 3 3 4 4 5
```

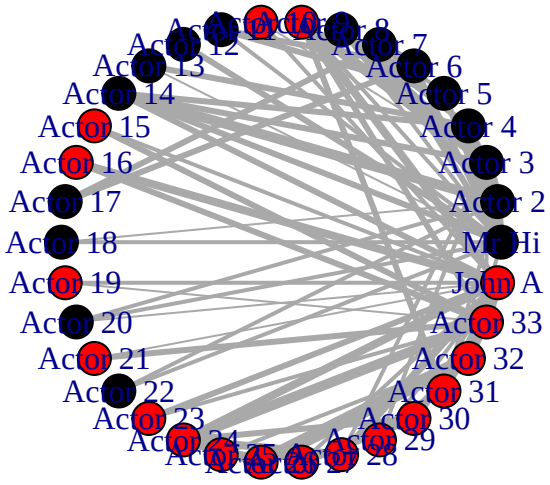
```
E(karate)$width <- E(karate)$weight
plot(karate)
```



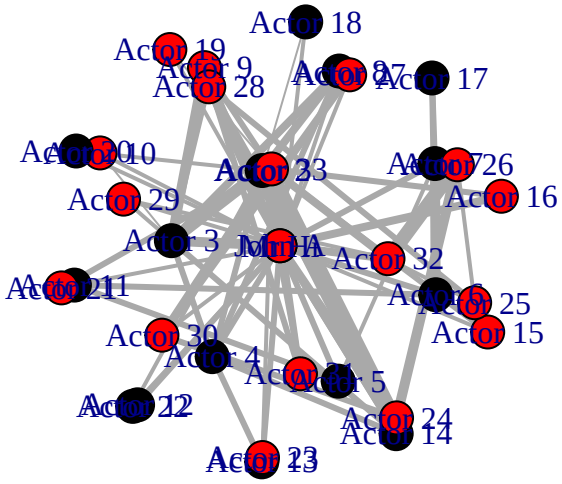

```
plot(karate, layout=layout.random)
```



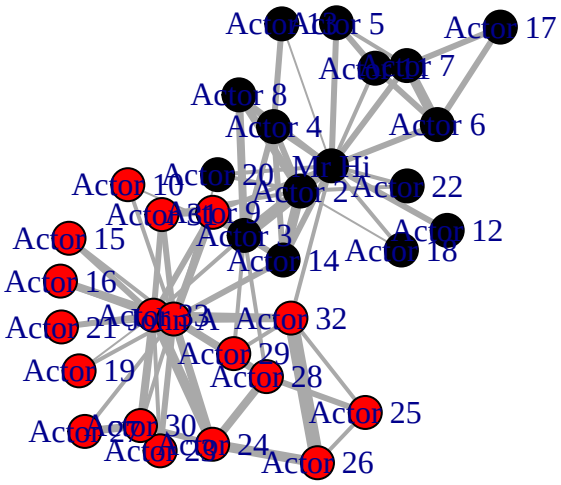

```
plot(karate, layout=layout.circle)
```



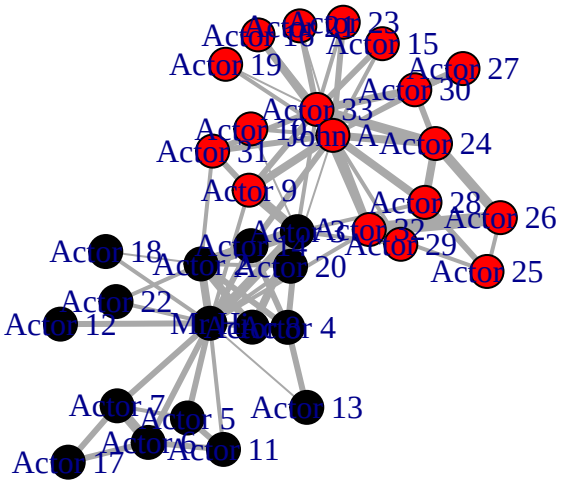

```
plot(karate, layout=layout.sphere)
```



```
plot(karate, layout=layout.kamada.kawai)
```



```
plot(karate, layout=layout.fruchterman.reingold)
```



```
library(survey)
```

```
##
## Attaching package: 'survey'
##
## The following object is masked from 'package:Hmisc':
##
##     deff
##
## The following object is masked from 'package:graphics':
##
##     dotchart
```

```
data(api)
dclus2 <- svydesign(id=~dnum+snum, fpc=~fpc1+fpc2, data=apiclus2)
srs <- svydesign(id=~1, weights=NULL, data=apiclus2)
```

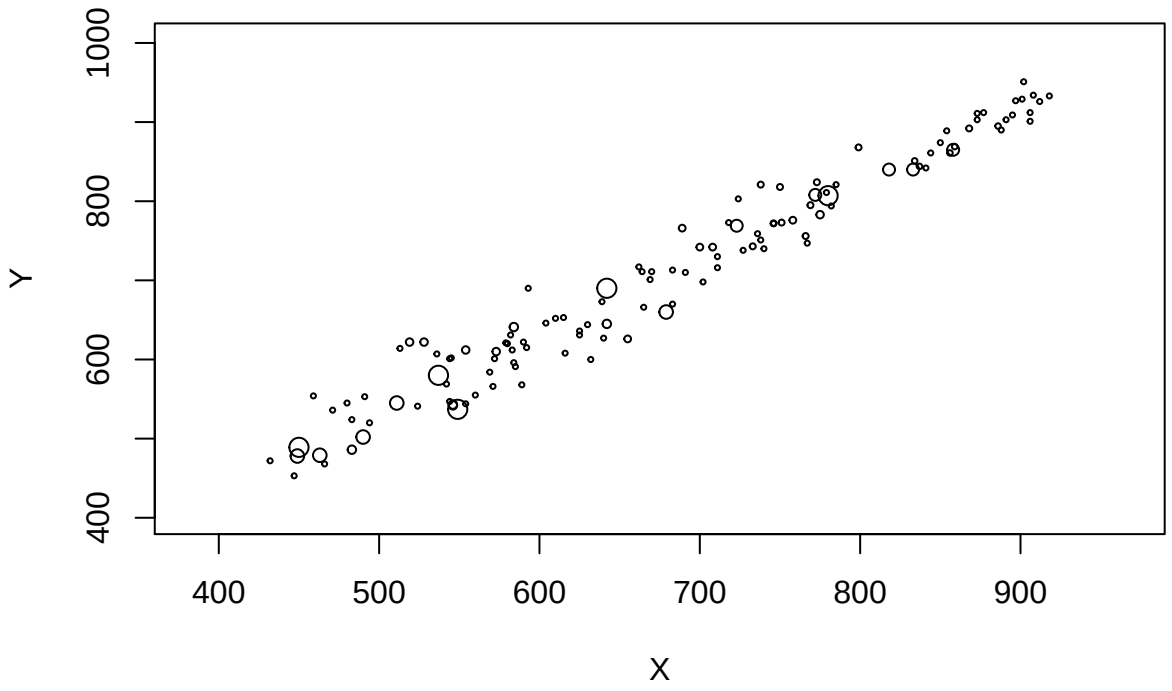
```
mns <- svymean(~api00+api99,dclus2)
svyby(~api99, ~stype, dclus2, svymean)
```

```
##   stype    api99       se
## E     E 658.4204 31.49901
## H     H 596.6209 17.33370
## M     M 630.8560 40.06717
```

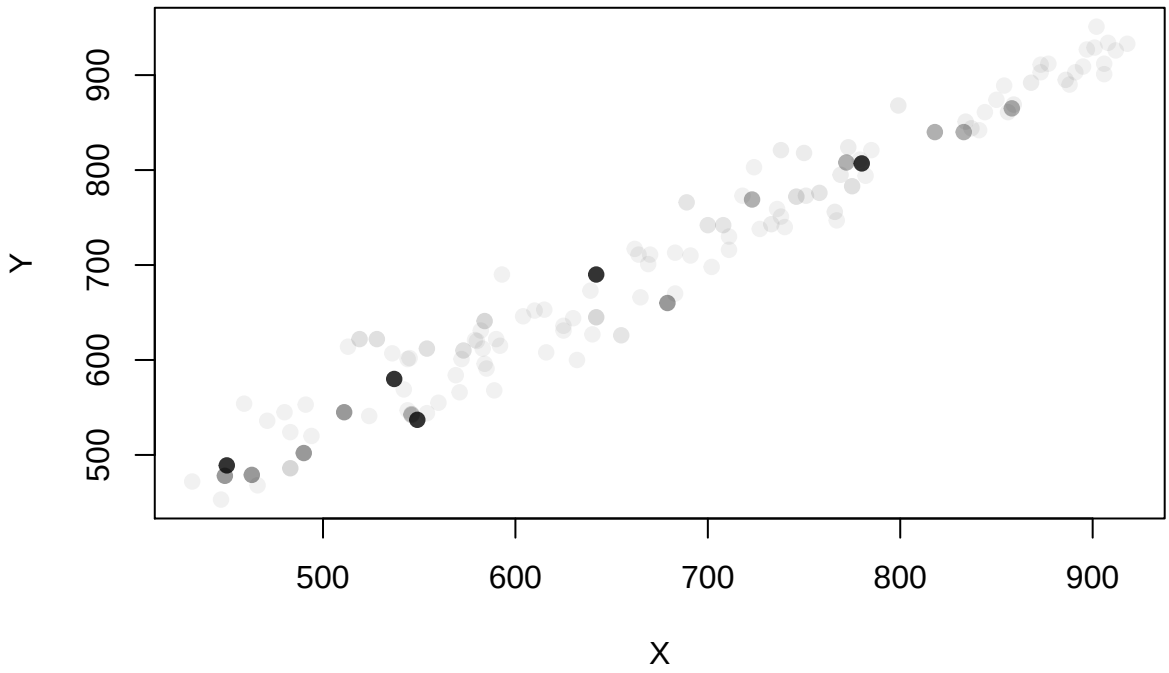
```
svyby(~api99, ~stype, dclus2, svyquantile, quantiles=c(.25, 0.5), ci=TRUE)
```

```
##   stype     0.25      0.5  se.0.25   se.0.5
## E     E 536.0799 641.7292 33.96602 57.57923
## H     H 465.7000 595.2000 36.46726 23.08532
## M     M 545.9167 569.7273 20.66885 77.00761
```

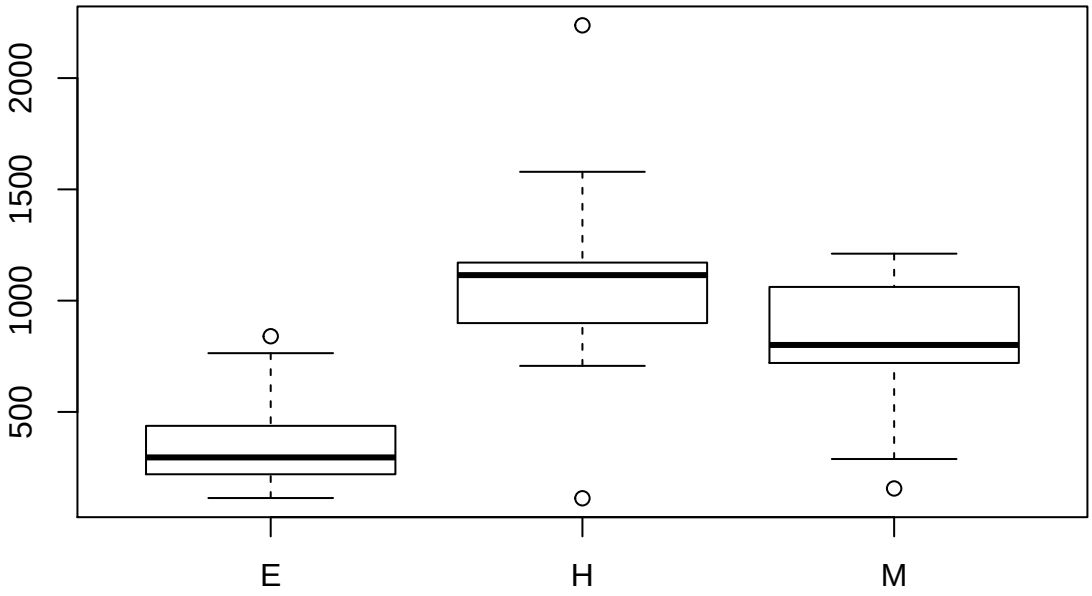
```
svyplot(api00~api99, dclus2)
```



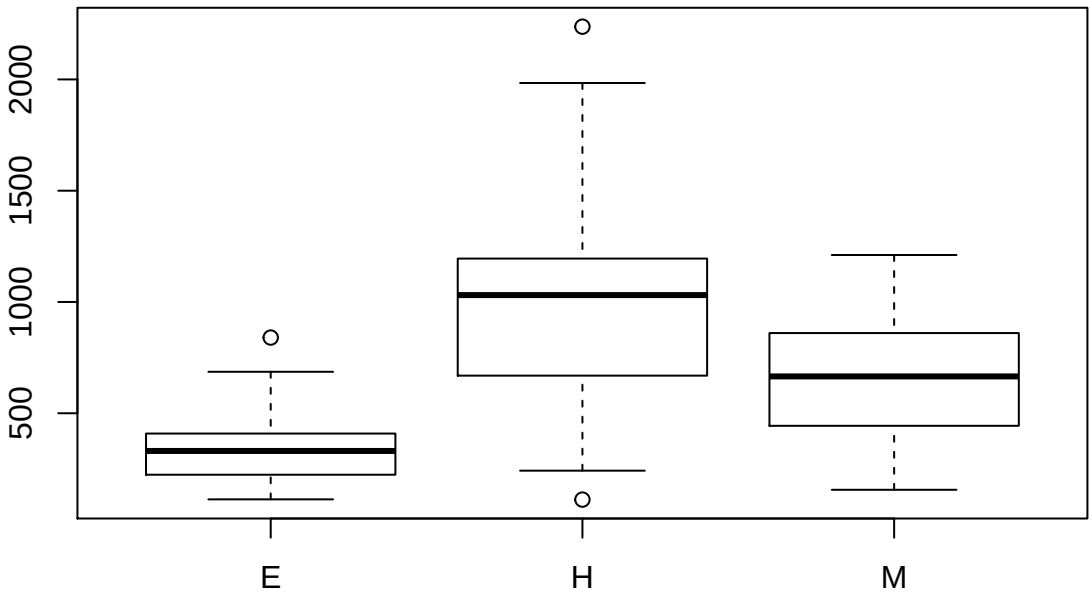

```
svyplot(api00~api99, design=dclus2, style="transparent",pch=19)
```




```
svyboxplot(enroll~stype, design=dclus2)
```



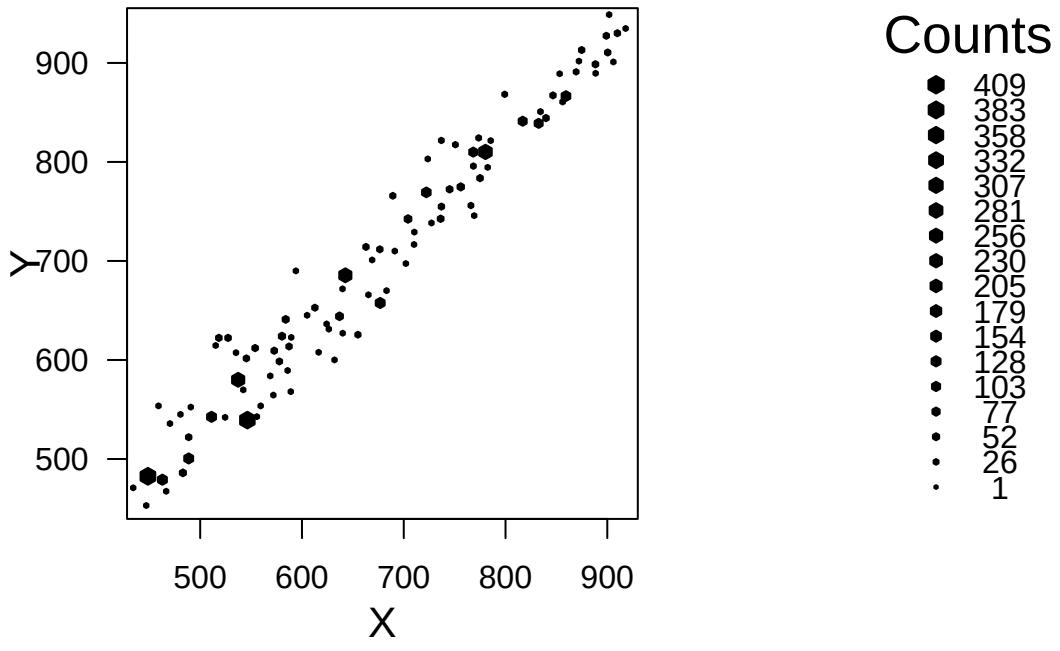

```
svyboxplot(enroll~stype, design=srs)
```




```
library(hexbin)
```

```
## Warning: package 'hexbin' was built under R version 3.1.2
```

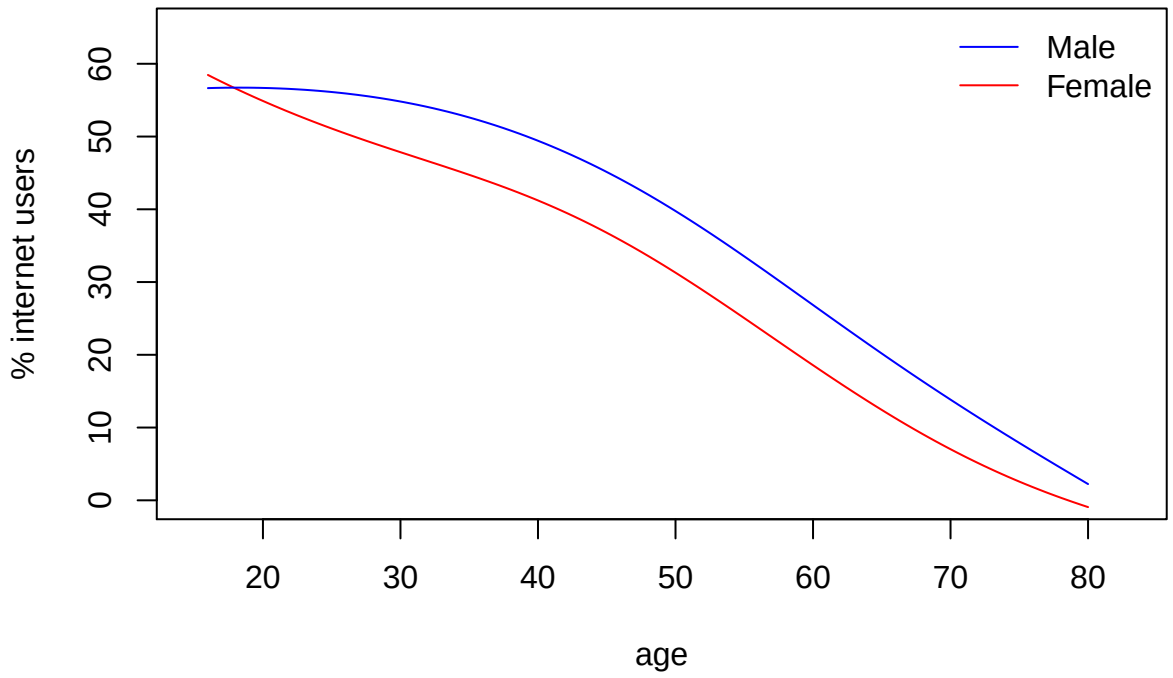
```
svyplot(api00~api99, style="hex", dclus2)
```



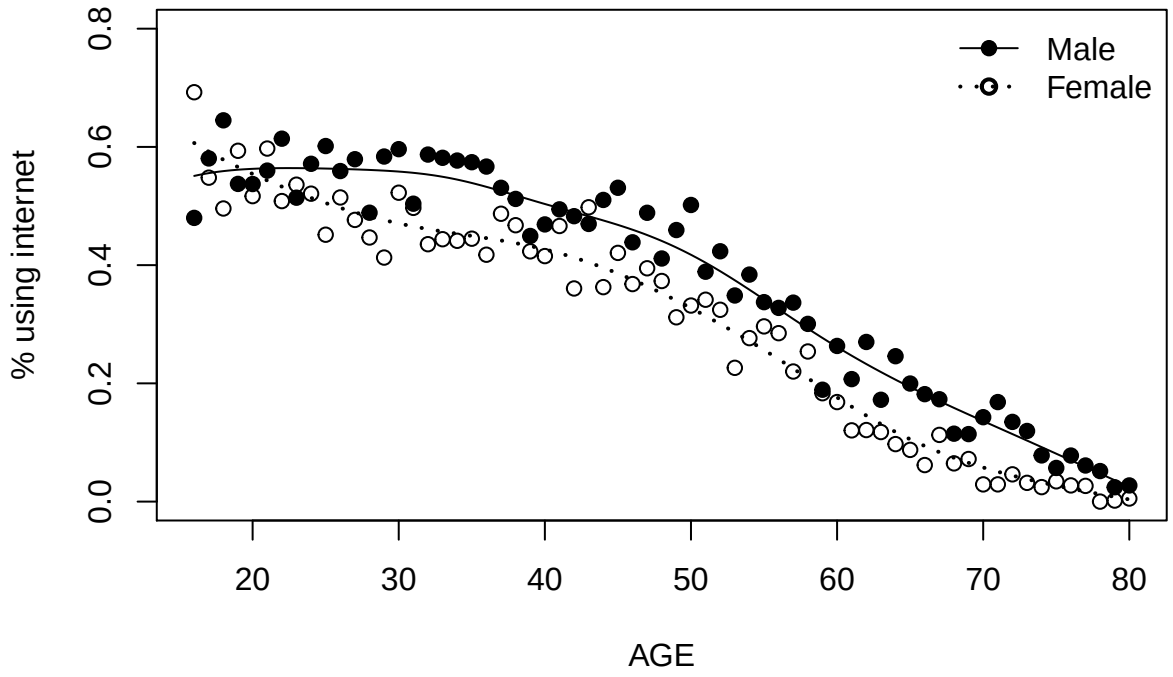

```
library(KernSmooth)
```

```
## KernSmooth 2.23 loaded
## Copyright M. P. Wand 1997-2009
```

```
shs <- read.csv('shs.csv', header=T)
shs.des <- svydesign(id=~PSU, weights=~GROSSWT,strata=~STRATUM,data=shs)
plot(c(15,83),c(0,65),type='n',xlab='age',ylab='% internet users')
legend("topright",lty=c(1,1),col=c("blue","red"),legend=c("Male","Female"),bty="n")
lf<-svysmooth(I(100*INTUSE)~AGE,subset(shs.des,SEX=="female" & !is.na(AGE)),bandwidth=10)
lines(lf,col="red")
lm<-svysmooth(I(100*INTUSE)~AGE,subset(shs.des,SEX=="male" & !is.na(AGE)),bandwidth=10)
lines(lm, col="blue")
```




```
bys<-svyby(~INTUSE,~AGE+SEX,svymean,design=shs.des)
plot(svysmooth(INTUSE~AGE,
design=subset(shs.des,SEX=="male" & !is.na(AGE)),
bandwidth=5),ylim=c(0,0.8),ylab="% using internet")
lines(svysmooth(INTUSE~AGE,
design=subset(shs.des,SEX=="female" & !is.na(AGE)),
bandwidth=5),lwd=2,lty=3)
points(bys$AGE,bys$INTUSE,pch=ifelse(bys$SEX=="male",19,1))
legend("topright",pch=c(19,1),lty=c(1,3),lwd=c(1,2),
legend=c("Male","Female"),bty="n")
```




```
byinc<-svyby(~INTUSE, ~SEX+GROUPINC, svymean, design=shs.des)
barplot(byinc)
```