**1 - Basic and Advanced Graphics in R**

There are numerous web resources for learning about graphics in R and I will try to link some that I find most useful to my website. There are a plethora of packages that allow R users to create specialized graphics. We will examine a few functions from R packages in this handout, however there is a lot more out there I will not be touching on here. You should have all of the data sets I am using in these examples in the mult.Rdata that you should have in an e-mail from me.

**R Packages using in this chapter:** ggplot2, s20x, violinmplot, corrgram, corrplot, lattice, ash, MASS, aplpack

* 1. **- Histograms and Kernel Density Estimates (KDEs)**

Example 1.1: Old data on the 77 largest U.S. cities

> City = read.csv(file.choose()) # read in City77.csv from my website.

> row.names(City) = City$City # assign city name as row label.  
> City = City[,-1] # remove city name column from data frame.

> names(City)  
 [1] "pop90" "pop80" "growth" "area" "popdens" "blackpop" "hispop" "asianpop"

[9] "pctblack" "pcthisp" "pctasian" "k12enr" "pct.bach" "pct.elder" "medinc" "taxes"

[17] "pct.pa" "poverty" "pct.1par" "unempl" "laborchg" "femlab" "pct.manu" "pubtrans"

[25] "medvalue" "medrent" "pct.rent" "oldhomes" "pct.cond" "pct.sing" "infmort" "mean.jul"

[33] "precip" "tot.mort" "heart" "cancer" "lung" "suicide" "vdeaths" "PM.max"

[41] "PM.ave" "SO2.ave" "CO.max" "NO2.ave" "Ozone" "Lead" "tot.tax" "COL.util"

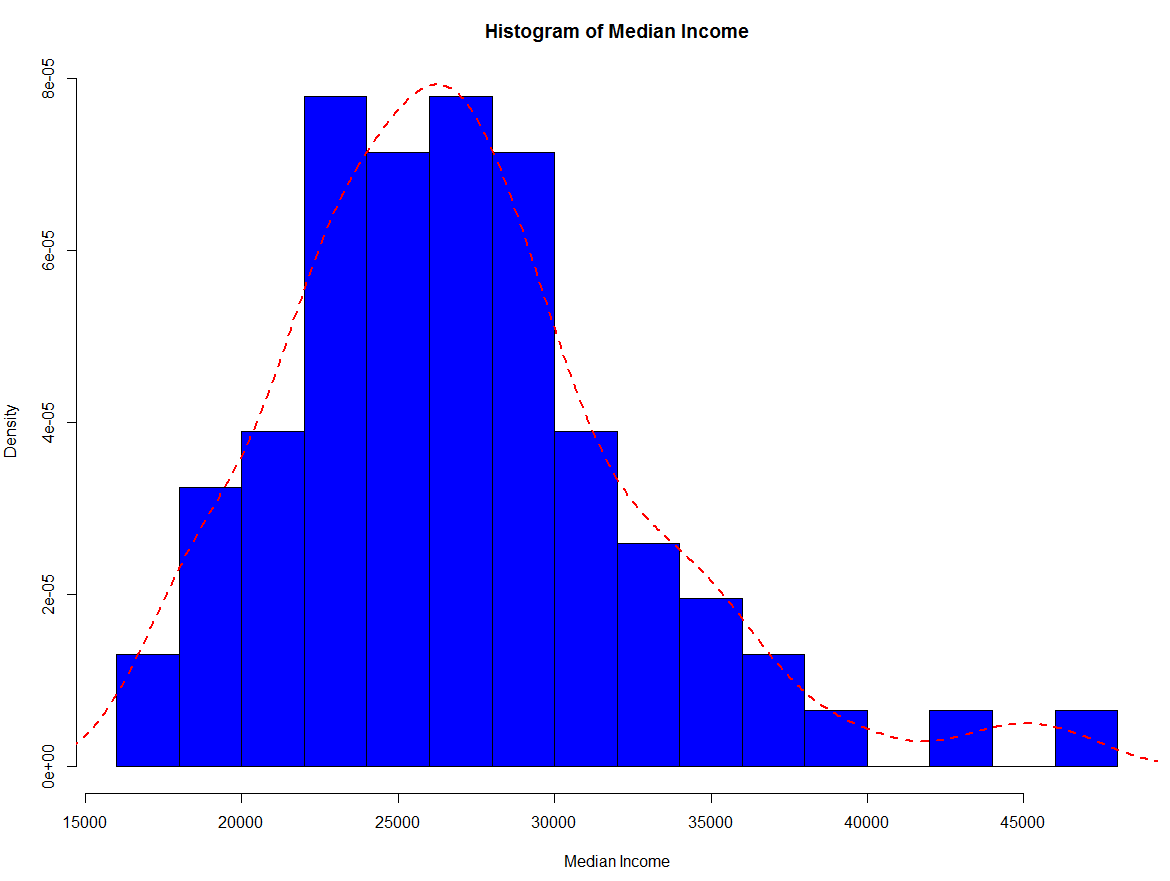
[49] "COL.ptax" "COL.food" "COL.tran" "job.grow" "cycthrt" "pct.mobi" "home90" "commute"

[57] "murder" "rape" "robbery" "assault" "violent" "burglary" "theft" "auto.thft"

[65] "tot.prop" "ctrend" "zeroday" "freeze" "hot" "ddays" "high.jul" "low.jan"

> hist(City$medinc,prob=T,nclass=20,col = “blue”,main="Histogram of Median Income",xlab=”Median Income”)   
  
**Note:** prob =T gives a probability histogram, nclass = 20 specifies that we wish to use approximately 20 class intervals to construct the histogram, col=”blue” specifies that we wish to have the bars colored blue, and main=”Histogram …” specifies a title for the plot. All of these are optional arguments that do not necessarily need to included. To see what other options are available a general help file for given command can be obtained if we type ? immediately followed by the name of the command. For example > ?hist will bring up the help file for this histogram function.

> lines(density(City$medinc),lty=2,col=”red”,lwd=2)

lines is a general command for adding a line to the current plot. It requires some mechanism for   
 generating the line. Here we have used a kernel density estimate which is essentially a smoothed   
 histogram. Also lty > 1 gives various types of dashed lines, lwd > 1 increases the line width, and   
 col = “red” sets color of the line.  
  


**In ggplot2**

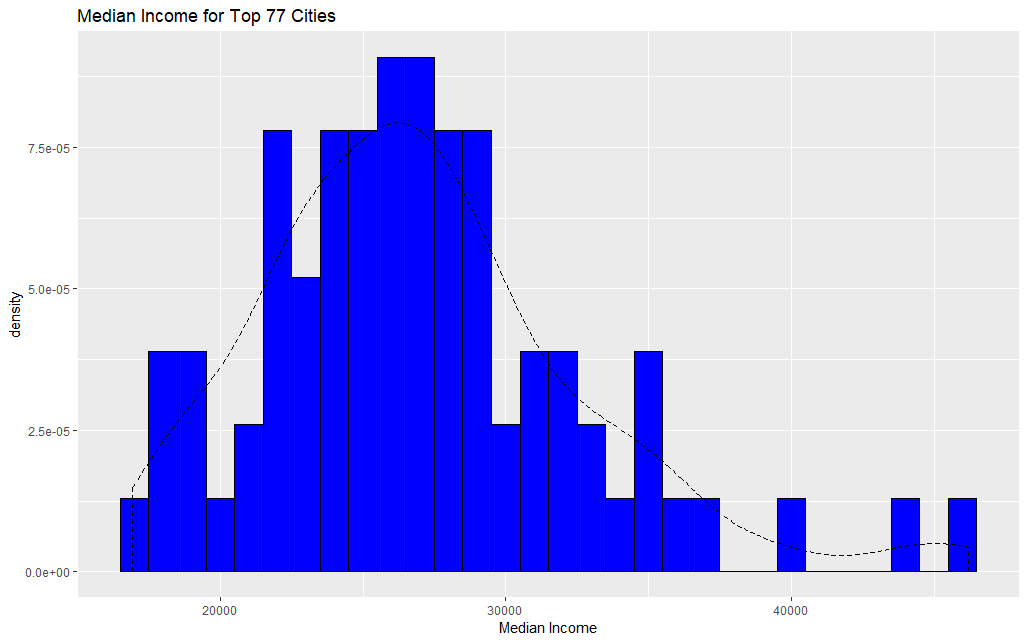
a = ggplot(City,aes(medinc))

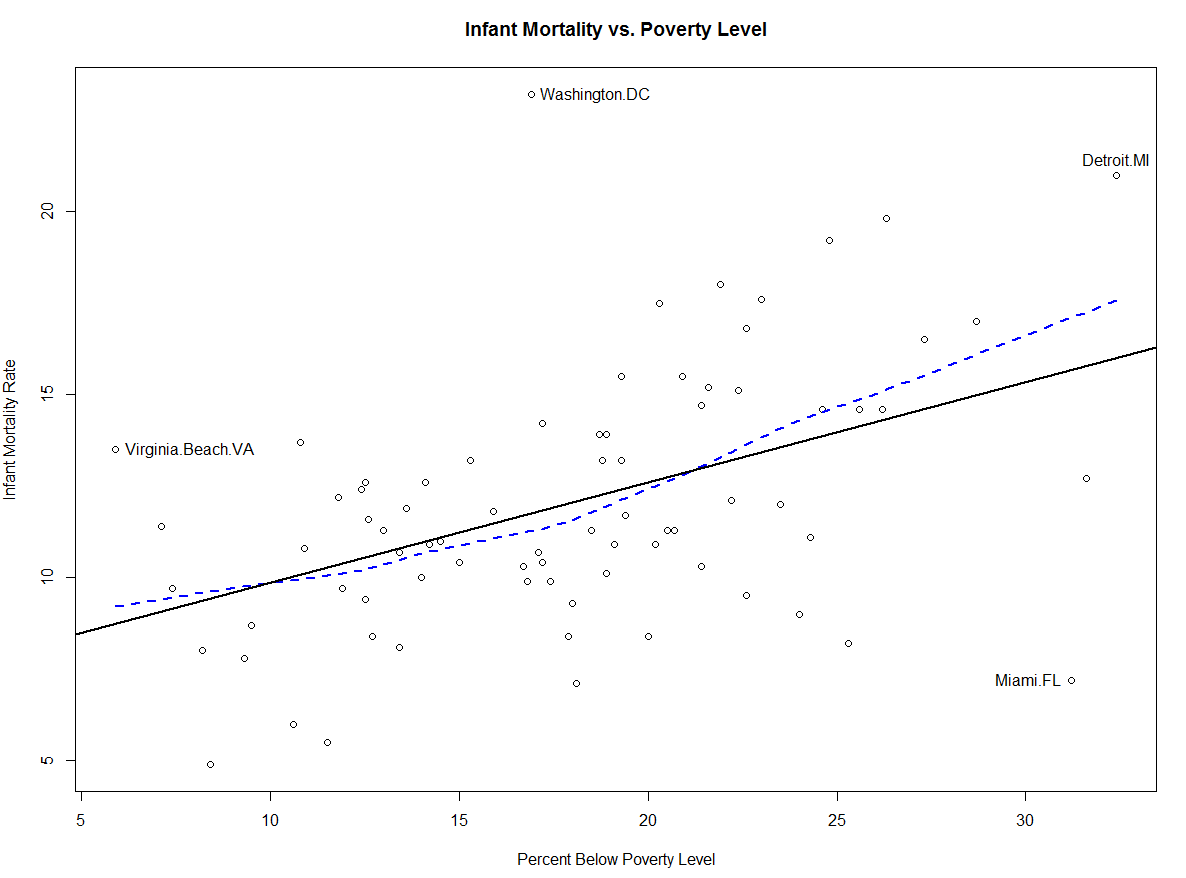
a + geom\_histogram(binwidth=1000,aes(y=..density..),fill="blue",color="black") +

geom\_density(linetype=2) +

xlab("Median Income") +

ggtitle("Median Income for Top 77 Cities")



**1.2 – Scatterplots and Identifying Cases**  
Example 1.1: U.S. Cities (cont’d)  
> plot(City$poverty,City$infmort,xlab="Percent Below Poverty Level",ylab="Infant Mortality Rate")  
> lines(lowess(City$poverty,City$infmort),lty=2,col="blue",lwd=2)    
> abline(lm(infmort~poverty,data=City),lwd=2)    
> identify(City$poverty,City$infmort,labels=row.names(City))    
[1] 19 44 33 9 🡨 these are the observation numbers that were highlighted using the mouse.  
> title(main="Infant Mortality vs. Poverty Level") # add a title to the final plot  


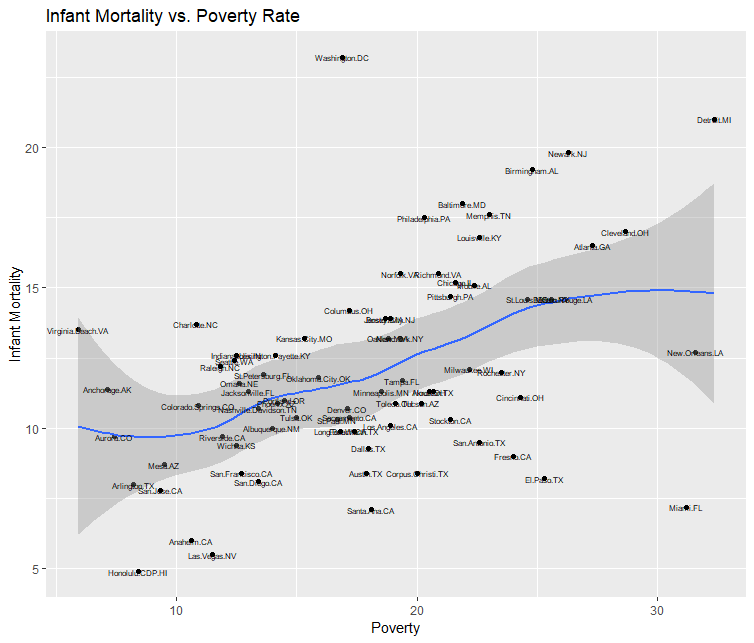
**In ggplot2**

a = ggplot(City,aes(poverty,infmort))

a + geom\_point() + geom\_smooth() +   
 geom\_text(aes(label=row.names(City)),size=2) +

xlab("Poverty") + ylab("Infant Mortality") +

ggtitle("Infant Mortality vs. Poverty Rate")



**1.3 - Comparative Boxplots and Violin Plots**When we are interested in comparing a numeric response across the levels of nominal/categorical/ordinal variable comparative boxplots are an effective graphical tool for making such comparisons.Example 1.2: Fatty Acid Content of Italian Olive Oils

> Olives = read.csv(file.choose()) # read in Olive Oils.csv  
> attach(Olives) # Generally we avoid doing this but here it will save time.

> names(Olives)

[1] "Region.Name" "Area.Name" "Palmitic" "Palmitoleic" "Stearic" "Oleic" "Linoleic"

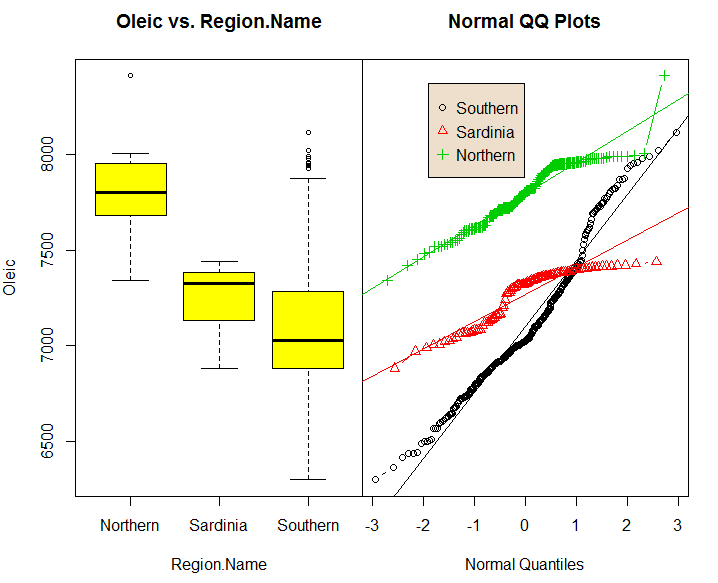
[8] "Eicosanoic" "Linolenic" "Eicosenoic"   
  
> par(mfrow=c(3,2)) # sets a plotting region for with 3 rows and 2 columns  
> boxplot(split(Oleic,Area.Name),xlab="Area",ylab="Oleic Acid")  
> boxplot(split(Linoleic,Area.Name),xlab="Area",ylab="Linoleic Acid")  
> boxplot(split(Stearic,Area.Name),xlab="Area",ylab="Stearic Acid")  
> boxplot(split(Linolenic,Area.Name),xlab="Area",ylab="Linolenic Acid")  
> boxplot(split(Eicosanoic,Area.Name),xlab="Area",ylab="Eicosanoic Acid")  
> boxplot(split(Eicosenoic,Area.Name),xlab="Area",ylab="Eicosenoic Acid")

Note that the basic form of the command is boxplot(split(variable,pop\_identifier))



I have written function which produces comparative boxplots along with normal quantile plots on the side much like those that can obtained in JMP called boxqq. An example of its use is shown below. This function is contained in the package s20x available on CRAN.

> boxqq(Oleic~Region.Name)

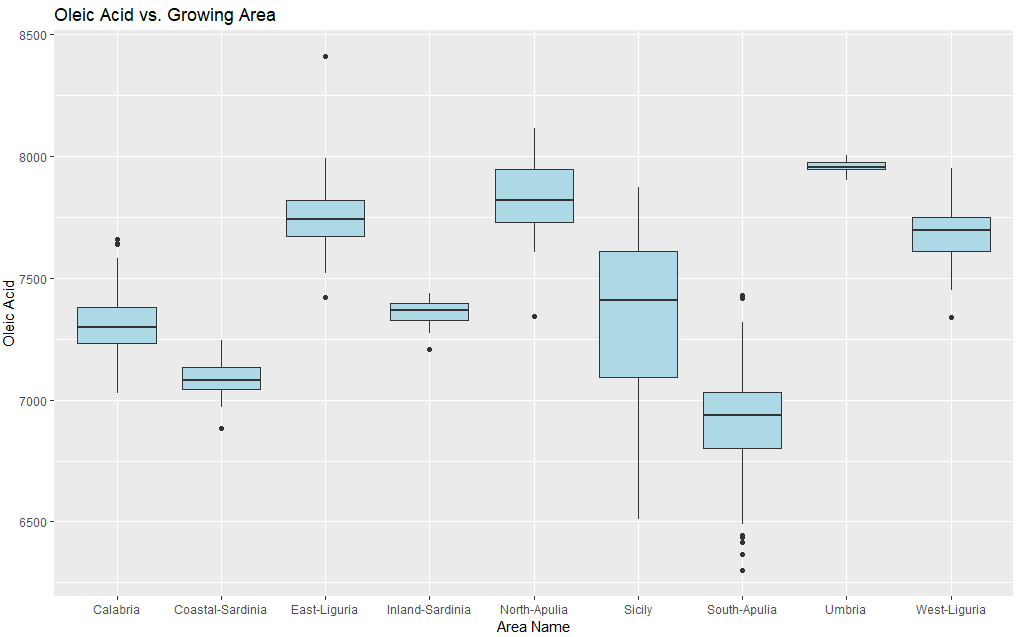


**In ggplot2**

a = ggplot(Olives,aes(Area.name,oleic))

a + geom\_boxplot(fill="lightblue") + xlab("Area Name") +

ylab("Oleic Acid") + ggtitle("Oleic Acid vs. Growing Area")

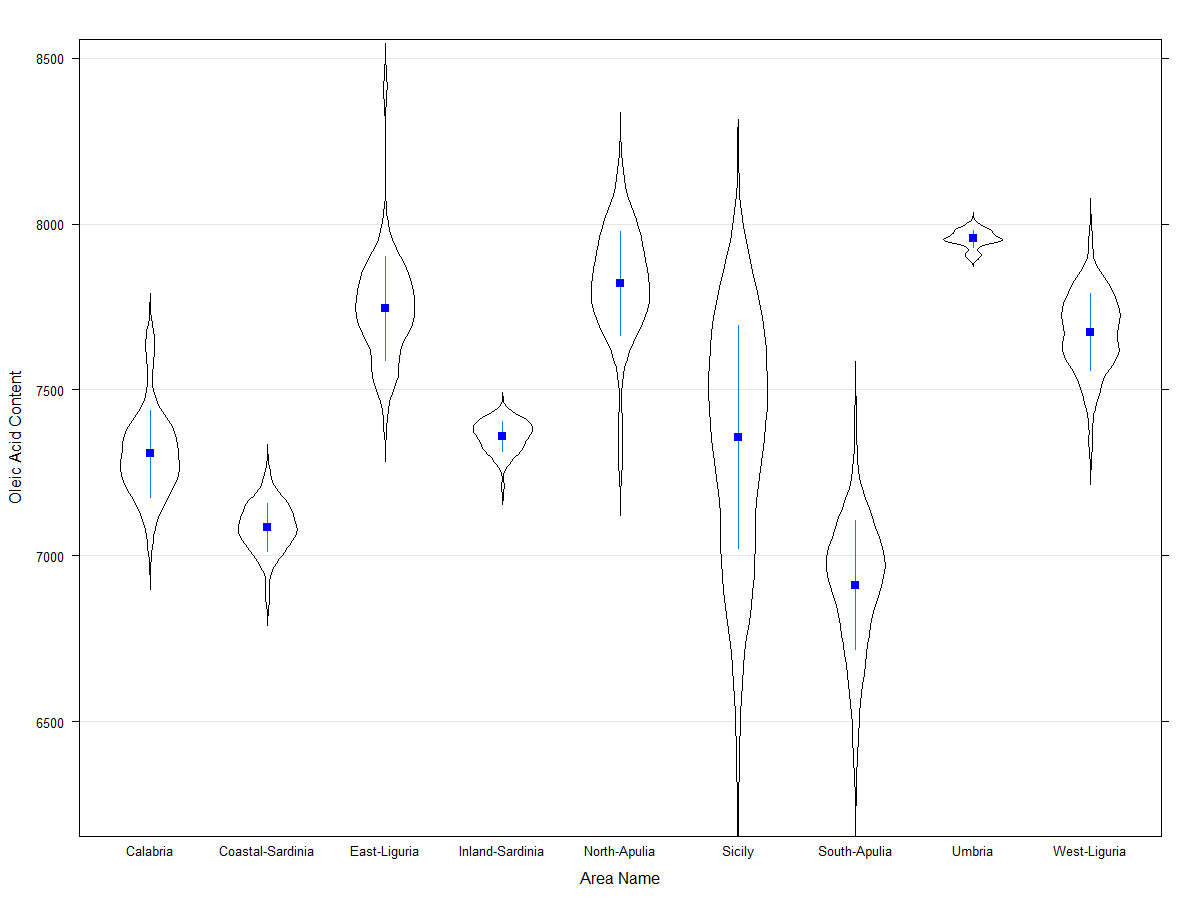
****

The package violinmplot from CRAN uses similar syntax to boxplot command above to create comparative violin plots. Violin plots show better detail on the distribution of the response within each group and provide mean & standard deviation information as well.

> library(violinmplot)

> violinmplot(Oleic~Area.Name,xlab=”Area Name”,ylab=”Oleic Acid Content”,data=Olives)

A violin plot uses a kernel density estimate on both sides of the “violin” to show the response distribution and shows the mean & standard deviation by using a square with extending whiskers.



Sample mean +/- SD

kernel density estimate

**In ggplot2**

a = ggplot(Olives,aes(Area.Name,Oleic))

a + geom\_violin(fill="lightblue") + xlab("Area Name") +

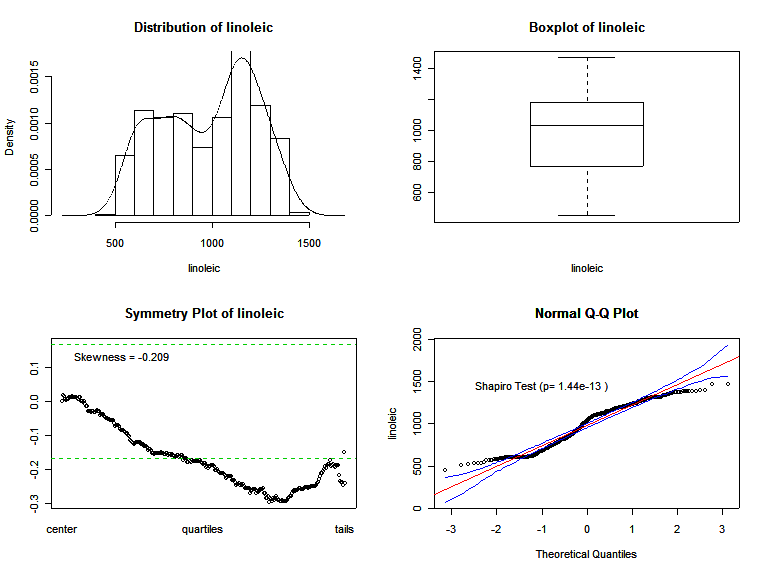
ylab("Oleic Acid") + ggtitle("Oleic Acid vs. Growing Area")

> detach(Olives)

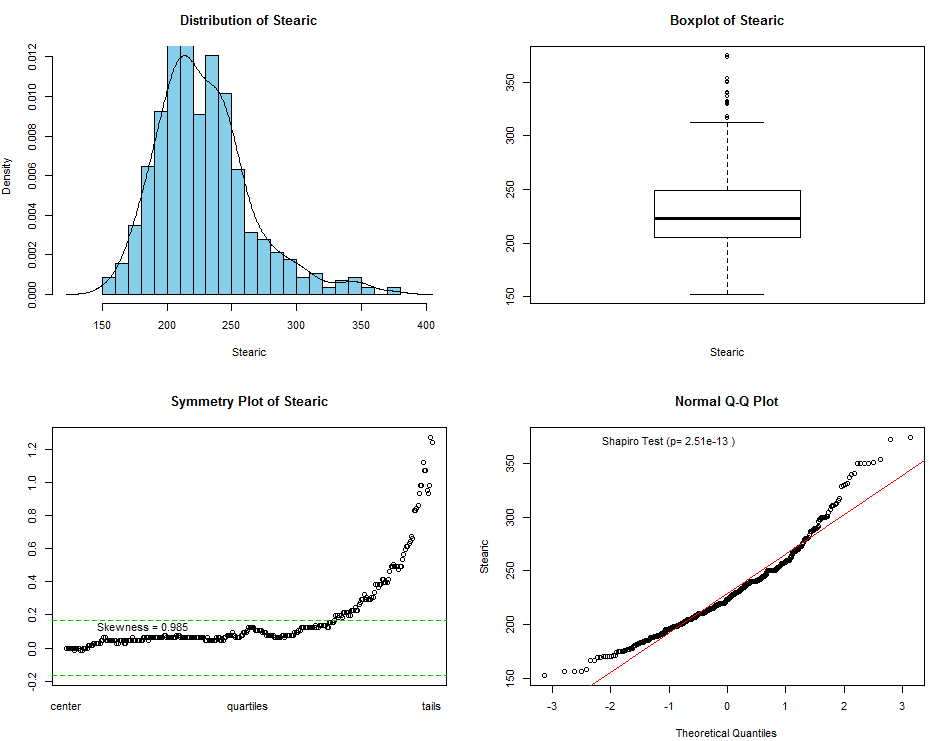
If you do choose to use the attach() function to allow you reference variables in a data frame directly (i.e. without using the data.frame$variable.name format) then it is VERY IMPORTANT that you detach the data frame when you are done using it. **In general it is best NOT to use attach, but rather use the data.frame$variable.name approach.**

**1.4 - More Univariate Displays**Statplot and Statplot2(Statplot2 - no simulated envelope on the qq-plot) are functions that I wrote to provide a series of univariate displays for a continuous variable.These functions return four univariate displays for continuous variables, a histogram with a kernel density estimate, univariate boxplot, symmetry plot, and a normal quantile plot. The normal quantile plot also gives the p-value from the Shapiro-Wilkes test for normality. If the reported p-value is less than 0.05 then we reject the null hypothesis that the distribution of the variable being consider is normally distributed or Gaussian.

Example 1.2: Italian Olive Oils (cont’d)  
  
> Statplot(Olives$Linoleic,xname=”linoleic”)



> Statplot2(Olives$Stearic,xname="Stearic Acid")



**1.5 – Displays for Categorical Data (Bar plots/graphs, Pie Charts, and Mosaic Plots)**Example 1.3: Gender Bias in Graduate School Admissions at UC-Berkley   
  
This example uses University of California – Berkeley graduate school admissions data for male and female students applying to six departments (labeled A,B,C,D,E,F). These data are contained in the data frame UCBAdmissions.

> UCBAdmissions

, , Dept = A

Gender

Admit Male Female

This is an example of array in R. An array is essentially a collection of matrices. This array is (the three dimensions are admittance, gender, and department applied to respectively). The numbers in the array represent the number of students in each category. For example the first value in the array is 512 which means that in department A there were a total of 512 male applicants admitted.

Admitted 512 89

Rejected 313 19

, , Dept = B

Gender

Admit Male Female

Admitted 353 17

Rejected 207 8

, , Dept = C

Gender

Admit Male Female

Admitted 120 202

Rejected 205 391

, , Dept = D

Gender

Admit Male Female

Admitted 138 131

Rejected 279 244

, , Dept = E

Gender

Admit Male Female

Admitted 53 94

Rejected 138 299

, , Dept = F

Gender

Admit Male Female

Admitted 22 24

Rejected 351 317

Adding across dimensions we can determine things like the total number of applicants to each department, the total number male and female applicants, etc.

**Barplot and Pie Chart** (univariate)

Let’s construct a simple bar plot or bar graph showing the number of total applicants to each department. To do this we first need to sum across the first two dimensions of the array, which is done using the command margin.table.

> DeptCount = margin.table(UCBAdmissions,3)

> DeptCount

Dept

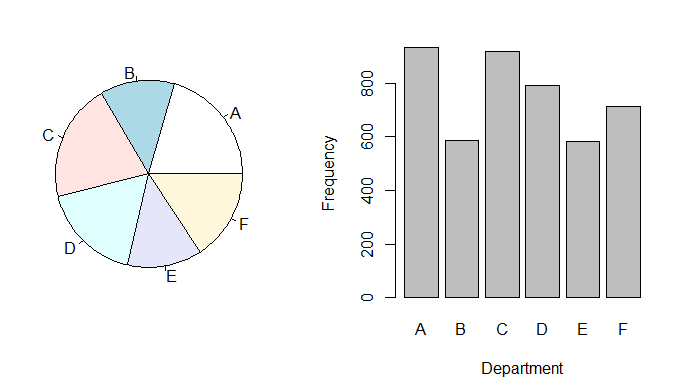
A B C D E F

933 585 918 792 584 714

> par(mfrow=c(1,2)) 🡨 sets up plotting region as 1 row and 2 columns of plots.

> pie(DeptCount)

> barplot(DeptCount,xlab="Department",ylab="Frequency")



We may want to look at admission rates by department and by gender of applicant. This can achieved by again summing across the appropriate dimensions and then constructing an appropriate plot.

> DeptAdmit = margin.table(UCBAdmissions,c(1,3))

> DeptAdmit

Dept

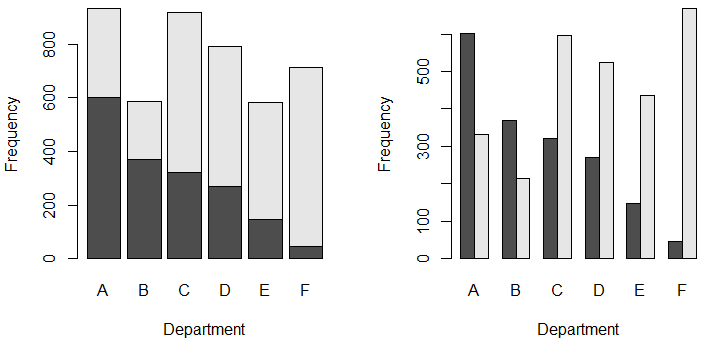
Admit A B C D E F

Admitted 601 370 322 269 147 46

Rejected 332 215 596 523 437 668

> barplot(DeptAdmit,xlab="Department",ylab="Frequency")

> barplot(DeptAdmit,xlab="Department",ylab="Frequency",beside=T)



> GenderAdmit = margin.table(UCBAdmissions,c(1,2))

> GenderAdmit

Gender

Admit Male Female

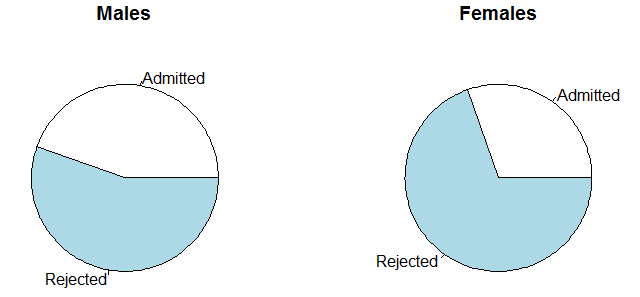
Admitted 1198 557

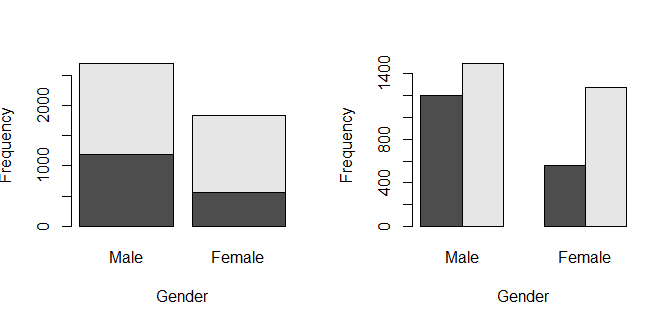
Rejected 1493 1278

> pie(GenderAdmit[,1],main="Males")

> pie(GenderAdmit[,2],main="Females")  
> barplot(GenderAdmit,xlab="Gender",ylab="Frequency")

> barplot(GenderAdmit,xlab="Gender",ylab="Frequency",beside=T)



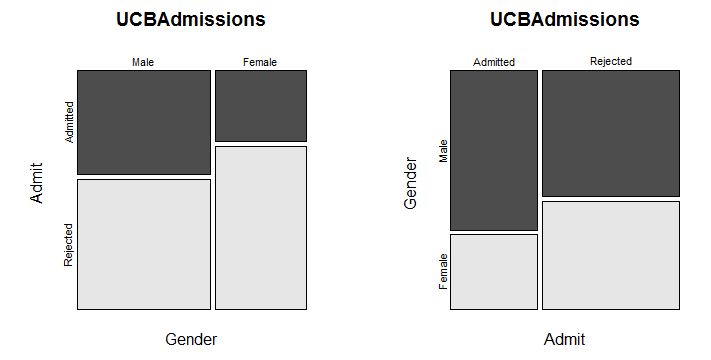


**Mosaic Plots** (visualizing two or more categorical variables simultaneously)

A mosaic plot is a variation on the stacked bar graph which allows us to examine the relationship between two or more categorical/ordinal variables. We first examine the relationship between admission rate vs. department and admission rate vs. gender. We will then use a mosaic plot to display all three dimensions simultaneously.

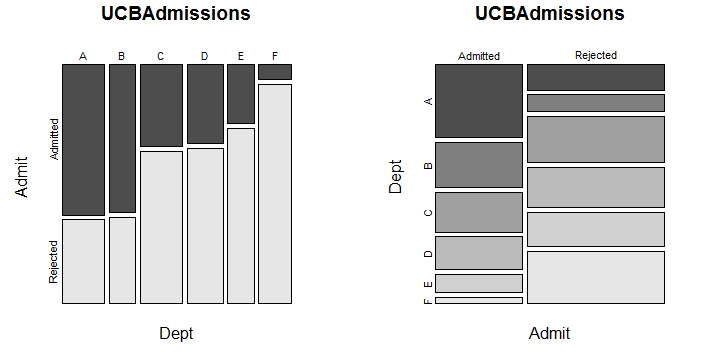
> mosaicplot(~Gender+Admit,data=UCBAdmissions,color=T)

> mosaicplot(~Admit+Gender,data=UCBAdmissions,color=T)

  
Which display is the appropriate display to consider if we are interested in determining if there is gender discrimination in the graduate school admissions at University of California-Berkeley?

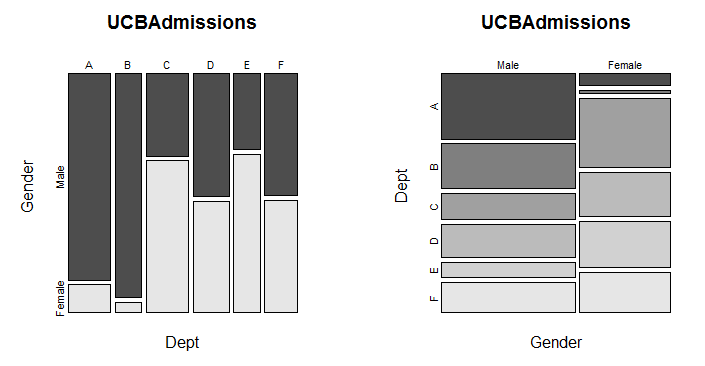
> mosaicplot(~Dept+Admit,data=UCBAdmissions,color=T)

> mosaicplot(~Admit+Dept,data=UCBAdmissions,color=T)



> mosaicplot(~Dept+Gender,data=UCBAdmissions,color=T)

> mosaicplot(~Gender+Dept,data=UCBAdmissions,color=T)

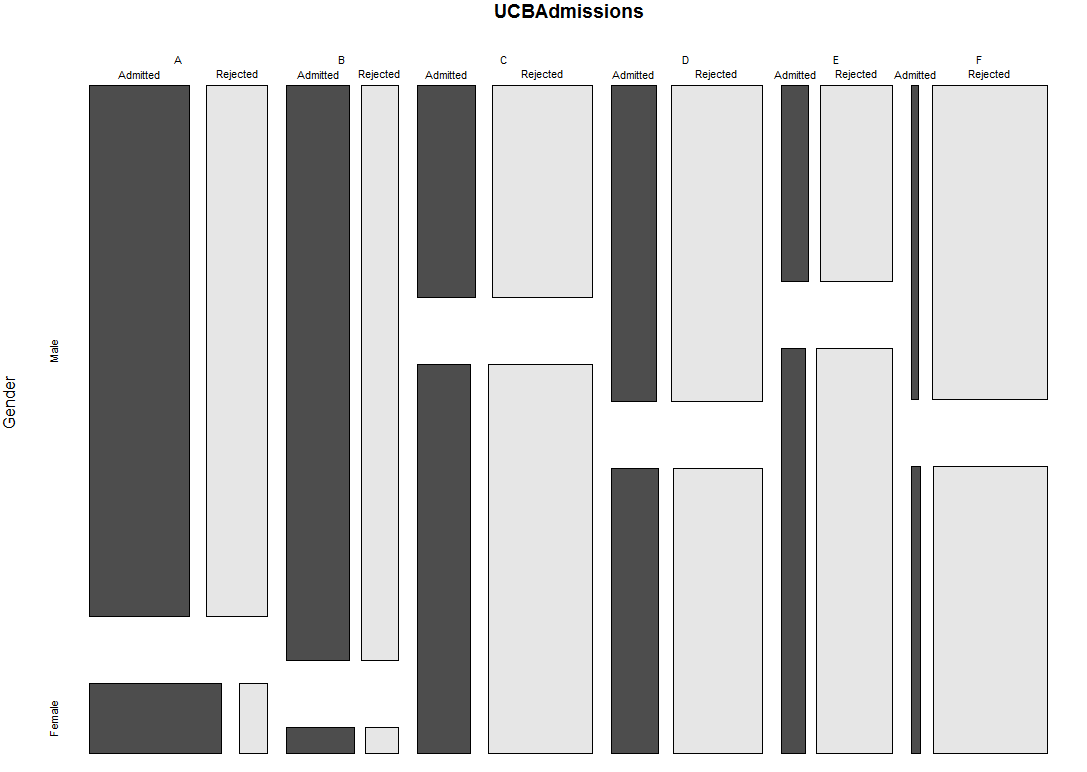


> par(mfrow=c(1,1))

This last command returns the plotting region to consisting of 1 row and 1 column of plots, i.e. a single plot.

To look at all dimensions simultaneously list all dimensions in the mosaic formula as demonstrated above. Be aware that ORDER matters a great deal here! Below is a reasonable ordering, given admittance is the key outcome of interest and gender bias is the issue.

> mosaicplot(~Dept+Gender+Admit,data=UCBAdmissions,color=T)



What does this mosaic plot suggest about admission bias against female applicants? Try changing the ordering of gender, admission status, and department in the function call to see how the resulting mosaic plot differs.

**1.6 - Scatterplot Matrices (with enhancements)**A scatterplot matrix is a multi-panel display plotting each pair variables vs. one another. The panels above and below the diagonal are essential the same plot with the exception that the horizontal () and vertical axis () variables are switched.  
Example 1.4: Italian Olive Oils (cont’d)  
> names(Olives)

[1] "Region.Name" "Area.Name" "Palmitic" "Palmitoleic" "Stearic" "Oleic" "Linoleic"

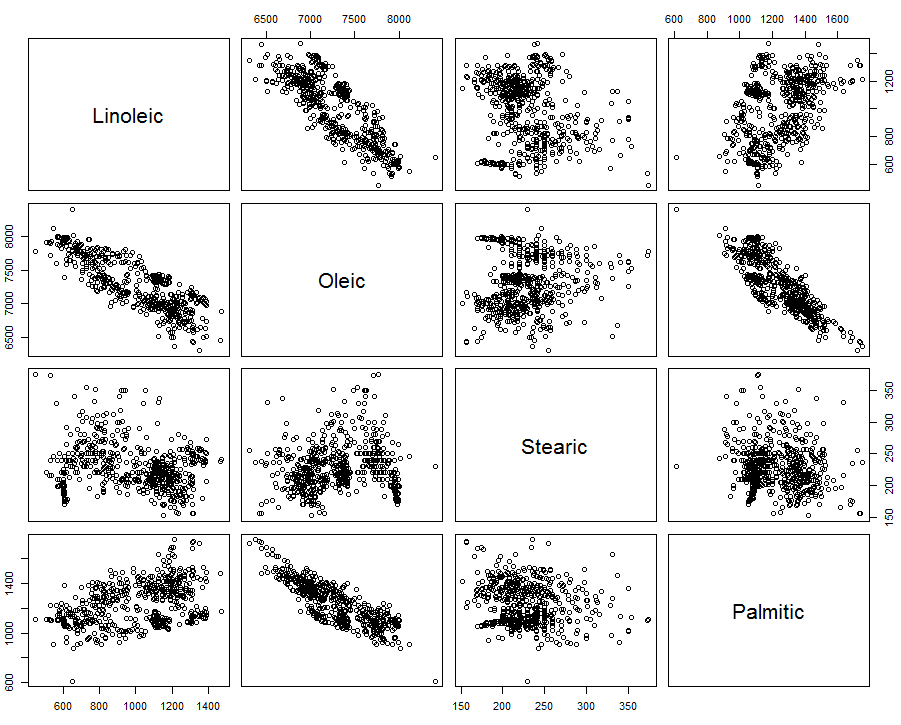
[8] "Eicosanoic" "Linolenic" "Eicosenoic"

> olive.mat <- Olives[,c(7,6,5,3)] # forms a matrix with these four acid levels forming the columns.

Alternatively we could form the matrix directly using the cbind command, which stands for column bind, i.e. bind variables together in columns to form a matrix.

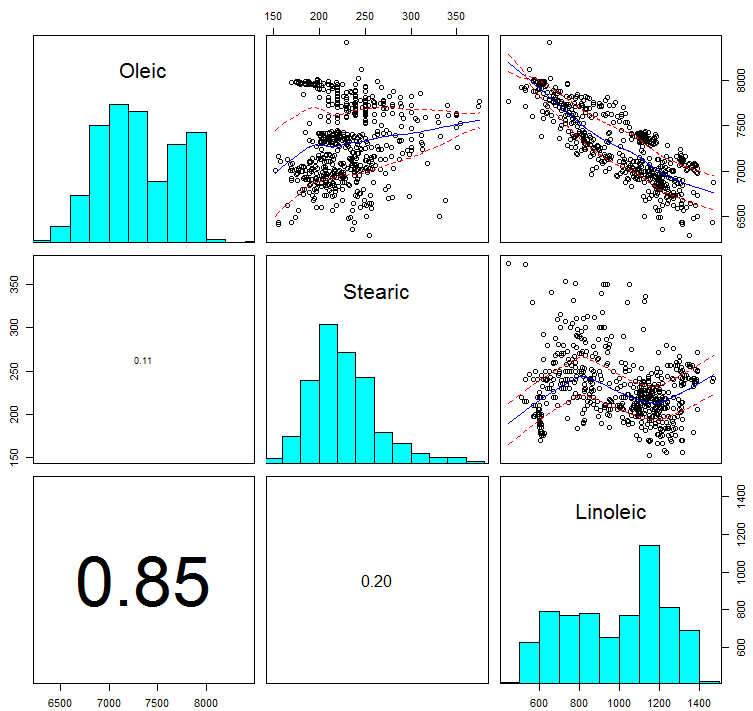
Attaching the data frame Olives can be done to alleviate the need for the $ notation.

> olive.mat = cbind(Olives$linoleic,Olives$oleic,Olives$strearic,Olives$palmitic)

The pairs command in R forms a scatterplot matrix of given matrix or data frame. The olive.mat matrix was formed using the first option presented above.  
  
> pairs(olive.mat) # basic function call, nothing fancy.  
  
  
We can create higher level plots by adding additional features to the scatterplot matrix. The function pairs.trendsd is shown below. It adds a lowess smooth with +/- SD lines, histograms for the variables, and pairwise correlations using their magnitude to control font size.

The complete code for this simple function is given below.

> pairs.trendsd <- function(data,...) {   
 pairs(data,lower.panel=panel.cor,upper.panel=panel.trendsd,  
 diag.panel=panel.hist,...)}

> pairs.trendsd(Olives[,c(6,5,7)])  


There are numerous packages on CRAN that have enhanced plotting methods. A couple packages that produce visualizations of the pairwise correlations between a set of numeric variables are corrgram and corrplot which are contained in the packages of the same name available from CRAN.   
  
> library(corrplot)  
> library(corrgram)

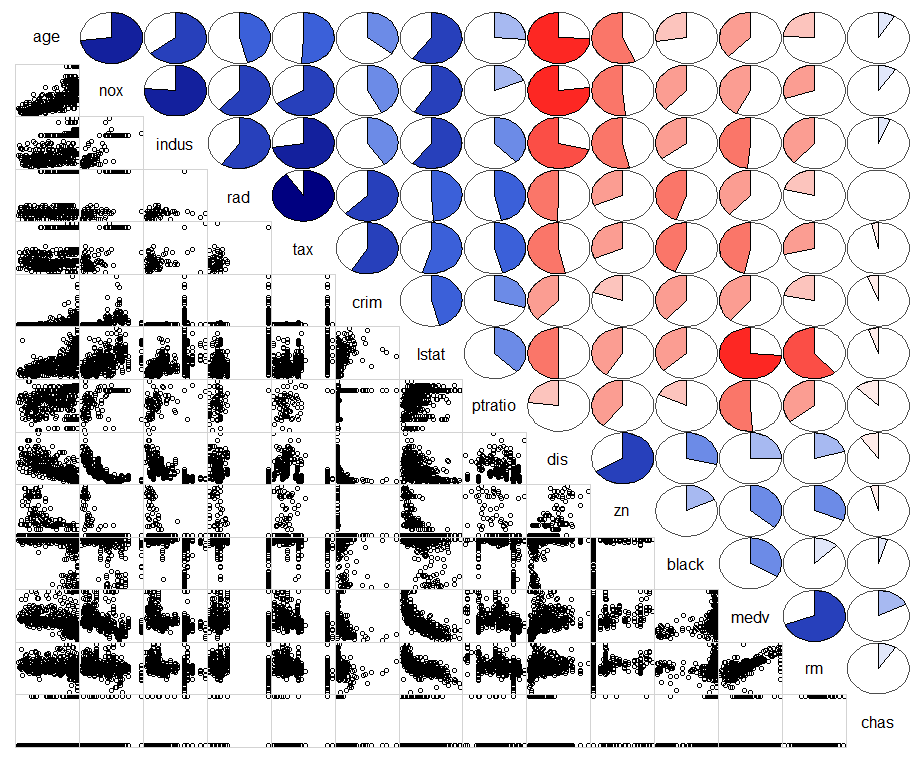
Example 1.4: Boston Housing Data  
Some data sets become famous, rightly or wrongly so, and the Boston Housing data is certainly one of these. These data come from a study of housing prices in census tracts in Boston, MA in 1972. The researchers were looking at the potential relationship between air pollution levels (measured by nitrogen oxide levels) and home prices. Other variables have to do with age of homes, percent of the census tract zoned for industrial use, crime indices, and distance from radial highways etc. We will use these data to examine results from the corrgram and corrplot function.  
   
> Boston = read.csv(file.choose()) # read in the file Boston.csv

> names(Boston)

[1] "age" "nox" "indus" "rad" "tax" "crim" "lstat" "ptratio" "dis" "zn"

[11] "black" "medv" "rm" "chas"

> corrgram(Boston,lower.panel=panel.pts,upper.panel=panel.pie)



Age = mean age of homes in census tract NOx = nitrogen oxide level in census tract

Indus = percent of census tract zoned industrial Rad = distance from tract centroid to radial hwy.

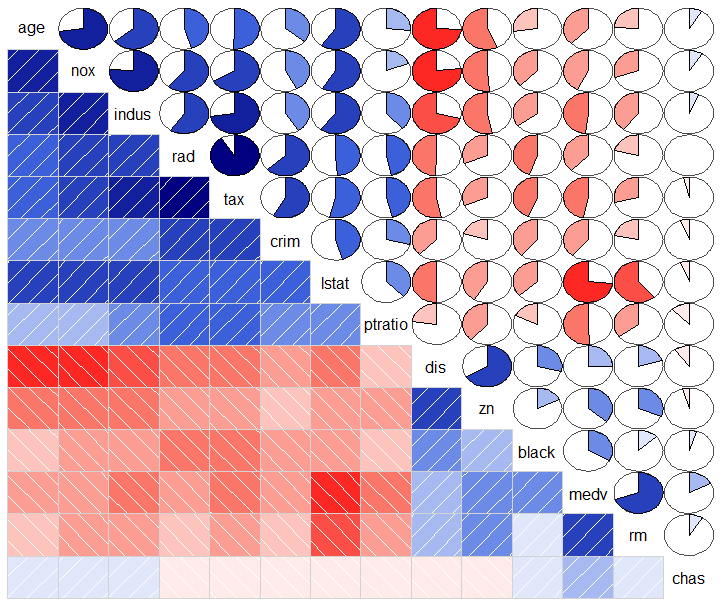
Tax = property tax Crim = crime rate in census tract.

Lstat = percent below poverty level in tract. PTratio = pupil/teacher ratio in tract.

Dis = distance from city center. Zn = percent zoned for lots over 25,000 sq.ft.

Black = measure based on percent blacks. Medv = median home value in tract.

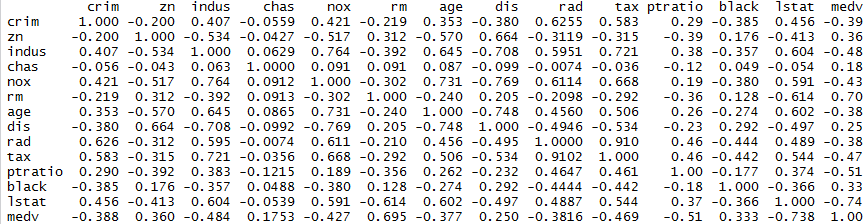
Rm = mean # of rooms per house Chas = is tract touching Charles River (1 = yes)?

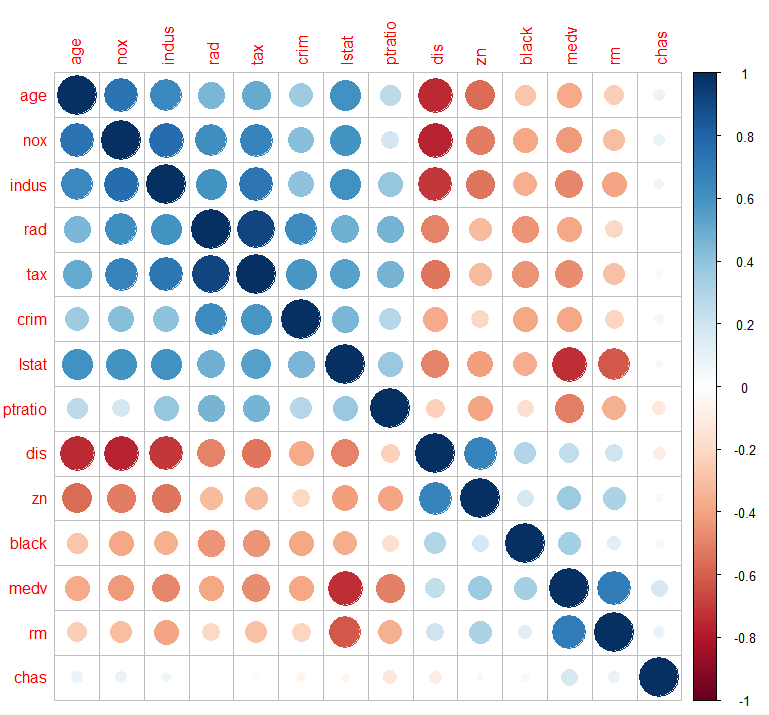
> corrgram(Boston,lower.panel=panel.shade,upper.panel=panel.pie)  


The corrplot function takes the pairwise correlations between the continuous variables in the data frame or matrix as the main argument, not the data itself.

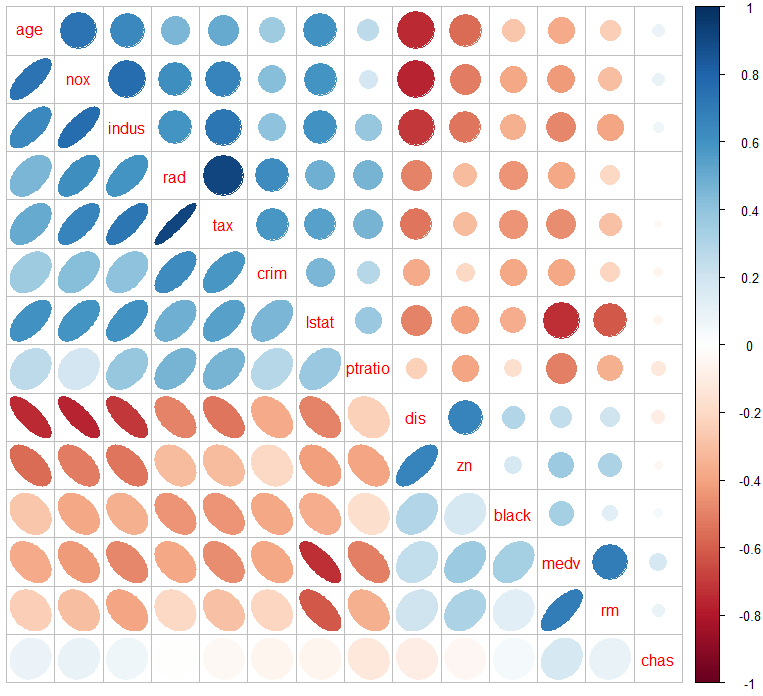
> Boston.corr = cor(Boston) # find pairwise correlations between all variables.  
> options(digits=2) # reduce the number of significant digits shown.

> Boston.corr

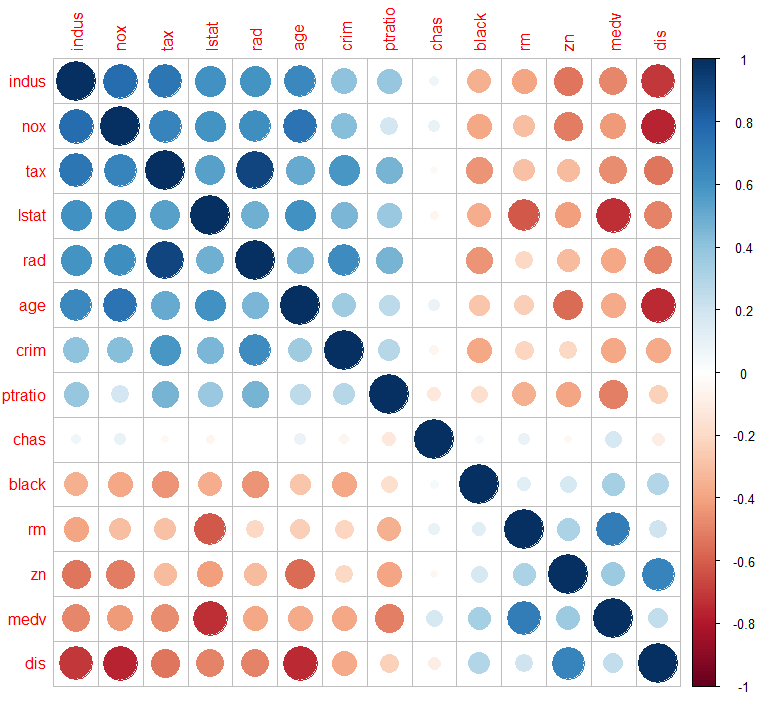
  
  
> corrplot(Boston.corr)



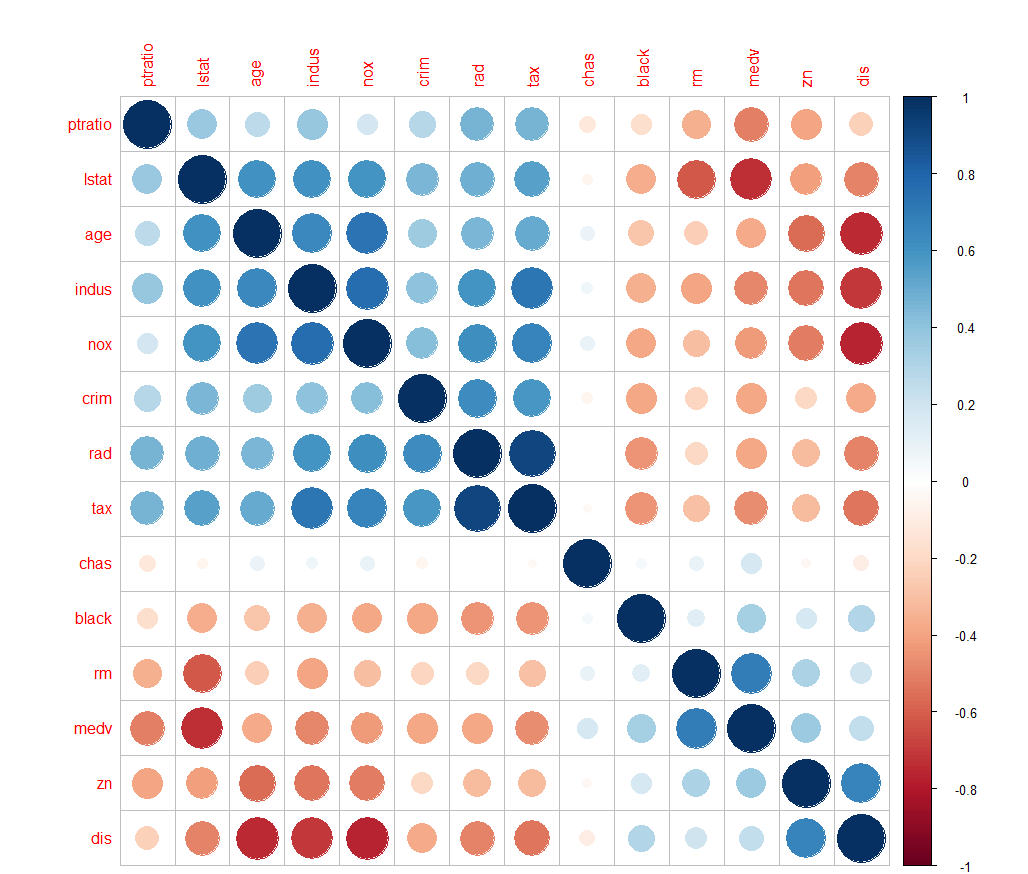
> corrplot(Boston.corr,method="ellipse") # not shown

> corrplot.mixed(Boston.corr,lower="ellipse",upper="circle")  


Variable reordering is very important for mining the hidden structure and pattern in a data matrix. There are four methods in corrplot (parameter order), named "AOE",

"FPC", "hclust", "alphabet". Of these hclust (Hierarchical Clustering) and FPC (first Principal Component) are the most useful and relate to topics covered later in the course.  
  
> corrplot(Boston.corr,order="FPC")  


> corrplot(Boston.corr,order=”hclust”)



**1.7 - Coplots or Trellised Graphs** (uses the package lattice)

We can create higher level scatterplot matrices using the splom command from the lattice library. The lattice contains numerous functions that allow for the creation of conditional plots or coplots. Conditional plots are basic plots like scatterplots, boxplots, histograms, etc. that are constructed separately for each level of a categorical factor. Some examples of conditional plots are shown below.

Example 1.5: Fisher’s Iris Data  
The next few examples use Fisher’s iris data which looks at four different flower measurements made on samples of three different iris species, Setosa, Versicolor, and Virginica.

> data(iris) # iris is data set included with base R

> attach(iris)

> names(iris)

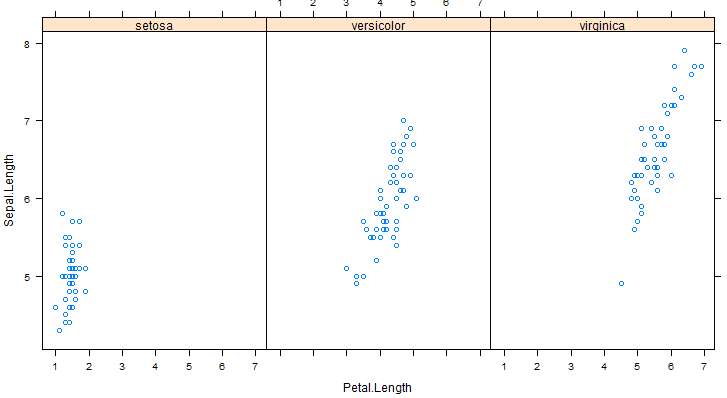
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"

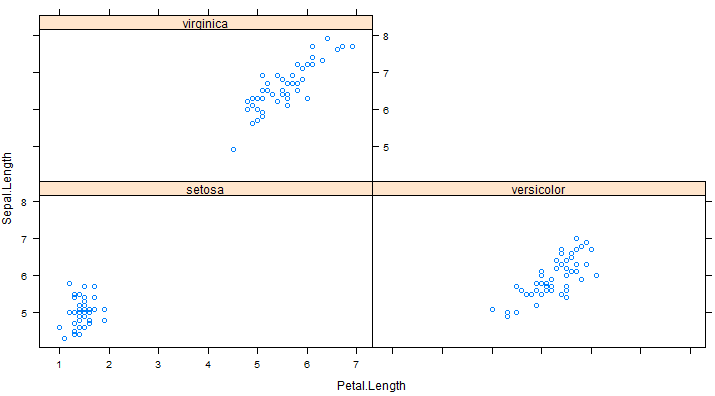
> library(lattice) # this package is included with base R I believe.

A basic trellis display can be plotting two characteristics, sepal length vs. petal length here, conditional on iris species. Trellis displays are created by using a formula to describe the form of conditioning plot. The formula is generally of the form

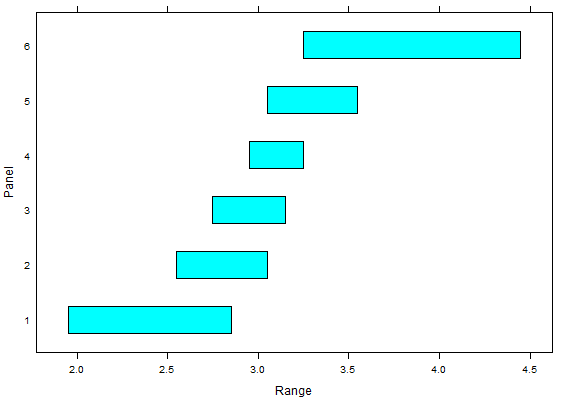
y ~ x | g1 \* g2

which indicates that plots of 'y' (on the y-axis) versus 'x' (on the x axis) should be produced conditional on the variables 'g1, g2, ...'. However, the conditioning variables 'g1, g2' may be omitted. The conditioning variables 'g1, g2' must be either factors (i.e. ordinal or nominal variables) or ***shingles***. Shingles are a way of processing numeric variables for use in conditioning. See documentation for the shingle command for details. Like factors, they have a `levels' attribute, which is used in producing the conditioning plots). For each unique combination of the levels of the conditioning variables 'g1, g2, ...', a separate panel is produced using the points '(x,y)' for the subset of the data defined by that combination.   
  
Numeric conditioning variables are converted to shingles by the function shingle however, using equal.count might be more appropriate in many cases, and character vectors are coerced to factors.

> xyplot(Sepal.Length ~ Petal.Length|Species)  
  


> xyplot(Sepal.Length~Petal.Length|Species,layout=c(2,2))  
  
  
We can also condition on a continuous variate by using equal.count to create *shingles*.

> SepWid <- equal.count(Sepal.Width)  
> plot(SepWid)



> print(SepWid)

Intervals:

min max count

1 1.95 2.85 47

2 2.55 3.05 64

3 2.75 3.15 61

4 2.95 3.25 50

5 3.05 3.55 48

6 3.25 4.25 42

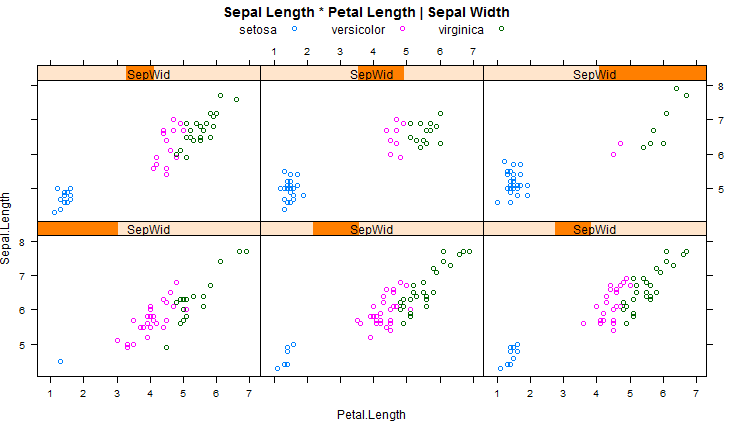
Overlap between adjacent intervals:

[1] 28 50 37 24 24

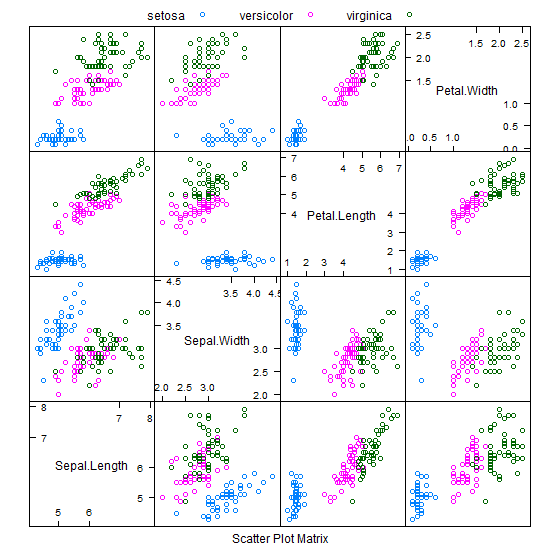
> xyplot(Sepal.Length ~ Petal.Length|SepWid)

A little fancier version uses different plotting colors each iris species, adds a title and key.

>xyplot(Sepal.Length~Petal.Length|SepWid,groups=Species,layout=c(3,2),  
 auto.key=list(columns=3),main="Sepal Length \* Petal Length | Sepal   
 Width")

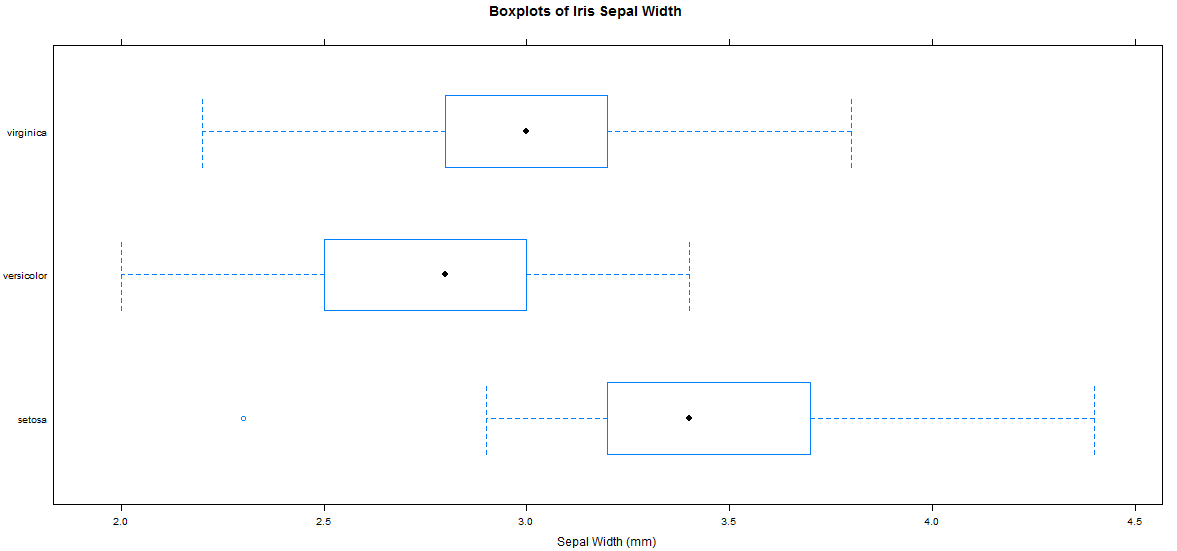


Scatterplot Matrix with Points Color-Coded by a Grouping Variable with Legend/Key

> splom(~iris[,1:4],groups=Species,auto.key=list(columns=3))  
  
The auto.key portion of the command is setting up a legend at the top of the scatterplot matrix using 3 columns to layout the legend.

**Boxplots in Lattice**

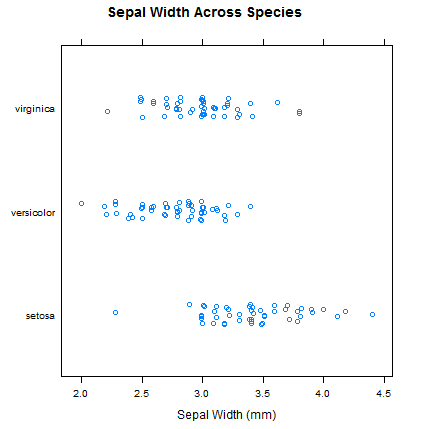
Basic Form: bwplot(GROUP ~ VARIATE)

> bwplot(Species~Sepal.Width,data=iris,xlab="Sepal Width (mm)",main="Boxplots of Iris Sepal Width")  


**Strip/Dot Plots**Basic Form: stripplot(GROUP ~ VARIATE)

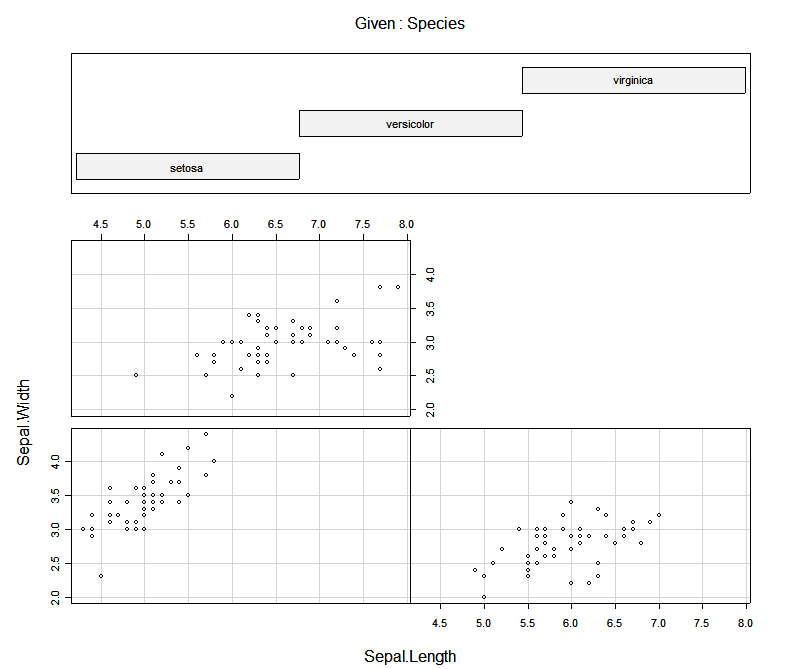
stripplot(Species ~ jitter(Sepal.Width), data = iris, aspect = 1,

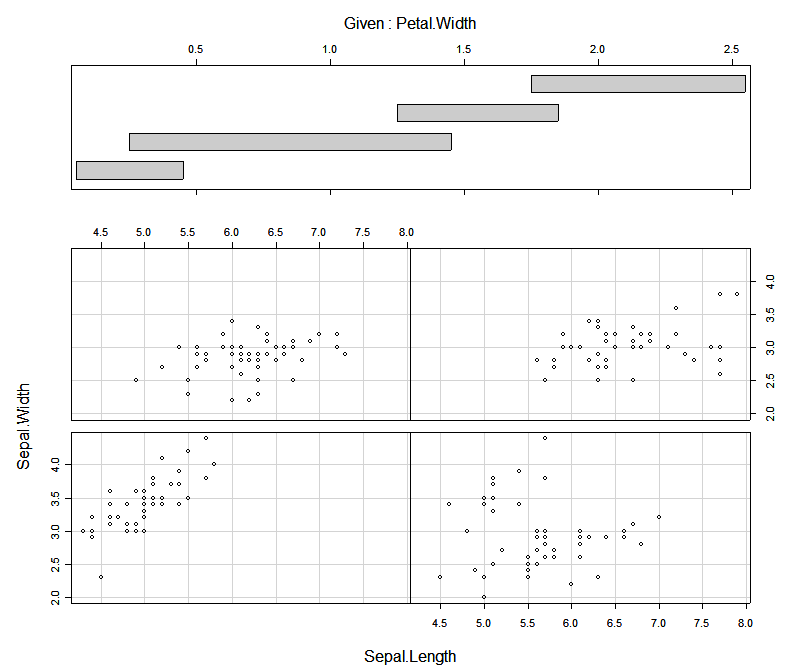
jitter=T,xlab="Sepal Width (mm)",main="Sepal Width Across Species")

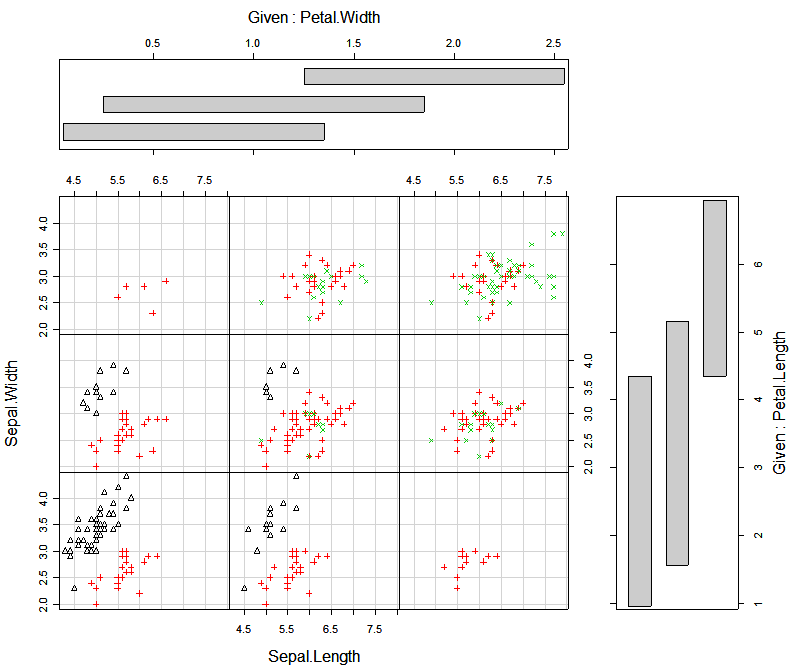


Another function that can be used to create conditional plots is the coplot function that is part of the R base package. Here are some examples of its use, again we start with Fisher’s iris data.

> coplot(Sepal.Width~Sepal.Length|Species)



> coplot(Sepal.Width~Sepal.Length|Petal.Width,number=4,overlap=.2)  


> coplot(Sepal.Width~Sepal.Length|Petal.Width\*Petal.Length,number=c(3,3),  
overlap=.5,col=as.numeric(Species),pch=as.numeric(Species)+1)  


In this last example different plotting symbols and colors are used to denote the iris species.

Can you construct similar types of lattice/trellis displays using the Italian olive oil data from Example 1.3?

As last example we consider ozone concentration data from the Los Angeles Basin. The data is contained in the data frame Ozdata.

> names(Ozdata)

[1] "day" "v500" "wind" "hum" "safb" "inbh" "dagg" "inbt" "vis" "upoz"

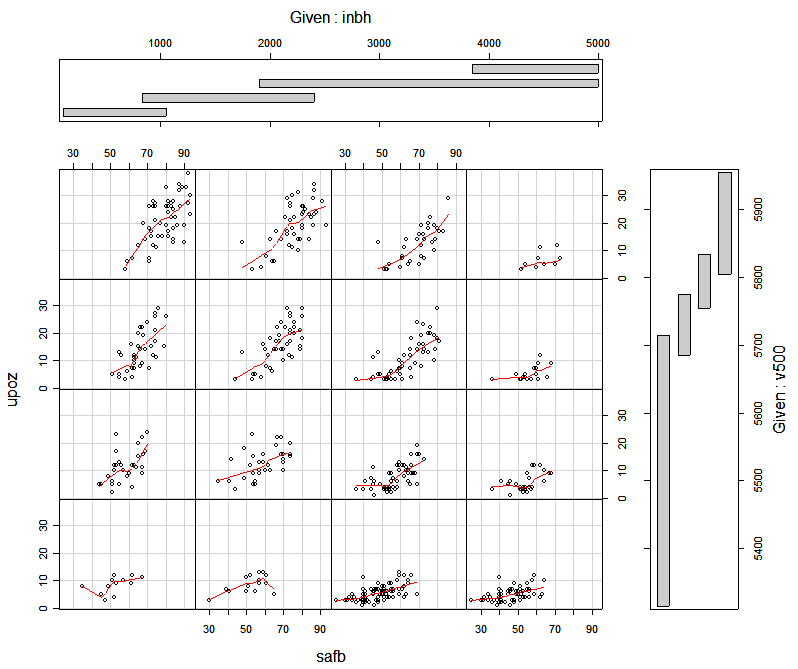
> attach(Ozdata)

> coplot(upoz~safb|inbh\*v500,panel=panel.smooth)

> coplot(upoz~safb|inbh\*v500,number=c(4,4),panel=panel.smooth)

> coplot(upoz~safb|inbh\*v500,number=c(4,4),overlap=.25,

+ panel=function(x,y,...) panel.smooth(x,y,span=.6,...))



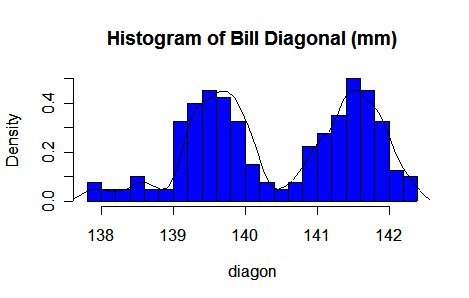
**1.8 – More on Density Estimates (**requires **library(ash))**The ash library contains functions for computing 1-dimensional and 2-dimensional density estimates using the average shifted histograms. Below you will find a couple of examples of their use.

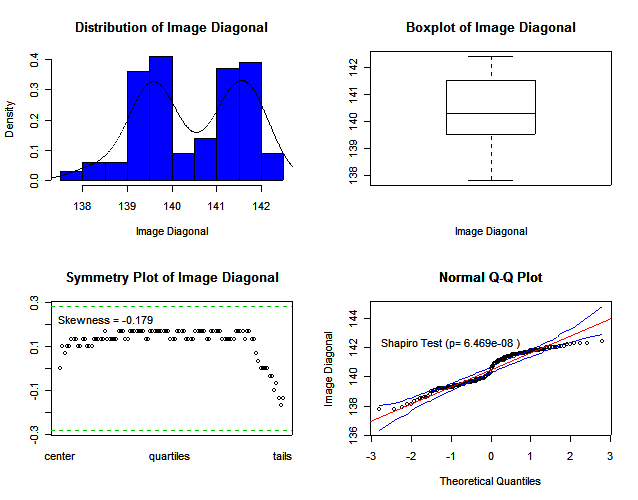
Example 1.6: Swiss Franc Data  
The examples use data from study of the differences between genuine and forged Swiss Francs. The variables represent different dimensional measurements made on the francs. There 100 genuine and 100 forged Swiss francs in the data set. Clearly interest would center on finding an easy way to distinguish forged bills from genuine bills. A few of the previously introduced functions will be demonstrated in this example as well.

**1-D ASH estimates**

> attach(Swiss)  
> library(ash)  
> names(Swiss)

[1] "id" "leng" "left" "right" "bottom" "top" "diagon" "genu"

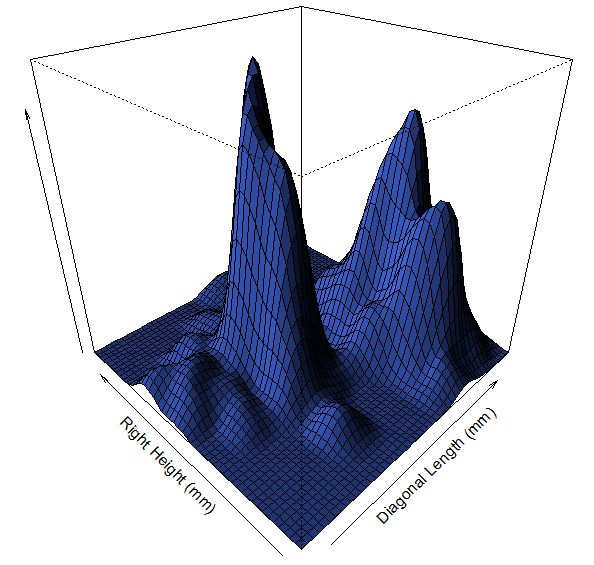
> d1 <- ash1(bin1(diagon,nbin=50),3)  
> hist(diagon,nclass=20,prob=T,col="blue",main="Histogram of Bill Diagonal (mm)")  
> lines(d1)  


> Statplot(diagon,xname="Image Diagonal")  


**2-D Density Estimate Viewed as Wire Frame Surface**  
> diagrt.bin <- bin2(cbind(diagon,right),nbin=c(50,50))

> diagrt.1 <- ash2(diagrt.bin,m=c(5,5))

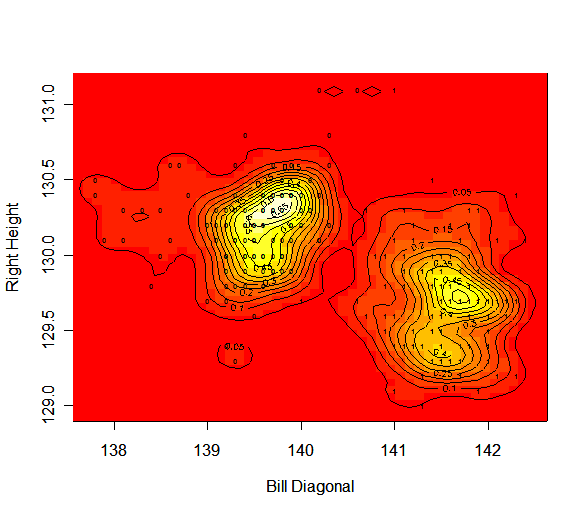
> persp(diagrt.1,xlab="Diagonal Length",ylab="Right Height",zlab="",cex=.5,theta=-45,phi=30,shade=1,col="royal blue")



**2-D Density Estimate Viewed as Image/Contour Map**

> image(diagrt.1,xlab="Bill Diagonal",ylab="Right Height")

> contour(diagrt.1,xlab="Bill Diagonal",ylab="Right Height",add=T)

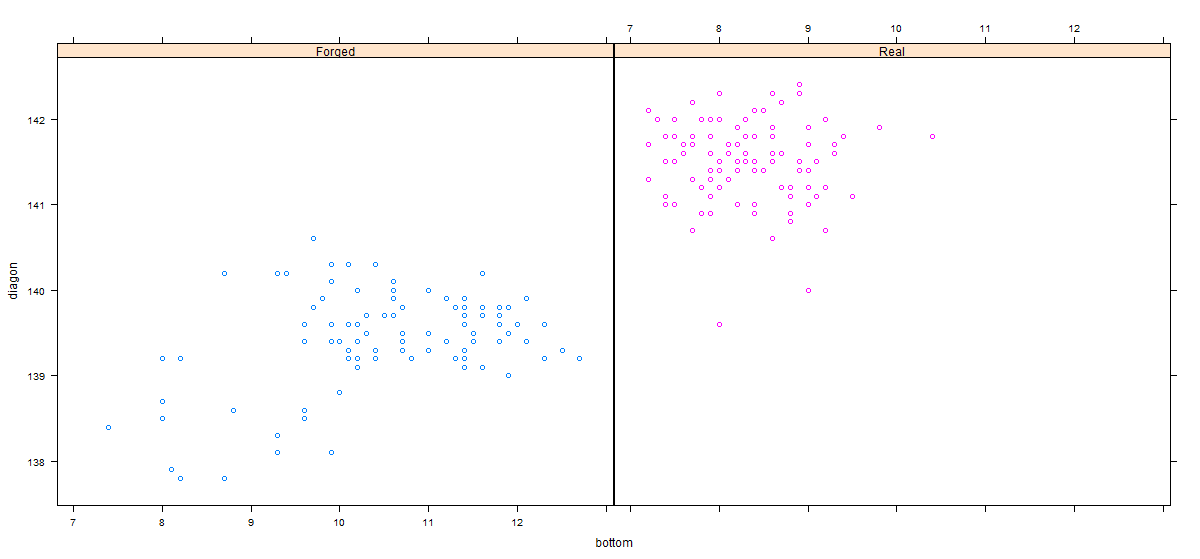
> points(diagon,right,pch=as.character(genu),cex=.4)  


**Trellis Display**

> Genuine <- genu

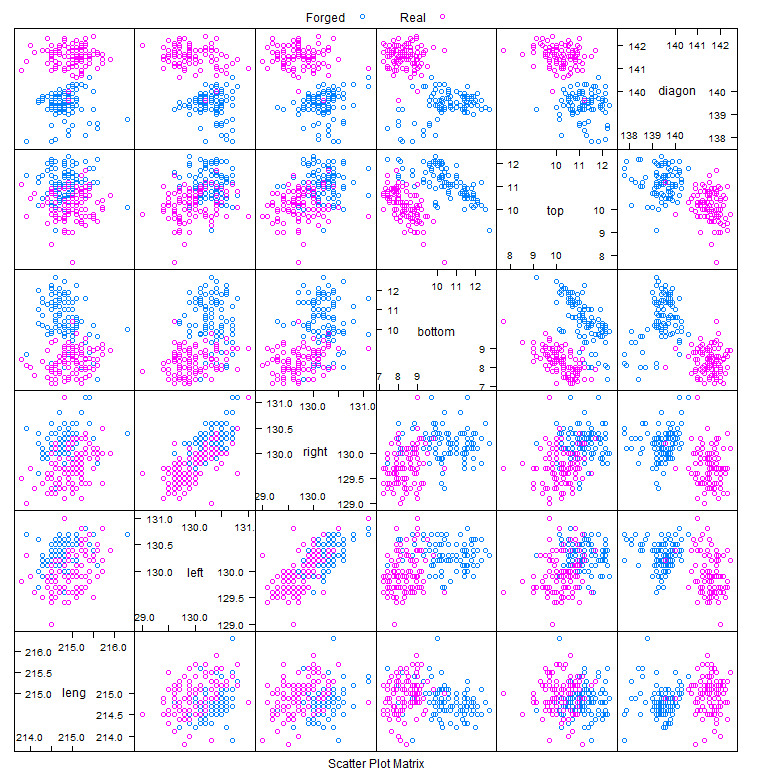
> Genuine[genu==0] <- "Forged"

> Genuine[genu==1] <- "Real"

> xyplot(diagon~bottom|Genuine,groups=genu)  


**Scatterplot Matrix Using splom**

splom(~Swiss[,2:7],groups=Genuine,auto.key=list(columns=2))



**Using Panel Functions with the Standard Scatterplot Matrix Command pairs**>pairs.image <- function(x) {

pairs(x,panel=function(x,y) {

foo <- bin2(cbind(x,y),nbin=c(75,75))

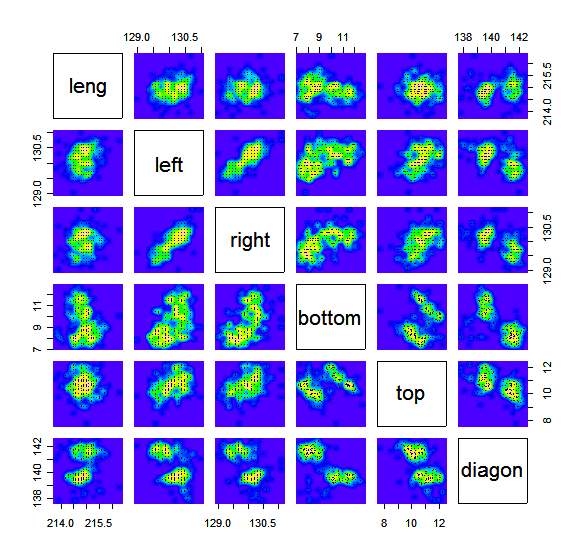
foo <- ash2(foo,m=c(6,6))

image(foo,add=T,xlab="",ylab="",col=topo.colors(1000))

points(x,y,pch=".")

})

}

> pairs.image(Swiss[,2:7])  


> pairs.persp <- function(x) {

par(bg="sky blue")

pairs(x,panel=function(x,y) {

foo <- bin2(cbind(x,y),nbin=c(75,75))

foo <- ash2(foo,m=c(8,8))

par(new=T)

persp(foo,xlab="",ylab="",theta=-45,phi=35,col="royal blue",

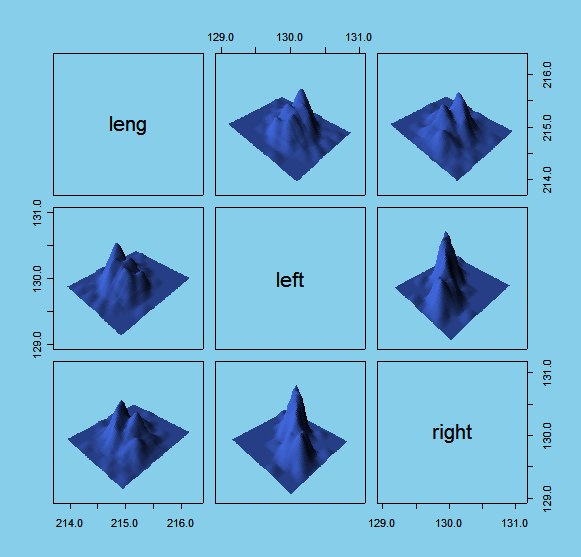
shade=.75,box=F,scale=F,border=NA,expand=.9)

})

par(new=F,bg="white")

}

> pairs.persp(cbind(leng,left,right))



**1.9 – Bubble or Symbol Plots**

The function symbols in R allows for the adding of symbols such as circles, squares, etc. to represent additional dimensions in a two-dimensional plot. In the examples below we focus on the use of circles to add a third dimension to scatter plots. The third dimension is added by placing circles with radii controlled by a designated numeric variable on top of an existing scatter plot. Other options such as thermometers and stars allow for adding information about three or more variables to a scatter plot, however these plots get much harder to read. See the symbols help file for details on how to use these options.

Example 1.7: NHL Team Statistics (Wins, Goals For, Goals Against)  
  
> attach(NHL)  
> names(NHL)

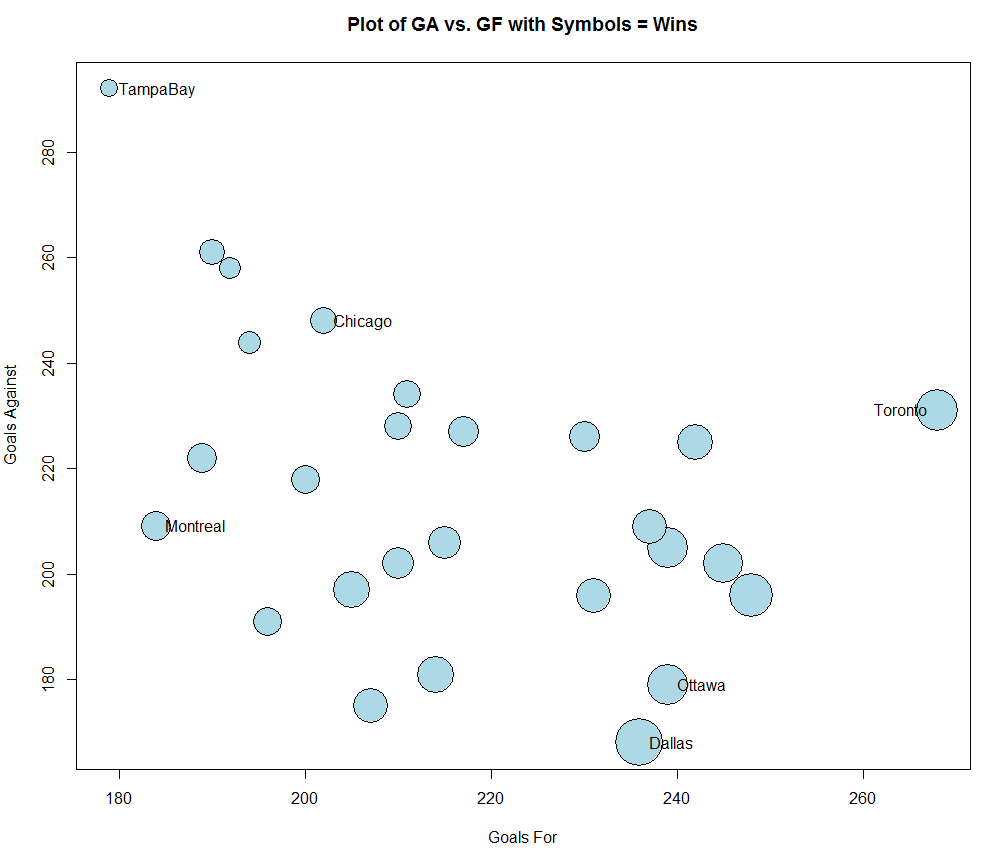
[1] "TEAM" "GP" "W" "L" "Ties" "GF" "GA" "PTS" "PCTG"

> plot(GF,GA,xlab="Goals For",ylab="Goals Against",main="Plot of GA vs. GF with Symbols = Wins")

> symbols(GF,GA,circles=W,add=T,inches=.25) # circles are too big if inches is not specified!

> symbols(GF,GA,circles=W,add=T,bg="lightblue",inches=.25) # bg option colors circles light blue

> identify(GF,GA,labels=row.names(NHL))



Example 1.8: King Crab Data from Kodiak Island, Alaska

As a second example consider these data that come from a multi-year study of the Kodiak Island King Crab catches in Alaska. It is all saved in a workspace or R directory called crab.

> attach(kodiak.crab) # This file contains latitude and longitude of the island coastline.

> attach(survey.crab) # This file contains crab counts by type, year, and location (lat, long).

> names(survey.crab)

[1] "year" "dist" "stat" "npots" "lat" "long"

[7] "pre4" "pre3" "pre2" "pre1" "nrec" "postrec"

[13] "juvfem" "adultfem"

Suppose we wish to visualize the density of crab catches (crabs/pot) by “station” location. A station is area where several pots are placed. The location of a station is recorded by the “average” latitude and longitude of the individual pots. Here we will add circles whose radius is proportional to some measure of crab density (e.g. # of crabs of a specified type caught per pot). Since the data comes from a number of study years it would probably be best to focus on a single year or a small range of study years at time. It should be noted that the station locations change somewhat from one year to the next.

> table(year) # what study years are included? 1973 - 1986

year

73 74 75 76 77 78 79 80 81 82 83 84 85 86

236 239 188 245 232 75 288 281 242 245 331 414 180 254

Here we plot the station location with circles proportional to the number of recruit (sexually mature) crabs per pot.

> plot(long[year==83],lat[year==83],pch=".",col="blue",

xlab="Longitude",ylab="Latitude",main="Circles Proportional To Number of Recruits Per Pot") # plot station locations with axes labels and title

> points(longit,latit,pch=".") # add the island for reference

> symbols(long[year==83],lat[year==83],circles=  
nrec[year==83]/npots[year==83],inches=.25,bg="blue",add=T)

Obviously this pretty tedious! If we wanted to frequently change the year and density measure this could become very cumbersome. The function below allows easily produced graphs of this type that take the year and the density measure (y) as arguments.

> crab.plot <- function(lat,long,y,year,yname=deparse(substitute(y))){

symbols(long[year==year],lat[year==year],circles=y[year==year],inches=.1,bg="blue",xlab="Longitude",ylab="Latitude",

main=paste("Circles Proportional to",yname))

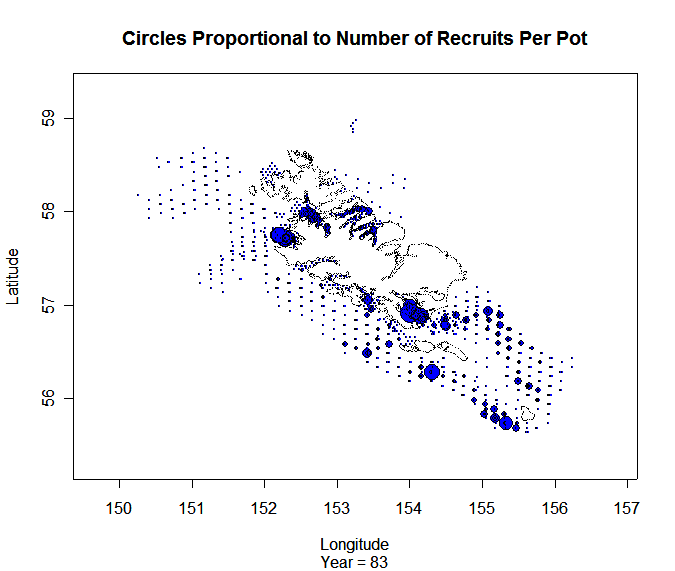
title(sub= paste("Year =",year))

points(kodiak.crab$longit,kodiak.crab$latit,pch=".")

}

> nr.pp <- nrec/npots # form density measure = num. of recruits per pot  
> crab.plot(lat,long,y=nr.pp,year=83,yname=”Number of Recruits Per Pot”)

# this function call achieves the same thing as the 1983 specific commands shown above using the crab.plot function.



> detach(survey.crab)

> detach(kodiak.crab)

**1.10 – More Multivariate Displays (Starplots, Parallel Coordinate Plots, Faces)**

**Starplots**  
Starplots and parallel coordinate plots are graphical methods for displaying multivariate data.  There are others, such as Chernoff faces, but they have received less attention and quite frankly are less useful. R does not have a built-in utility for creating either star plots or parallel coordinate plots, rather they are available in packages available from CRAN. The package gclus contains functions for creating both star plots and parallel coordinate plots. The package MASS, which comes with the base R installation also contains a function for creating parallel coordinate plots.

After loading both gclus and MASS you should be able to do work through the following examples.In R you can get help and see examples of a commands use by typing ?*function name*. For example, for the star plot command we type:

> ?stars

Example 1.1: U.S. Cities (cont’d)  
Using the data from the 77 largest U.S. cities in 1990 we will now consider some examples of starplots.

> attach(City) # This command readies the data frame City for analysis.

> names(City)  # The names command lists the names of the variables in a data set.

You can see the entire contents of this data frame by using the View command.

> View(City)

Because the function stars takes a matrix as its main argument we will need to first form a matrix that consists of the variables in our data frame that we wish to plot. The command cbind, which stands for column bind, is used to create a data matrix in R. Here will examine population density, percent of population that is black, percent of population that is Hispanic, median income, percent of city residents who receive welfare, percent of population living at or below poverty level, infant mortality rate (deaths per 100,000 births), and percent of population unemployed. Because we have attached the City data frame we can refer to the variable names directly.

> city.mat <-cbind(popdens,pct.black,pct.hisp,income,welfare,

poverty,infmort,unemprt)

> stars(city.mat, key.loc = c(15,1.25),main = "Starplots for U.S.

Cities",label=row.names(City),cex=.7)   
   
The above command produces basic starplots for the U.S. cities using the variables chosen. The location of the key is chosen by specifying x- and y-coordinates. I would use values similar to those used in the example, but may have to be tweaked a little.

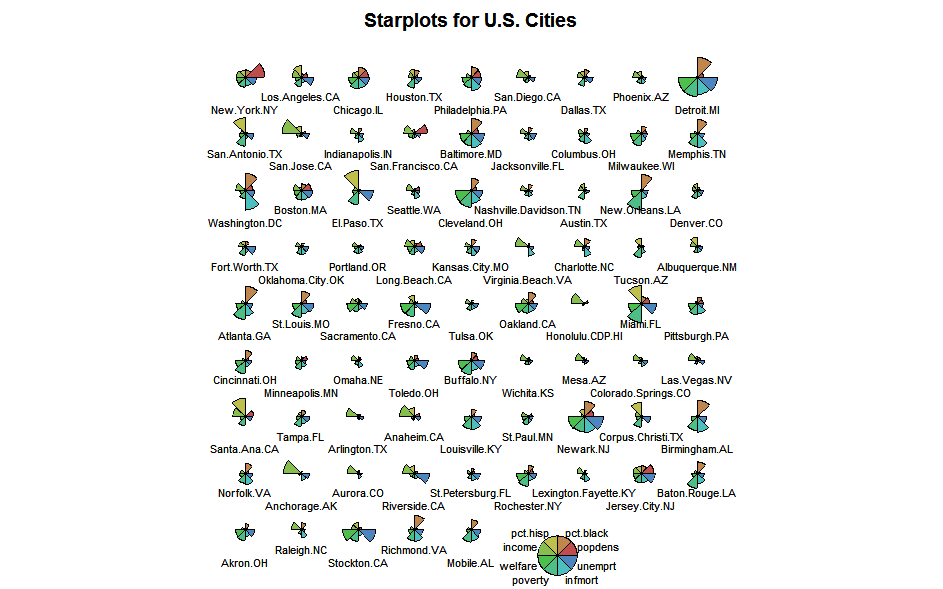
The optional argument label=row.names(City) specifies that we want the name of the city to appear below each star. If a data frame has names associated with the cases, as the City data frame does, the command row.names will extract them. The optional argument cex = .7 shrinks the font size to 70% of default. You will see the argument cex is used in lots function calls where text will be added to a plot.

> palette(rainbow(12, s = 0.6, v = 0.75))

> stars(city.mat,len=.80,key.loc = c(15,1.25),main = "Starplots for

U.S. Cities",draw.segments = TRUE,label=row.names(City),cex=.7)

> palette(“default”)



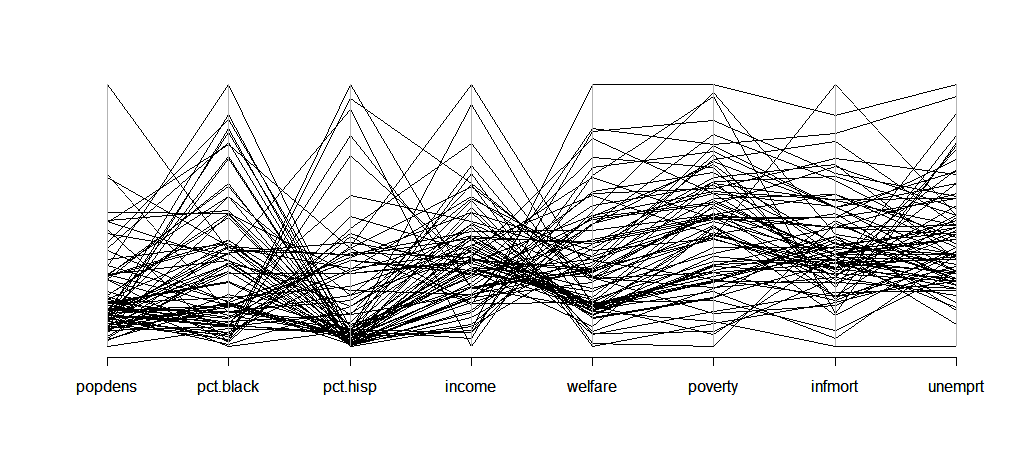
Some Observations:

Detroit, Baltimore, Newark, Cleveland have very similar shaped glyphs with relatively high values for welfare, poverty, infant mortality, and unemployment rate. Washington D.C. has a particularly high infant mortality rate. San Antonio, El Paso, Miami, and Santa Ana have large Hispanic populations. Anchorage and San Jose have particularly large median incomes and New York has a very large population density relative to the other cities. Etc...

**Parallel Coordinate Plots** (parcoord function in MASS library)

Parallel coordinate plots give a “profile” of variable values for each observation in a data set. The variables are scaled so values on each variable are comparable (e.g. z-scores) and line segments are used to connect values for each variable in the data set.

> parcoord(city.mat)



These plots are not very useful unless you can interactively click on profiles and see case labels. They can be useful however when you have data which comes from separate populations or groups. As an example consider the Italian Olive Oil data which we will be seeing throughout the semester.

Example 1.3: Italian Olive Oils

> names(Olives)

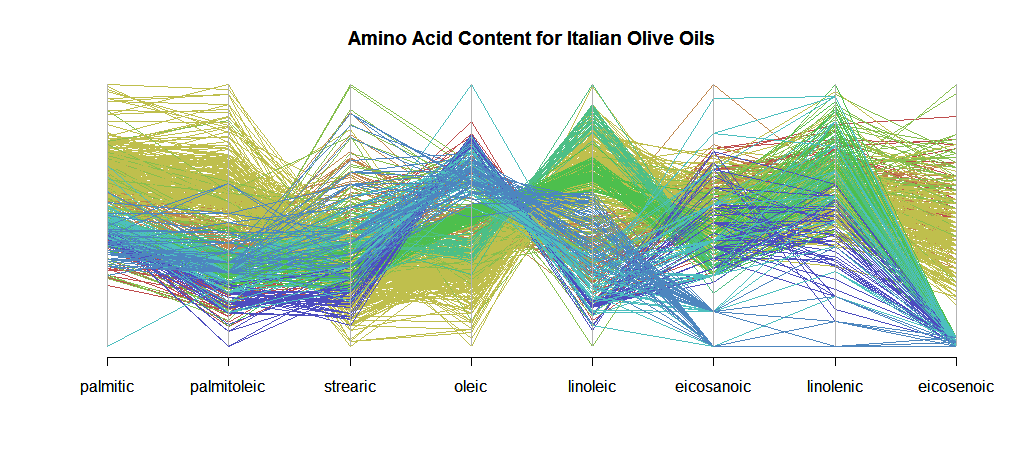
[1] "Region.name" "Area.name" "Region" "Area" "palmitic" "palmitoleic" "strearic"

[8] "oleic" "linoleic" "eicosanoic" "linolenic" "eicosenoic"

> olive.mat <- Olives[,5:12]

> palette(rainbow(12, s = 0.6, v = 0.75))

> parcoord(olive.mat,col=as.numeric(Area))



Here colors are used to denote the area of Italy the olive oil came from. It appears that oils grown in the same areas of Italy have similar amino acid concentration profiles.

> table(Area.name)

> table(Area.name)

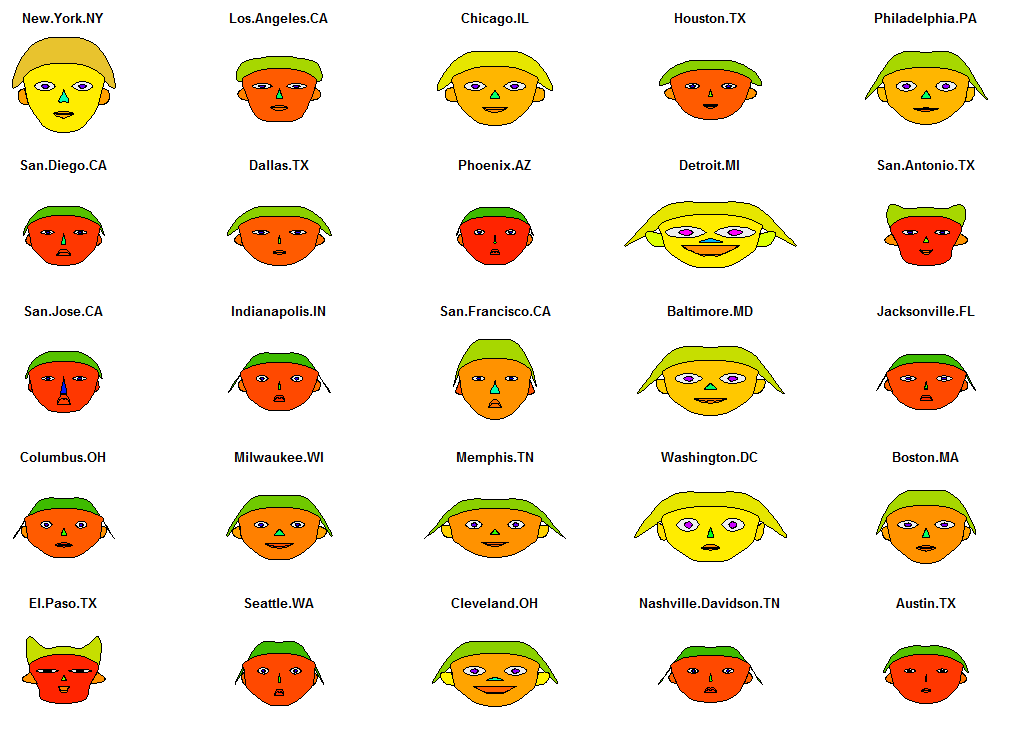
  
> parcoord(olive.mat,col=as.numeric(Area.name)) # produces the same result as above

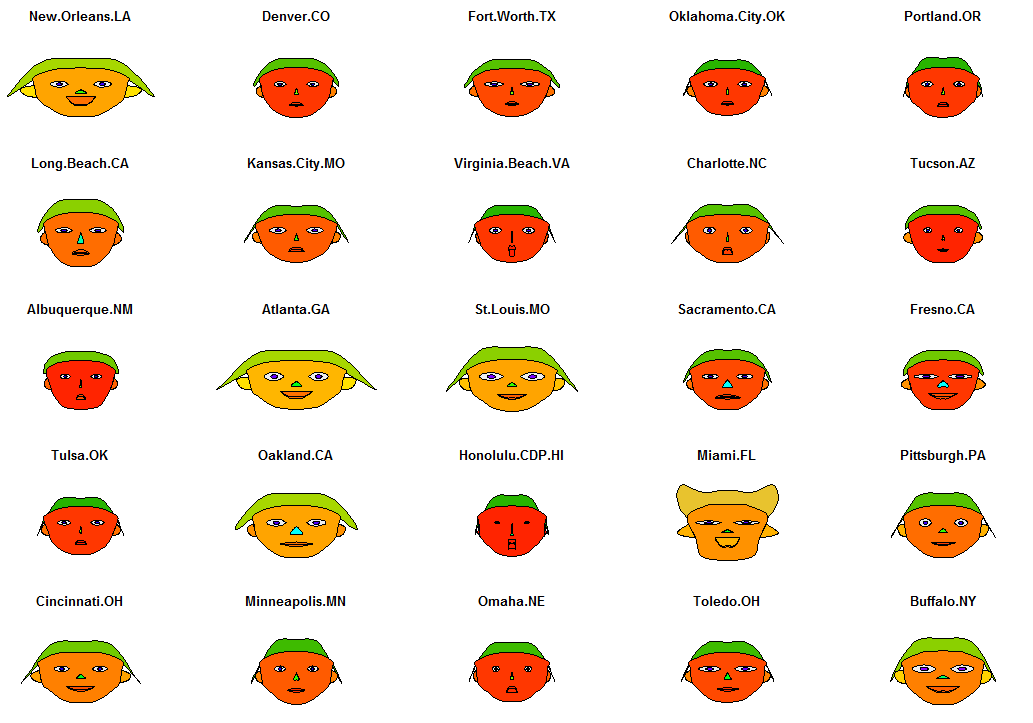
**Chernoff Faces** (requires library aplpack)

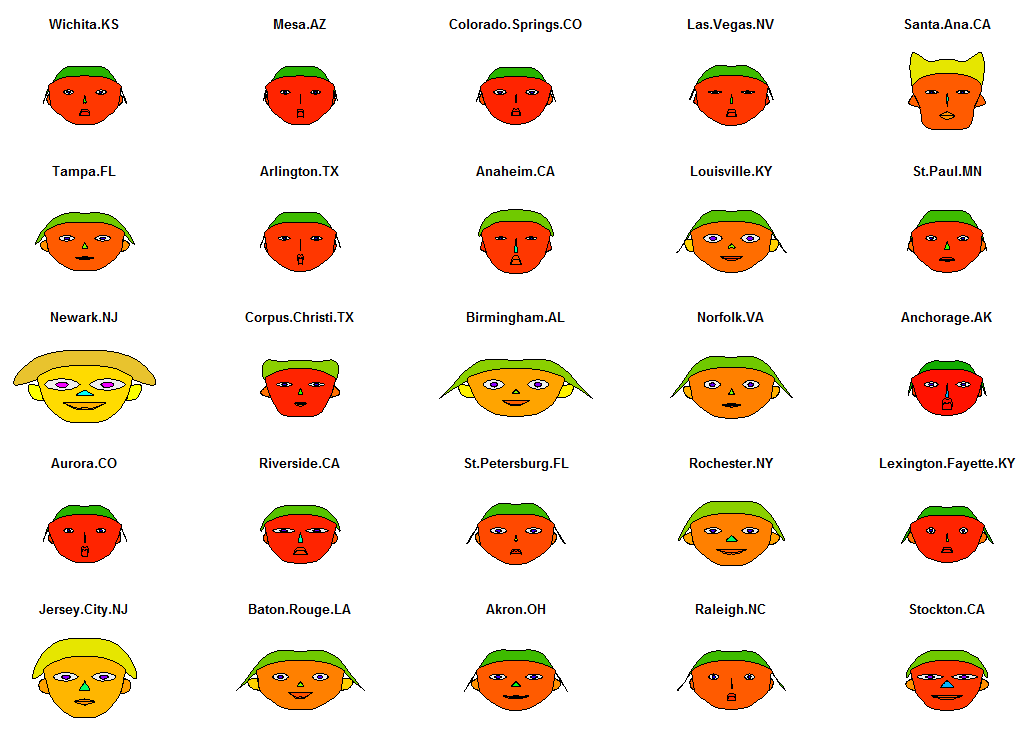
The function faces produces Chernoff faces from a supplied data matrix. Because they are larger than star plots we need to limit the number of faces per page by specifying the number of rows and columns of faces per window. Specifying the par(ask=T)prompts the user to hit enter before displaying the next page of faces.

> par(ask=T) # hit enter to see next plot

> faces(city.mat,nrow.plot=5,ncol.plot=5,labels=as.character(row.names(City)))









effect of variables:

modified item Var

"height of face " "popdens"

"width of face " "pct.black"

"structure of face" "pct.hisp"

"height of mouth " "income"

"width of mouth " "welfare"

"smiling " "poverty" 🡨 THIS MAKES NO SENSE!

"height of eyes " "infmort"

"width of eyes " "unemprt"

"height of hair " "popdens"

"width of hair " "pct.black"

"style of hair " "pct.hisp"

"height of nose " "income"

"width of nose " "welfare"

"width of ear " "poverty"

"height of ear " "infmort"

> par(ask=F) # toggle the ask feature back to default = FALSE

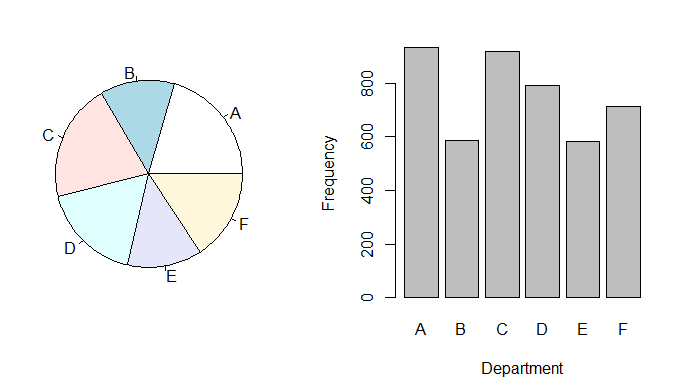
Clearly we need to consider putting variables in the matrix in a different order, cities with high poverty rates (e.g. Detroit) should not be smiling broadly. We can also see that we could have selected additional variables to control more face features. Once the list of variables provided is exhausted, it repeats.

**1.11 - General Guidelines for Graphical Displays of Data**

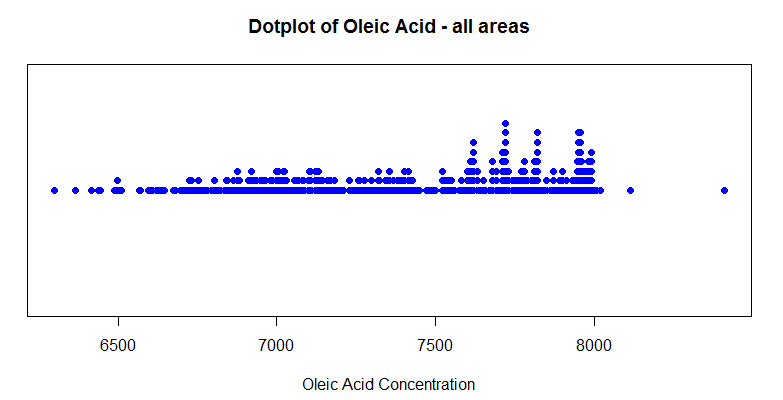
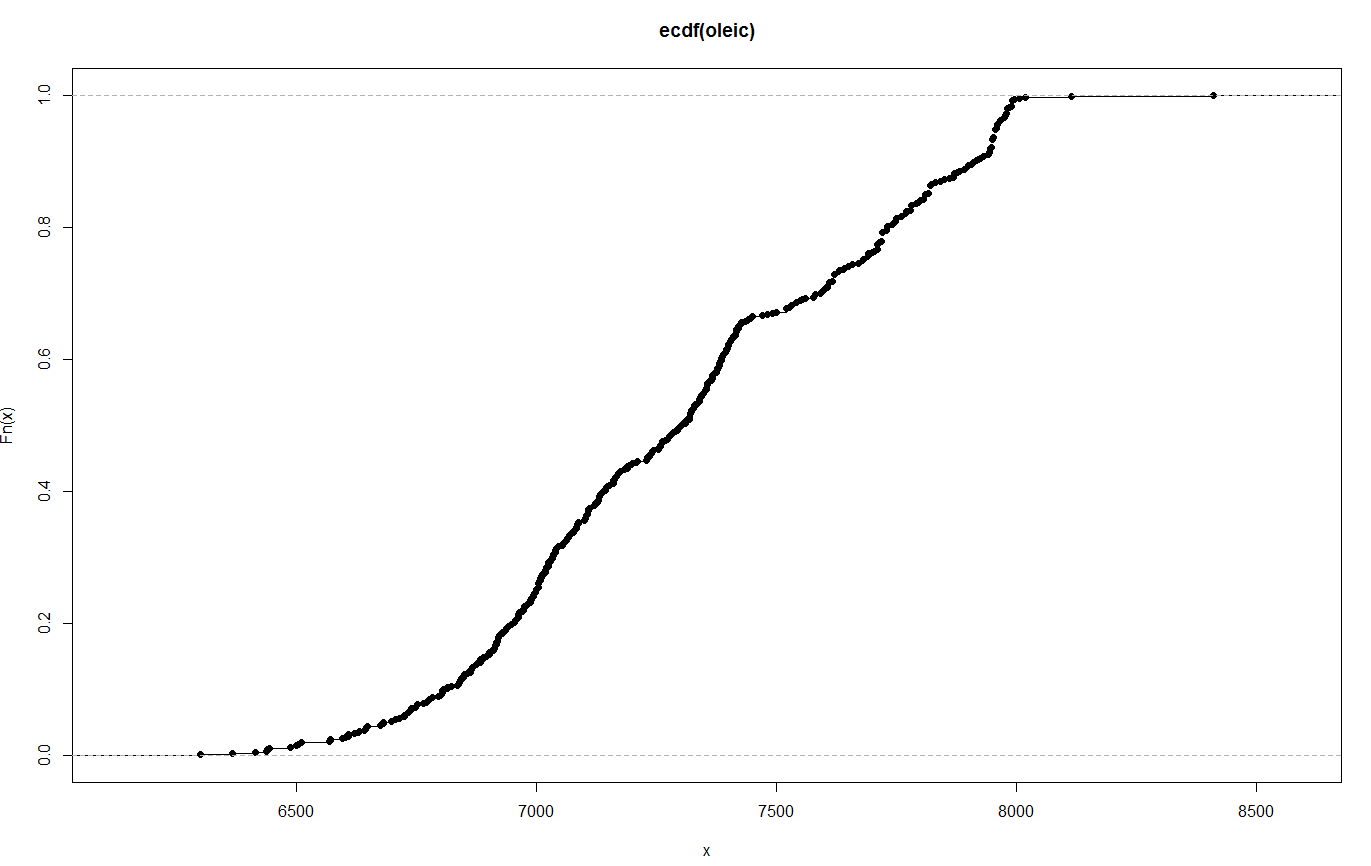
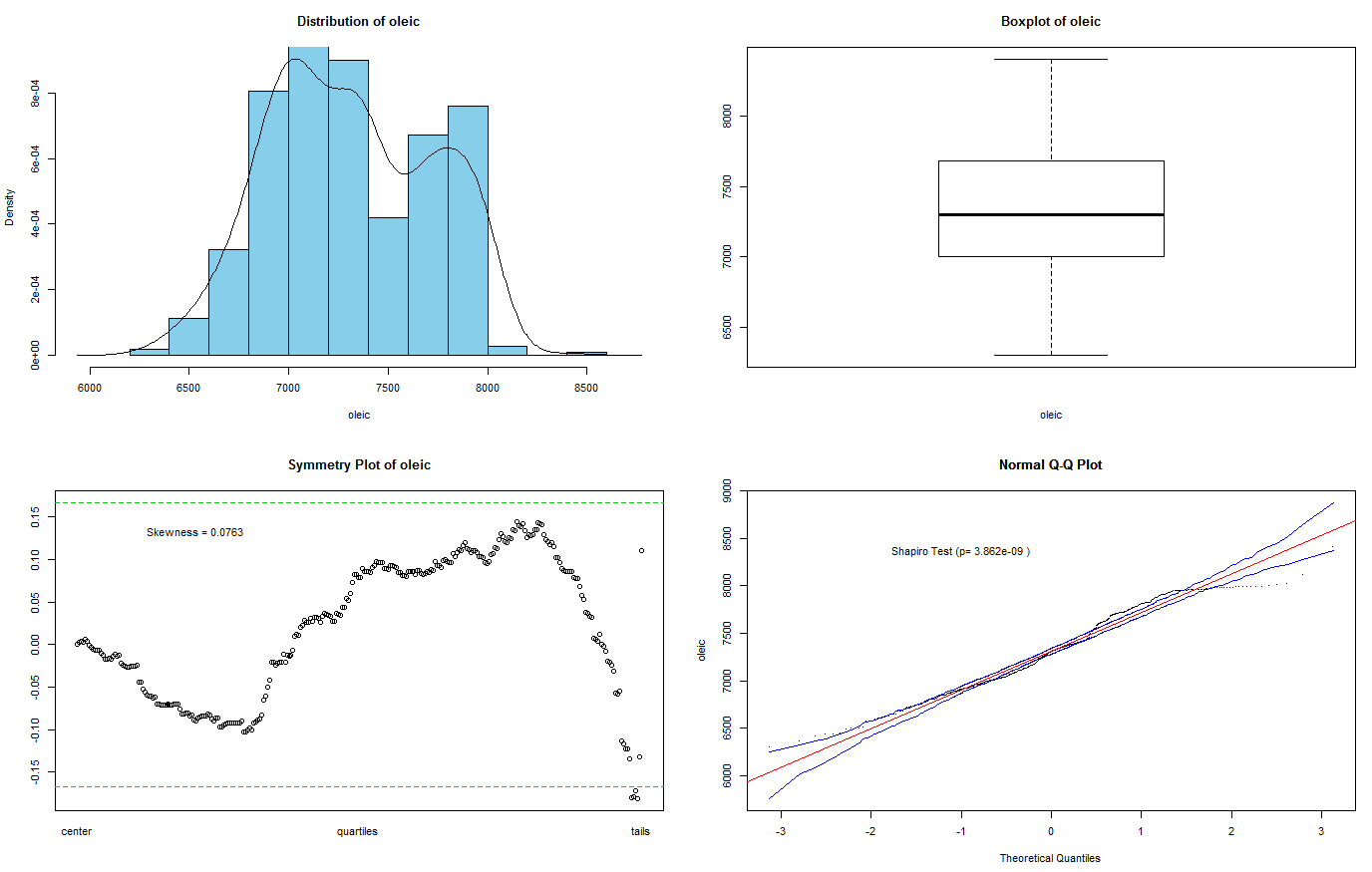
**Data/Variable types:**

* Numeric/continuous - measured quantities, meaningfully numeric!
* Nominal/categorical – non-numeric variable, categories are NOT ordered. Maybe coded numerically, BUT should NEVER be treated as a number.
* Ordinal – really nominal/categorical variable where the categories are meaningfully ordered. Coded numerically, but generally shouldn’t be treated as one.

**1-D Displays:**

Nominal/Ordinal - bar graphs, pie charts  


Continuous – histograms, dotplots, boxplots, symmetry plots, normal quantile plots, CDF plots, density estimates, others….



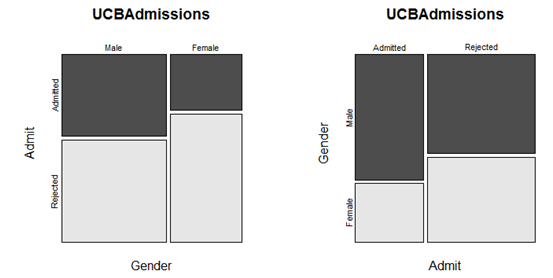
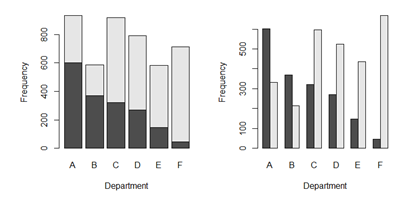
**2-D Displays:**

The type of display depends on the data type of two variables involved.

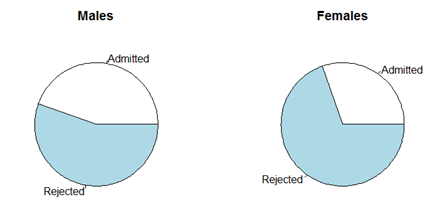
* If both variables are ordinal/nominal 🡪 mosaic plot, stacked bar graph, side-by-side bar graph
* If both variables are continuous 🡪 scatterplot, 2-D density estimate
* If one variable is ordinal/nominal and the other is continuous 🡪 comparative boxplots/dotplots, trellis/conditional displays

Both ordinal/nominal

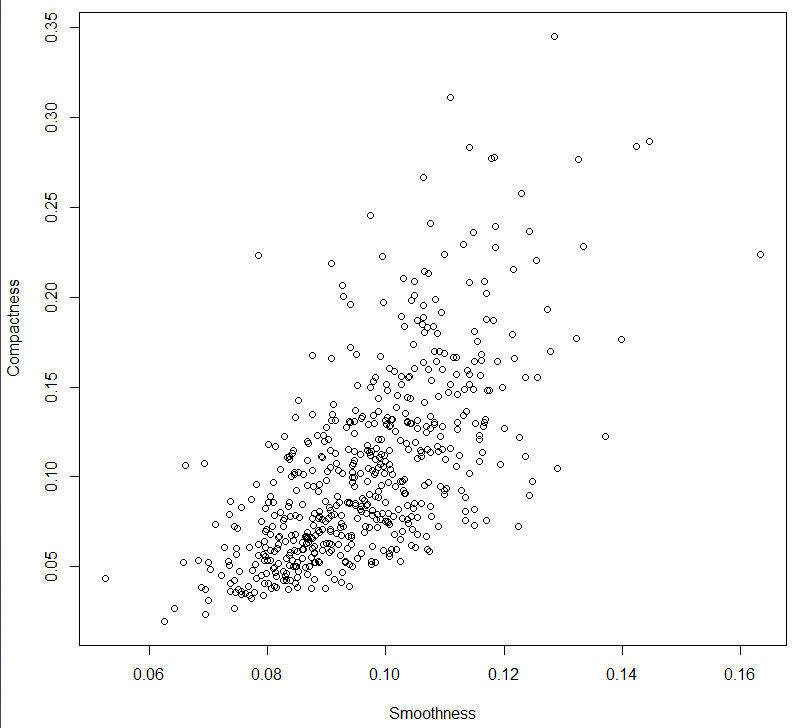
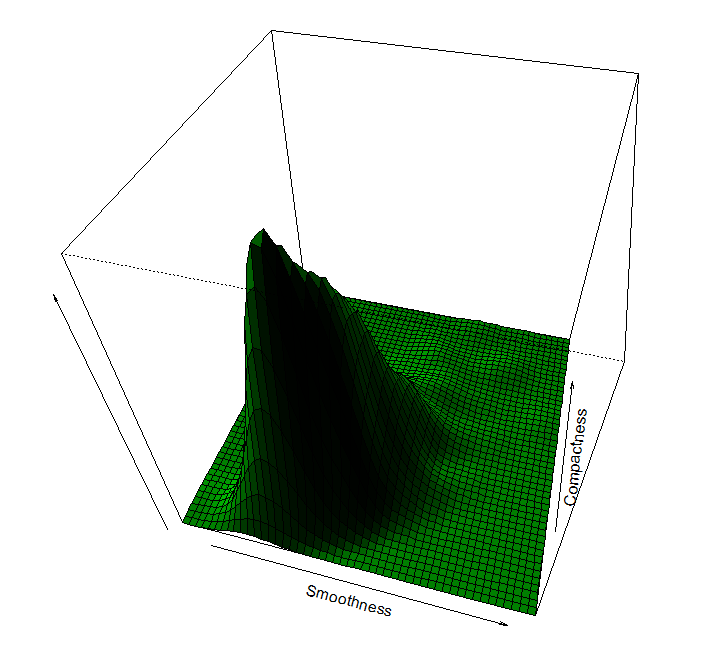
Bar graphs Mosaic plots

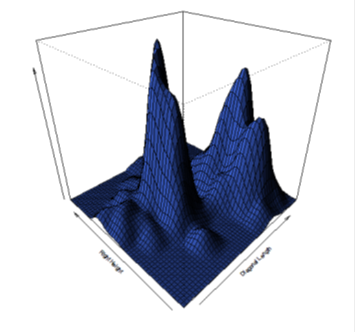
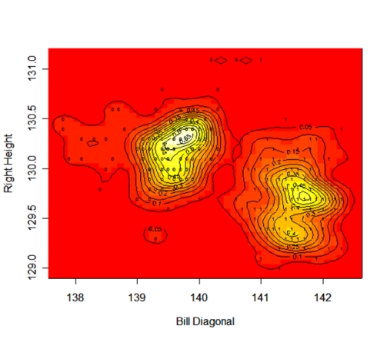


Conditional pie charts



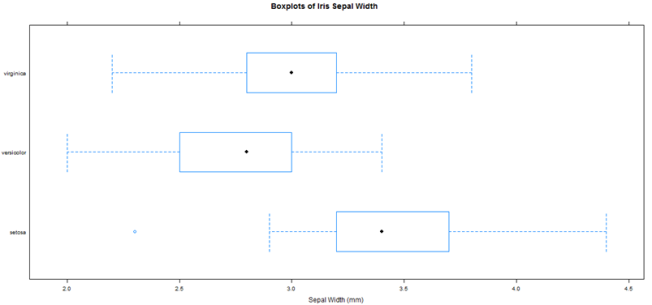
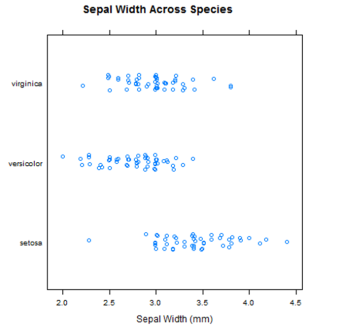
Both continuous  
 Scatterplot 2-D Density Estimates

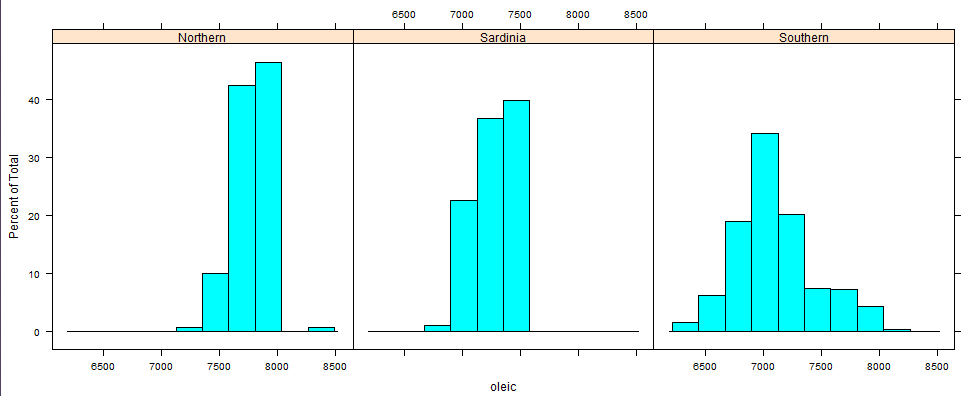
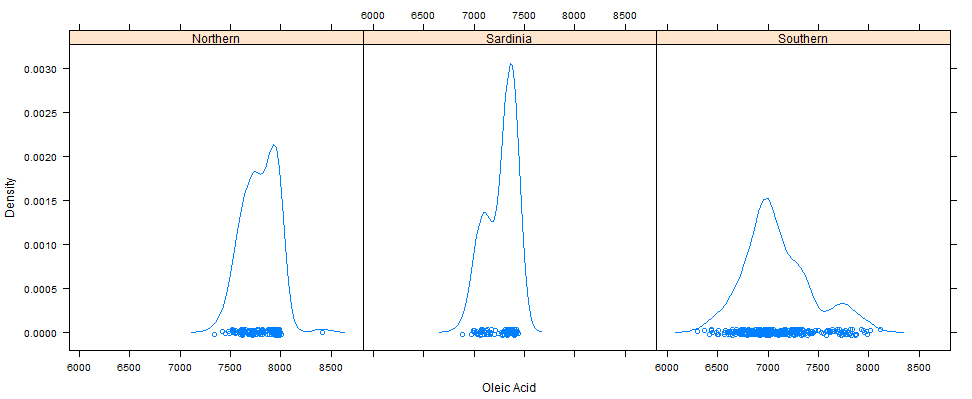
 

One Continuous (response) and One Nominal/Ordinal (group identifier)

Comparative Boxplots Comparative Dotplots/Stripcharts

Conditional Histograms (package lattice) Conditional Density Plots (package lattice)  
 histogram(~response|group) densityplot(~response|group)

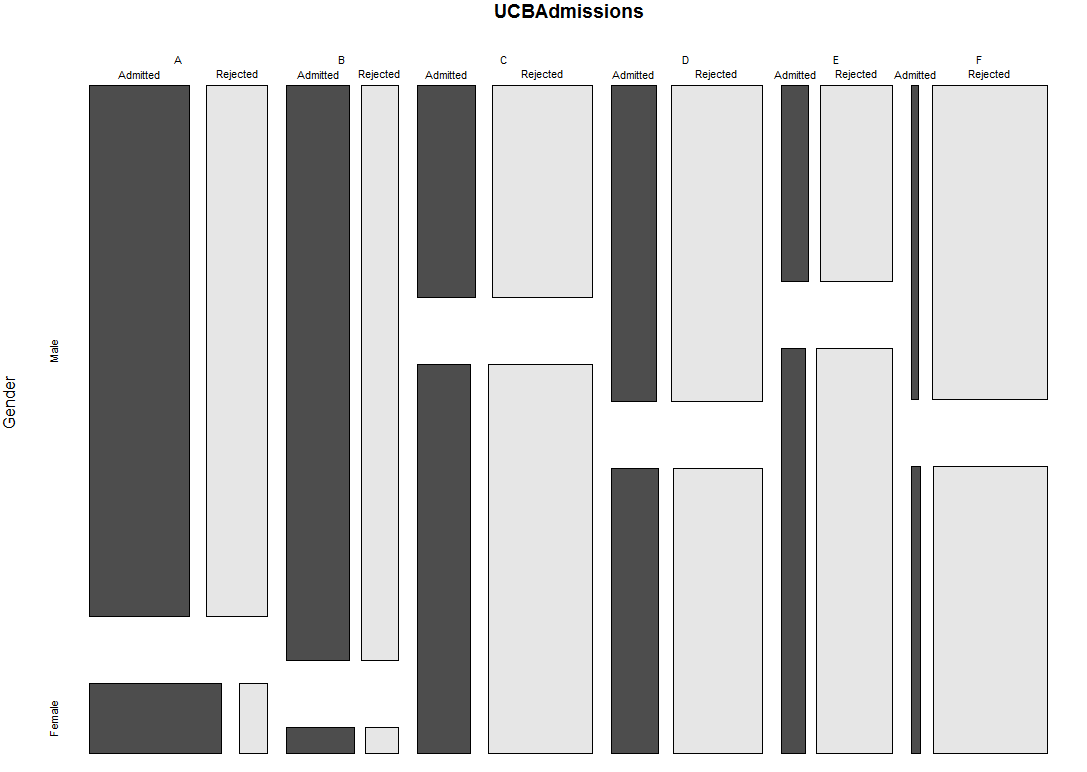
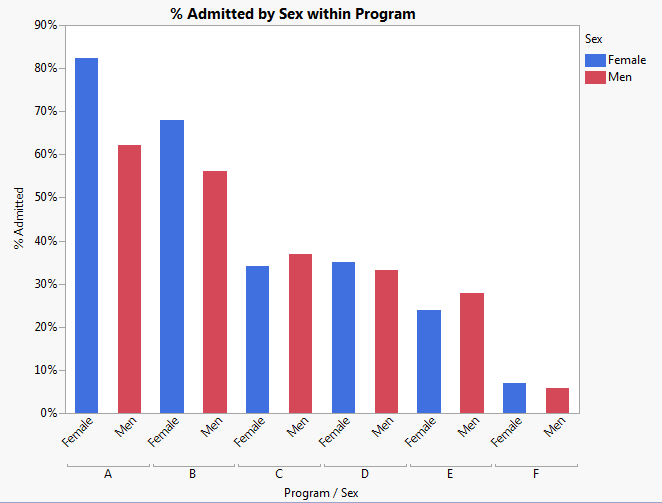
 

**3-D Displays**

The type of display depends on the data type of three variables involved.

* If all variables are ordinal/nominal 🡪 mosaic plots, side-by-side bar graphs
* If all variables are continuous 🡪 3-D scatterplot/spinning plot, scatterplot with bubbles/symbols, conditional scatterplots
* If two variables are continuous and one is ordinal/nominal 🡪 scatterplot with symbols/color, conditional 2-D displays
* If two variables are ordinal/nominal and one is continuous 🡪 conditional 1-D plot of continuous variable

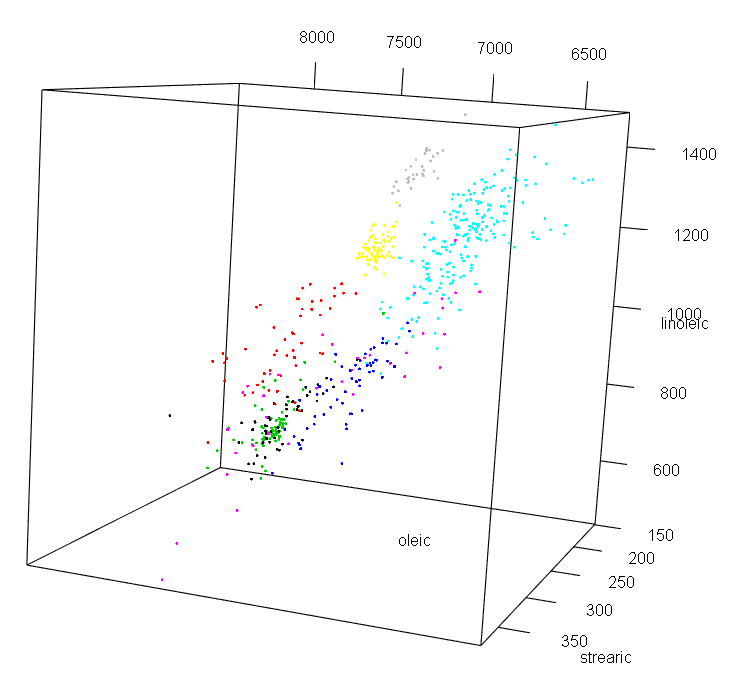
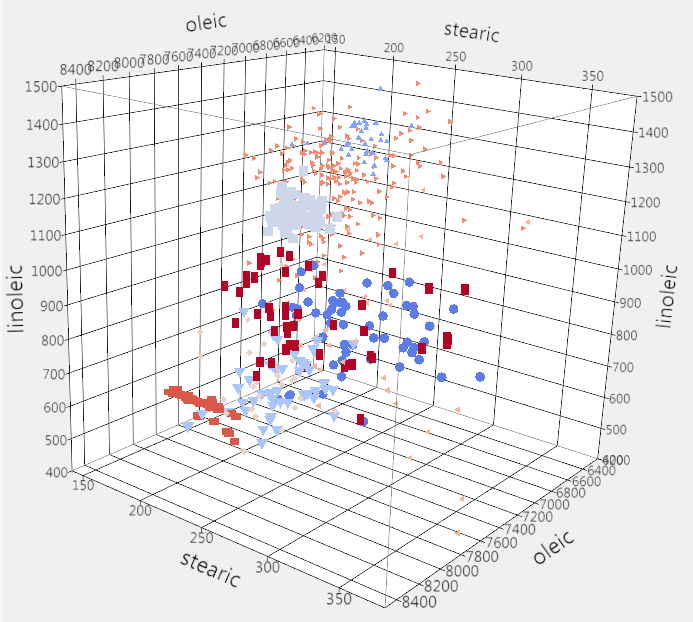
Three nominal/ordinal  
 Mosaic plot Bar Graph

Three continuous

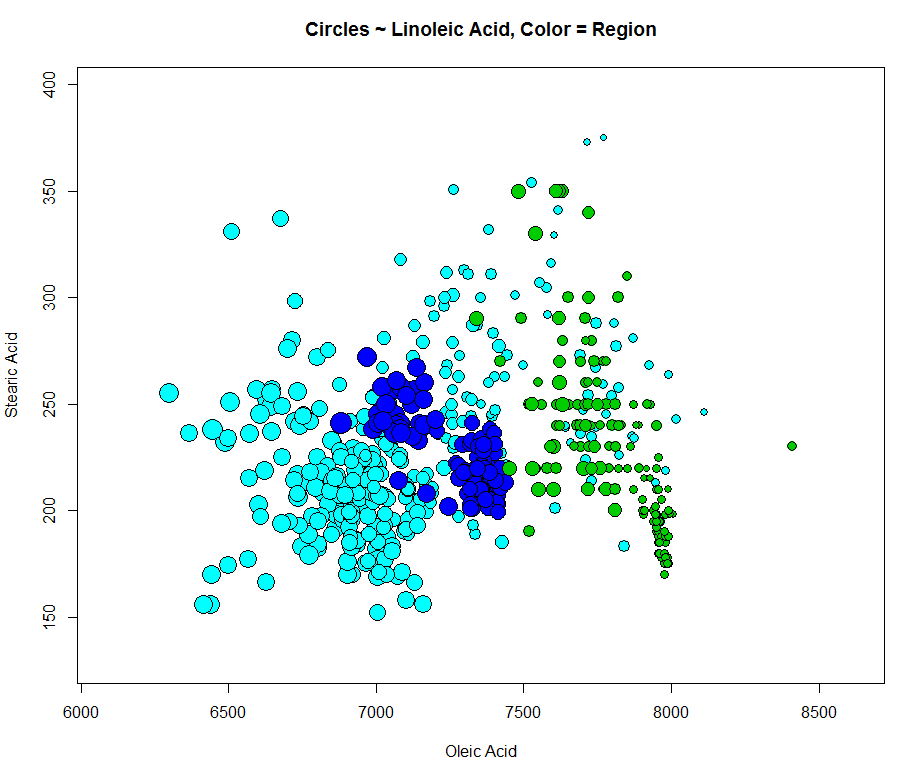
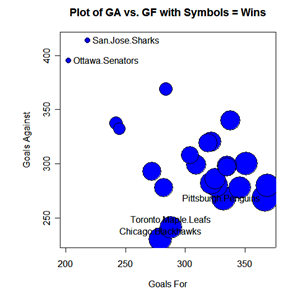
3-D scatterplot/spinning plot (package rgl)

plot3d(x,y,z,col=as.numeric(GROUP)+2) Scatterplot 3D in JMP (Graphs Menu)

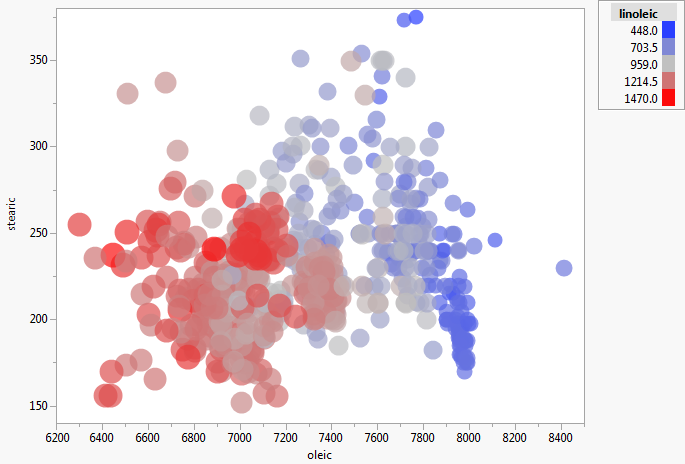
 

**Scatterplot with symbols**

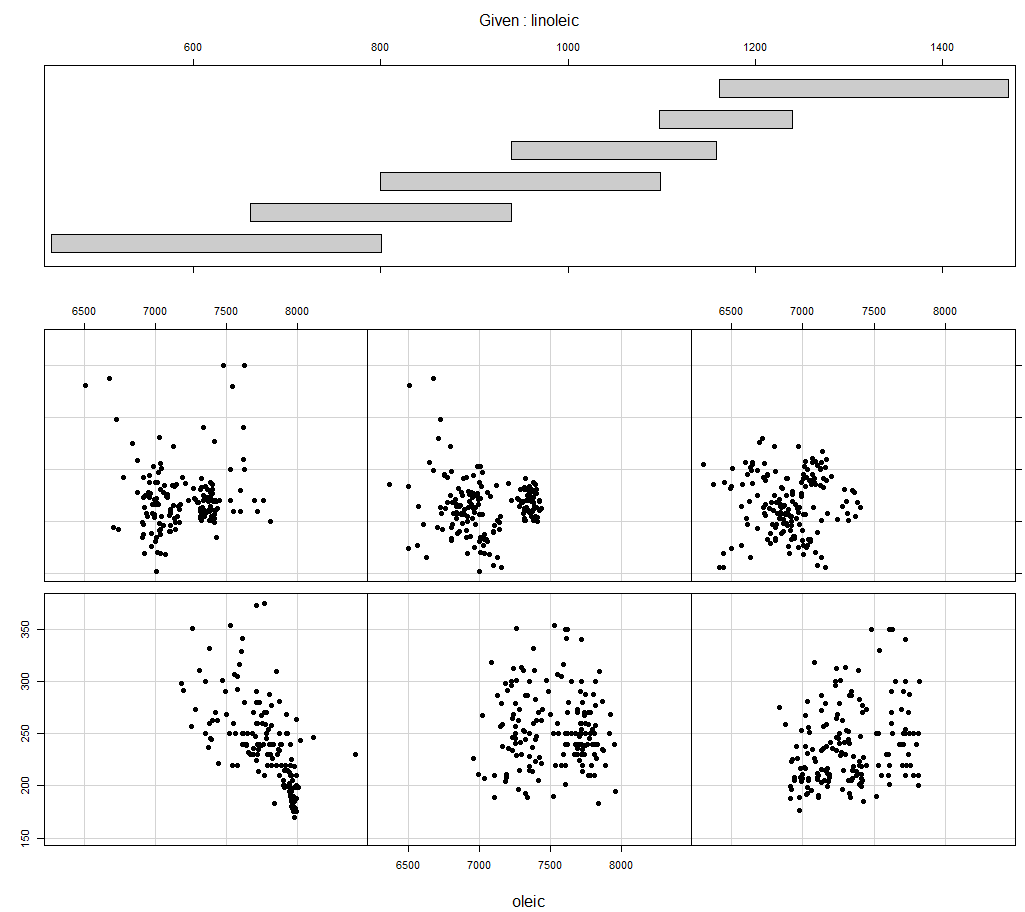
symbols(oleic,stearic,circles=linoleic,inches=.25,bg=as.numeric(Region.name)+2)



Using Bubble Plot in JMP (Graphs menu)

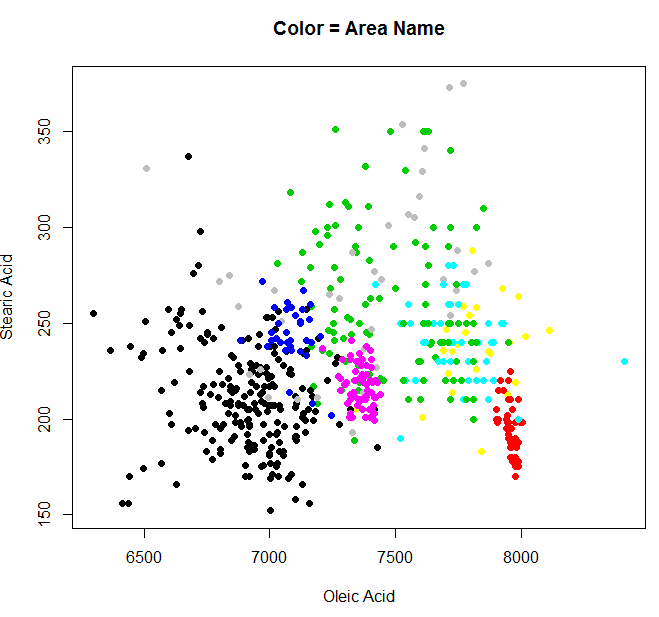
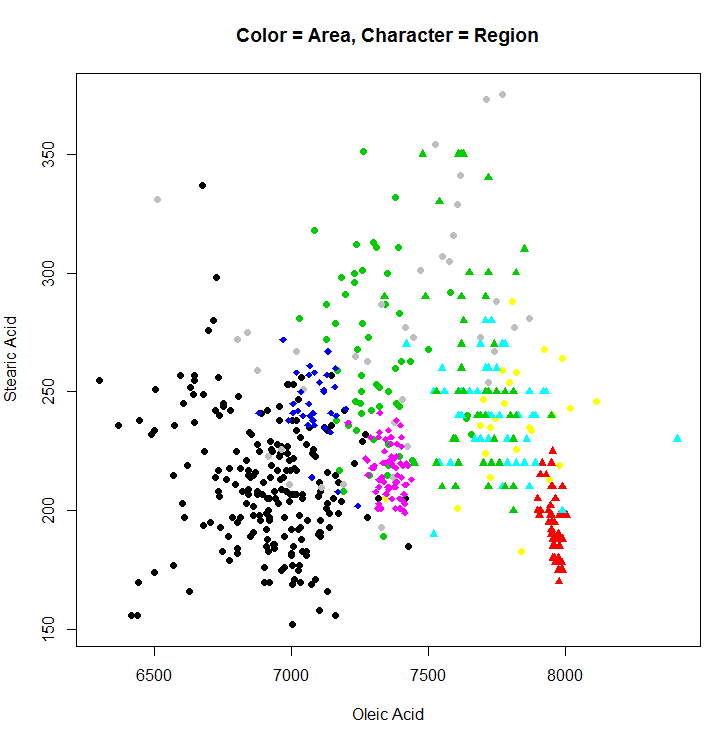


Conditional scatterplots (package lattice)

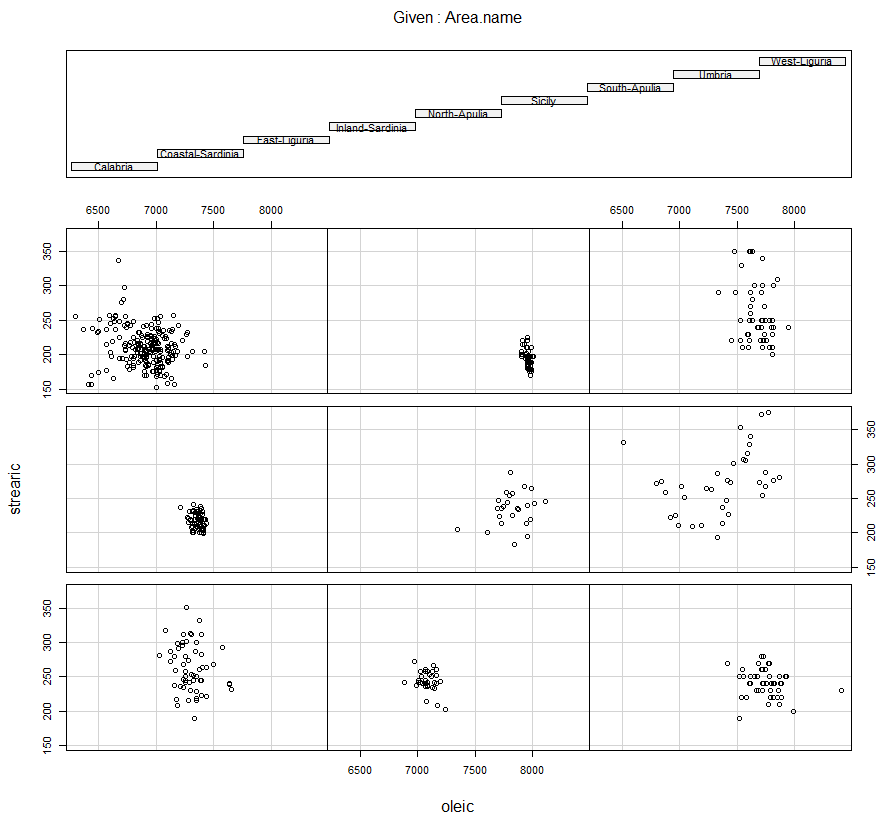
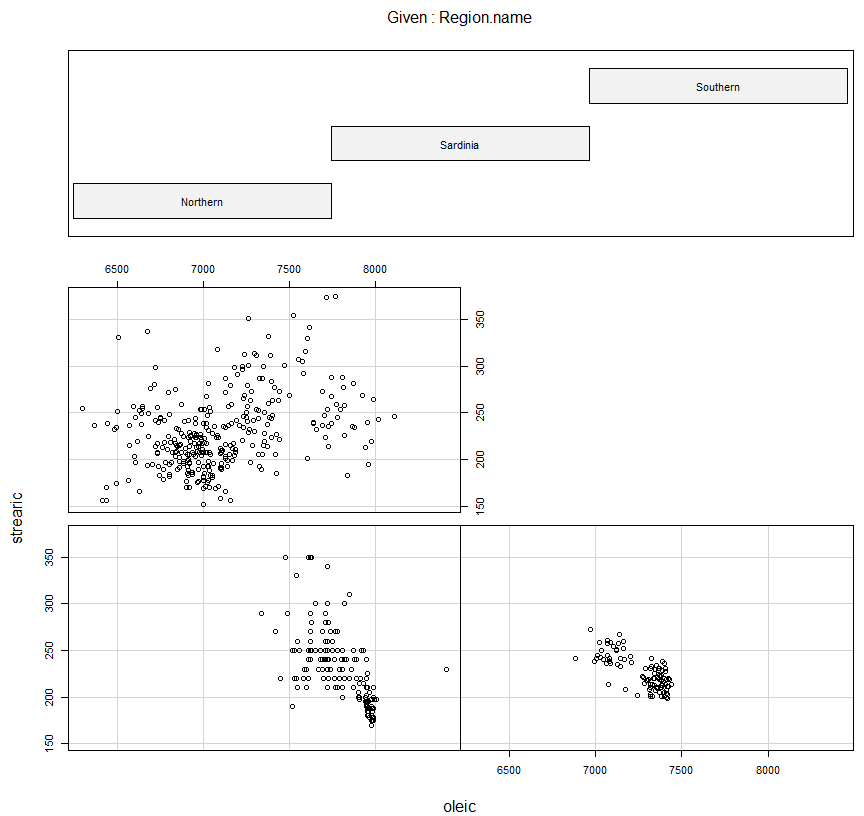
coplot(y~x|z)  


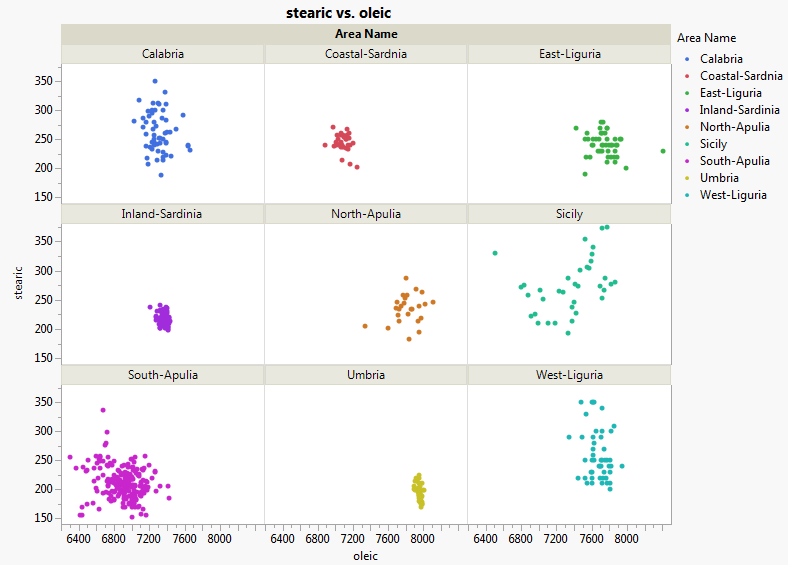
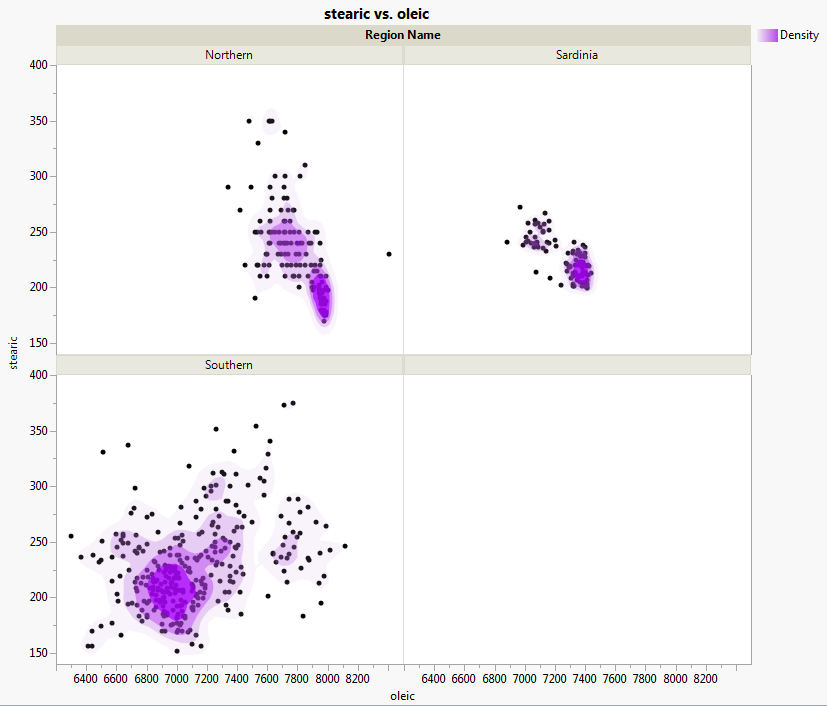
Two continuous & one nominal/ordinal

Scatterplots with color and/or symbols

Conditional scatterplots (package lattice)

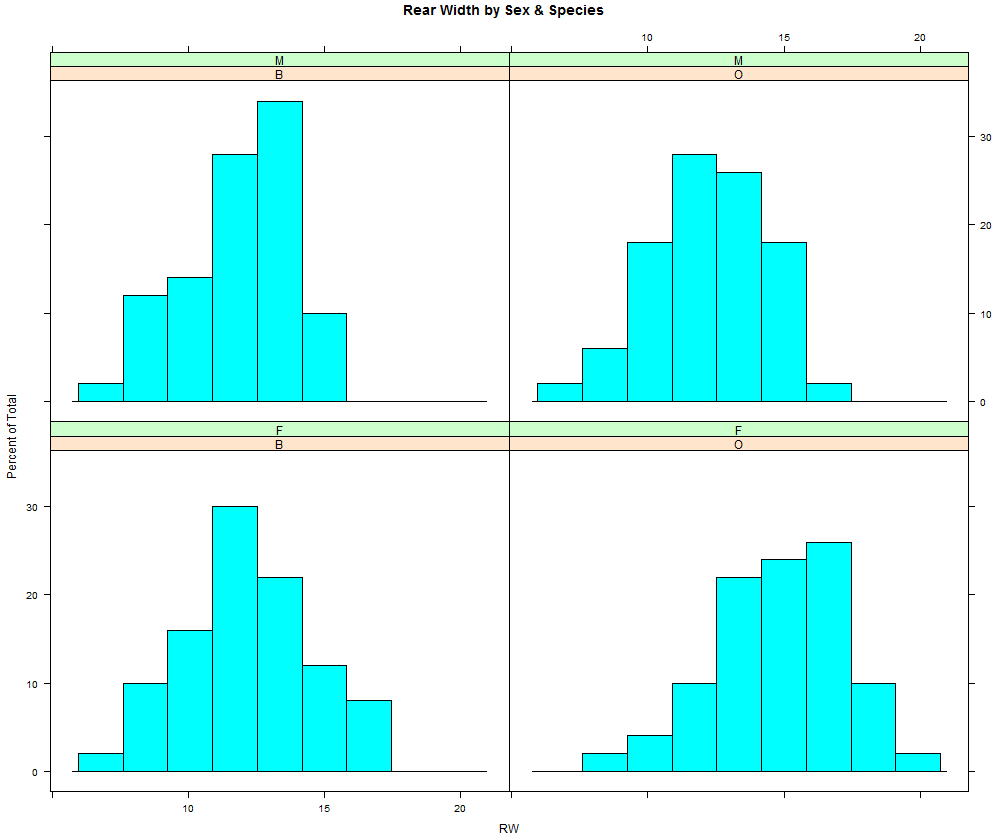
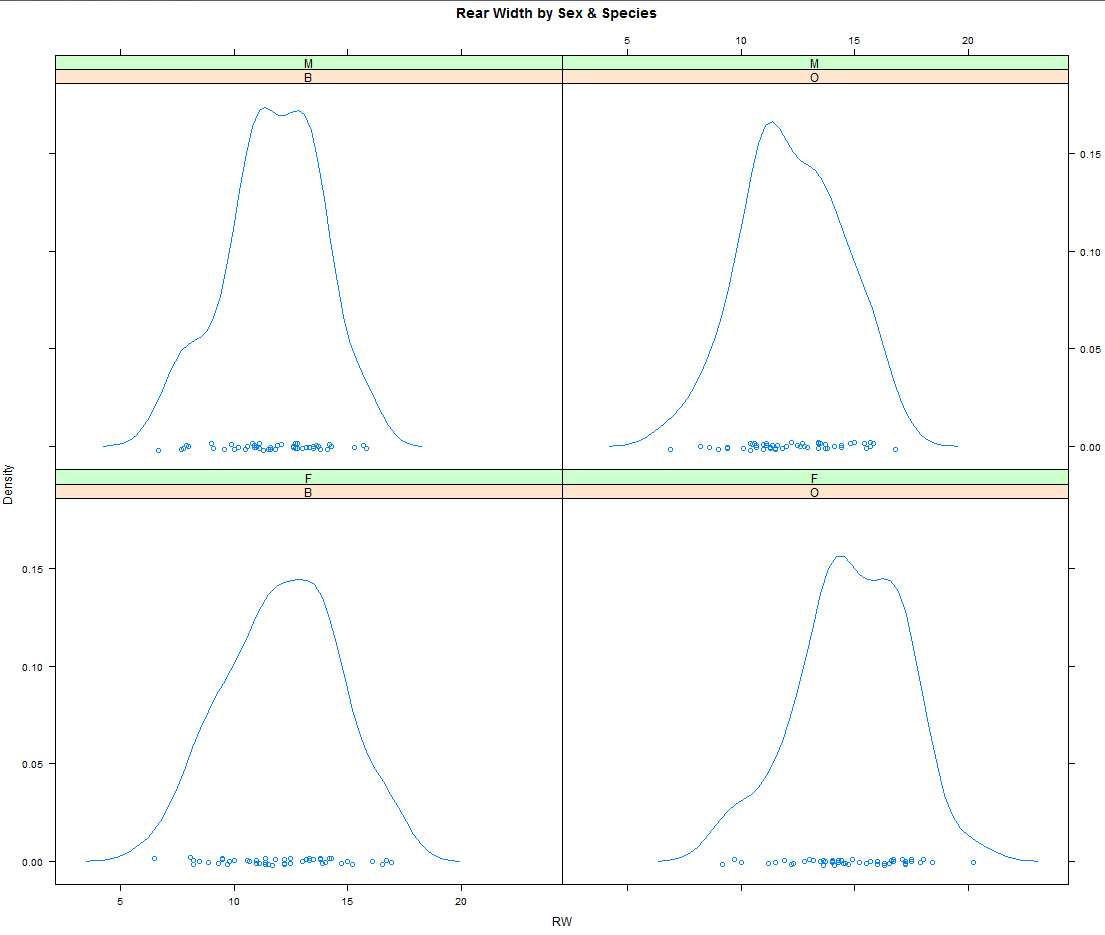
coplot(y~x|group)  
 

Note: These last two displays were created using Graph Builder in JMP (Graphs menu), put the nominal/ordinal variable in the Wrap portion of the graph builder and the continuous variables in X and Y.

One continuous & two nominal/ordinal

Conditional histograms (package lattice) Conditional density estimate (package lattice)  
 histogram(~RW|sex\*sp,data=Crabs) densityplot(~RW|sex\*sp,data=Crabs)

As we have many of the displays in this chapter could be created using functionality in ggplot2 which is package which emphasizes the ***grammar of graphics*** (i.e. gg). Here some additional good resources to get you started on ggplot2 path.

* ***R Graphics Cookbook*** by Winston Chang (O’ Reilly Publishing, 2012).
* ggplot2 website: www.ggplot2.org
* ggplot2 Cheat Sheet (I only used this cheat sheet in creating the ggplot examples in this chapter.)  
  (<https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>)

If I were just beginning to learn R I would probably start by learning ggplot2 versus the plethora of functions I have presented in this chapter.