Basic and Advanced Graphics in R

Histograms and Kernel Density Estimates

`> names(City)` # lists variables that make up data set

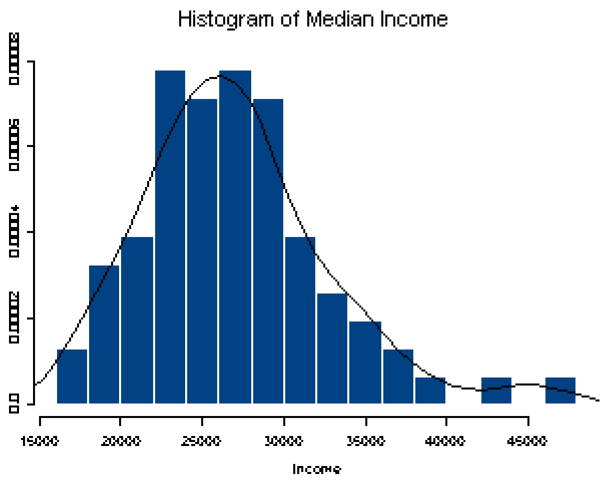
```
[1] "pop1" "pop2" "growth" "area" "popdens" "black.pop" "hisp.pop" "AIP.pop"
[9] "pct.black" "pct.hisp" "pct.AIP" "pctenr" "pctdeg" "pctold" "income" "taxes"
[17] "welfare" "poverty" "pct1par" "unemprt" "laborchg" "femlab" "pctmanu" "ptrans"
[25] "medv" "medrent" "pctrent" "oldhous" "pctcondo" "pct1hous" "infmort" "crime"
[33] "july" "precip"
```

`> attach(City)` # loads data set so variables can be referenced by name

```
> hist(income,prob=T,nclass=20,col = "blue",main="Histogram of Median
Income")
```

# `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 of 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(income))` # 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.



Scatterplots

```
> plot(poverty,infmort,xlab="Percent Below Poverty Level",ylab="Infant
Mortality Rate")
```

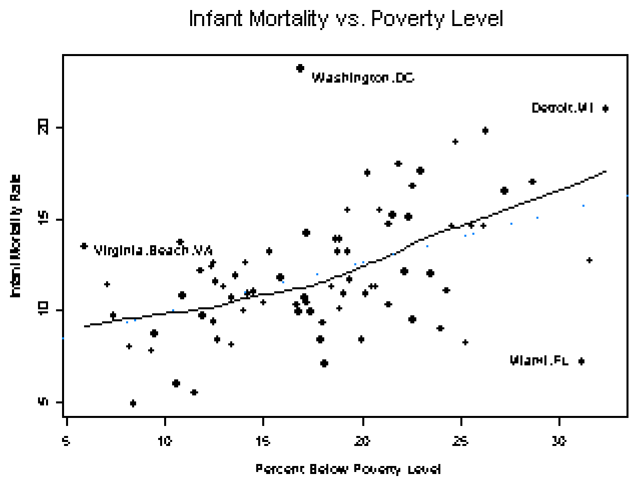
`> lines(lowess(poverty,infmort))` # add loess smooth to plot

`> abline(lm(infmort~poverty),lty=2)` # add a OLS fit and use a dashed line which is obtained by using the optional argument `lty = 2` which means use line type 2

`> identify(poverty,infmort,labels=row.names(City))` # interactively identify some observations

`[1] 19 44 33 9` ← these are the observation numbers that were highlighted.

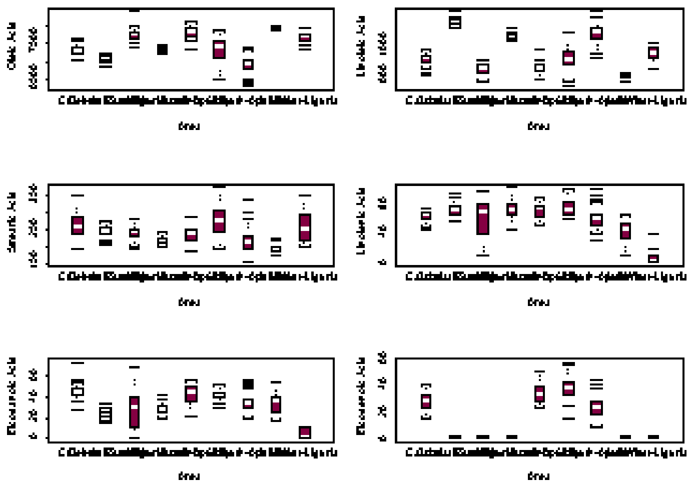
`> title(main="Infant Mortality vs. Poverty Level")` # add a title to the final plot



Comparative Boxplots

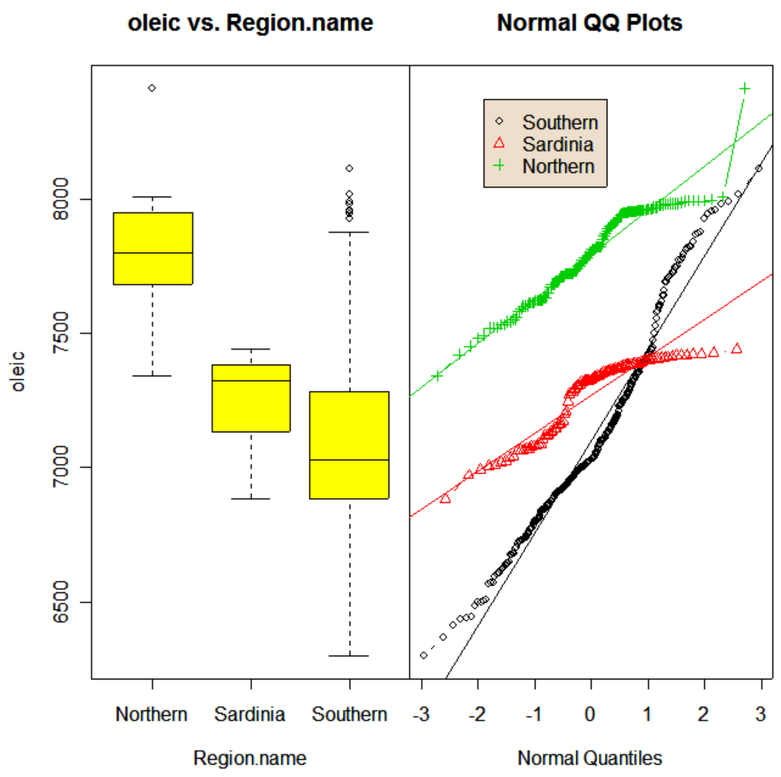
```
> attach(Olives)
> par(mfrow=c(3,2)) # sets a plotting region for with 3 rows and 2
columns
> boxplot(split(oleic,Area.name),xlab="Area",ylab="Oleic Acid")
> boxplot(split(linoleic,Area.name),xlab="Area",ylab="Linoleic Acid")
> boxplot(split(strearic,Area.name),xlab="Area",ylab="Strearic Acid")
> 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.

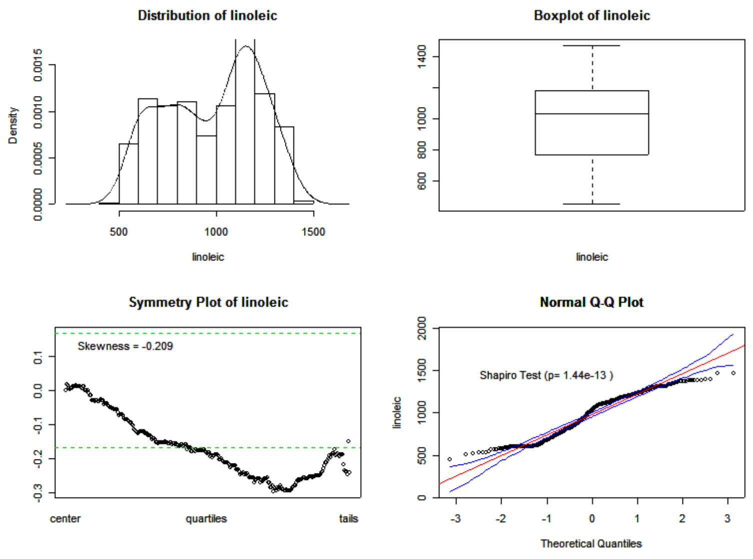
```
> boxqq(oleic~Region.name)
```



statplot

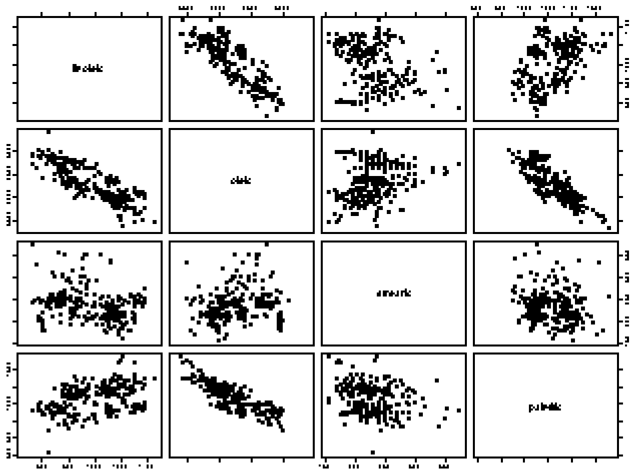
This function returns four univariate displays for continuous data (histogram, boxplot,symmetry plot, and a normal quantile plot).

```
> statplot(linoleic)
```



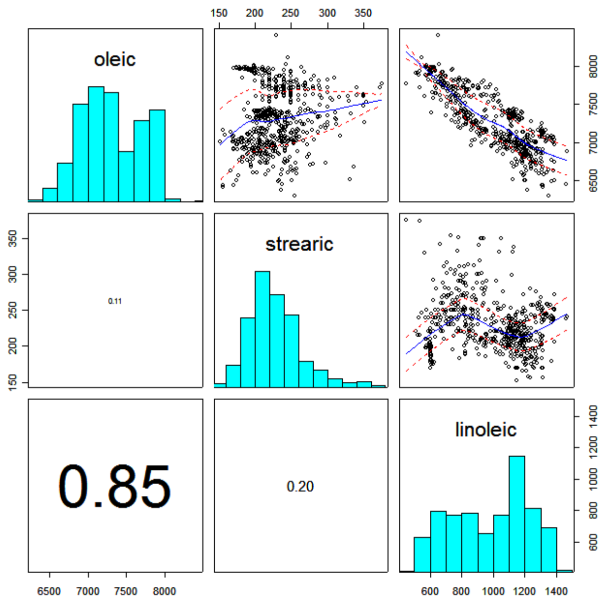
Scatterplot matrices

`> olive.mat <- cbind(linoleic,oleic,strearic,palmitic)` # forms a matrix with these four acid levels forming the columns.
`> 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 magnitude to control font size.

```
> pairs.trendsd <- function(data,...) {
pairs(data,lower.panel=panel.cor,upper.panel=panel.trendsd,diag.panel=p
anel.hist,...)
}
> pairs.trendsd(cbind(oleic,strearic,linoleic))
```



Coplots or Trellised Graphs (`library(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.

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)
> attach(iris)
> names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

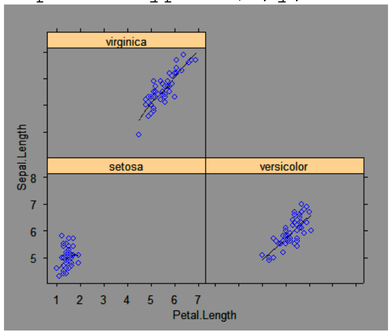
```
> library(lattice)    # load this package from CRAN or copy of the class server.
```

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 * ...' indicating 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 or shingles (Shingles are a way of processing numeric variables for use in conditioning. See documentation of `shingle` 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)
```

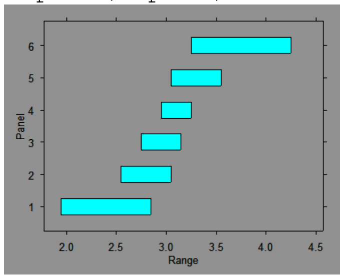
The `panel` option can be used to specify different plot features that can be added to each panel in the trellis display. In the example below we add a loess smooth with span = .75 to each panel and plot data points using blue for the color.

```
> xyplot(Sepal.Length ~ Petal.Length|Species,panel=function(x,y,...){
+ panel.loess(x,y,span=.75)
+ panel.xyplot(x,y,col="blue")})
```



We can condition on a continuous variate by using `equal.count` to create shingles.

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



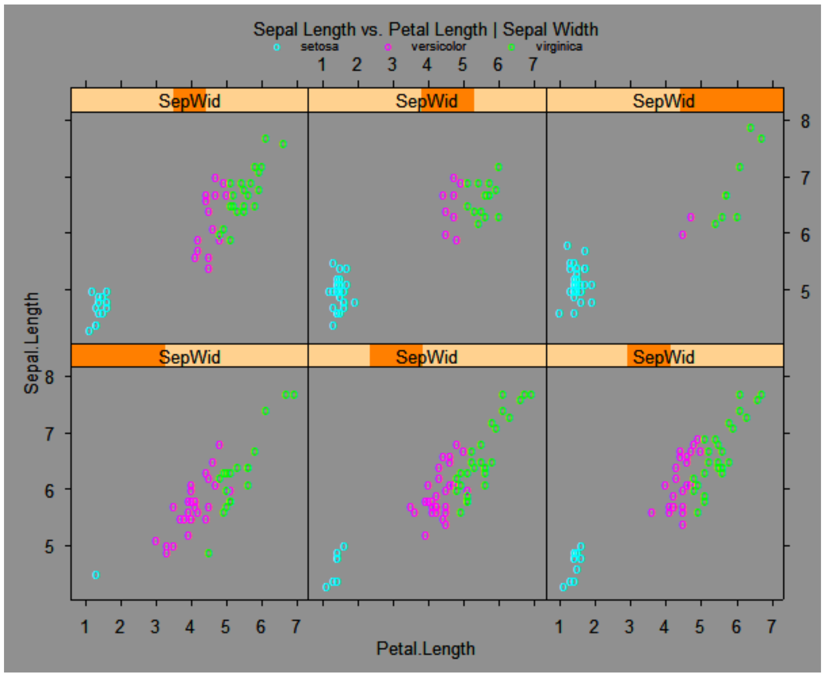
```
> print(SepWid)
Intervals:
   min  max count
1 1.95 2.85    47
2 2.55 3.05    64
3 2.75 3.15    61
4 2.95 3.25    50
5 3.05 3.55    48
6 3.25 4.25    42

Overlap between adjacent intervals:
[1] 28 50 37 24 24

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

A much fancier version uses different plotting symbols for each iris species, adds a title and key.

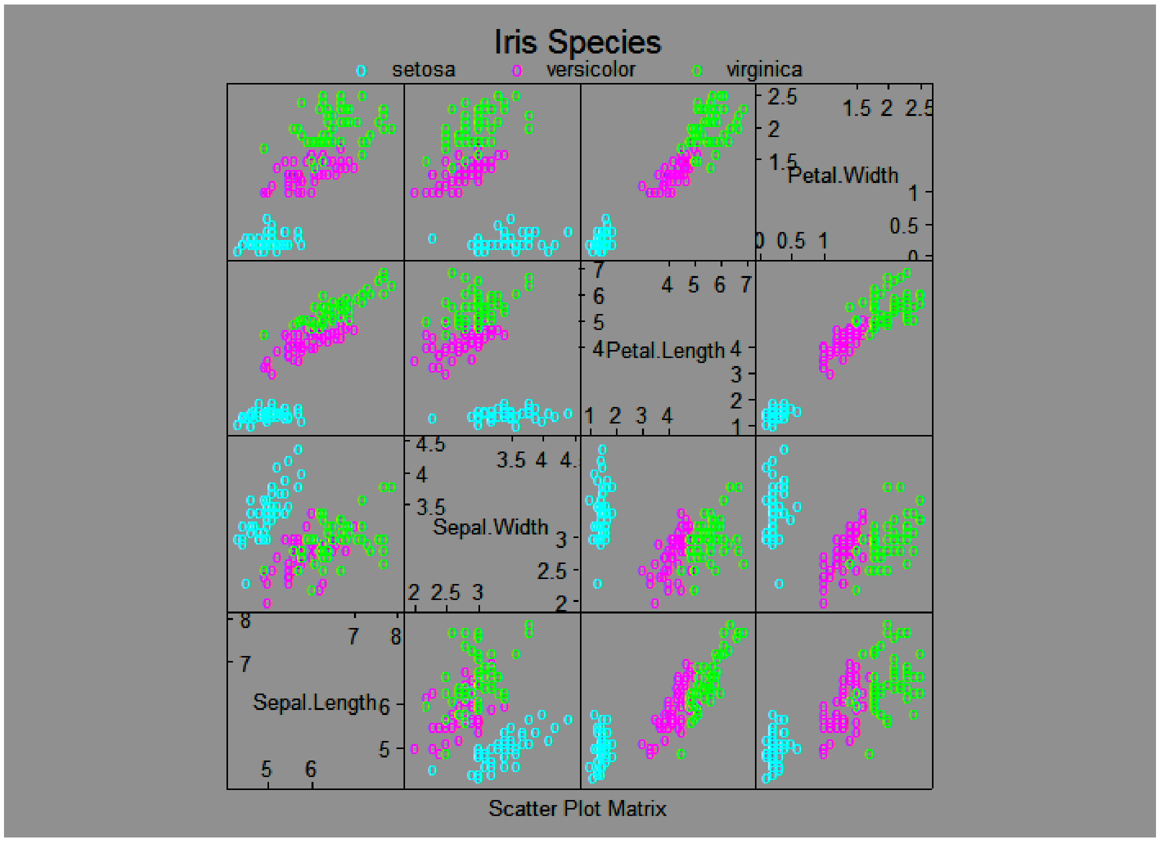
```
> xyplot(Sepal.Length ~ Petal.Length|SepWid,groups=Species,panel=panel.superpose,
+ key=list(title="Sepal Length vs. Petal Length | Sepal Width",
+ columns=3,points=list(pch=super.sym$pch[1:3],col=super.sym$col[1:3]),
+ text=list(levels(Species)),cex=.7))
```



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

```
> splom(~iris[1:4],groups=Species,panel=panel.superpose,
+ key=list(title="Iris Species", columns=3,
+ points=list(pch=super.sym$pch[1:3],col=super.sym$col[1:3]),
+ text=list(levels(Species))))
```

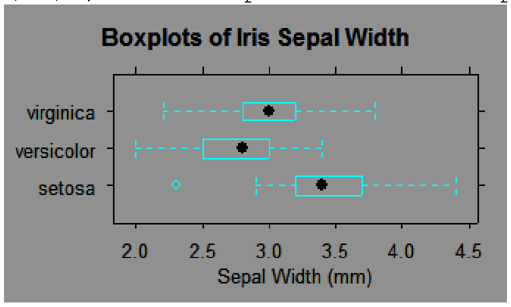
The `key` portion of the command is setting up a legend at the top of the scatterplot matrix. The `panel.superpose` function says to use different colored plotting symbols for each species.



Boxplots

Basic Form: `bwplot(GROUP ~ VARIATE)`

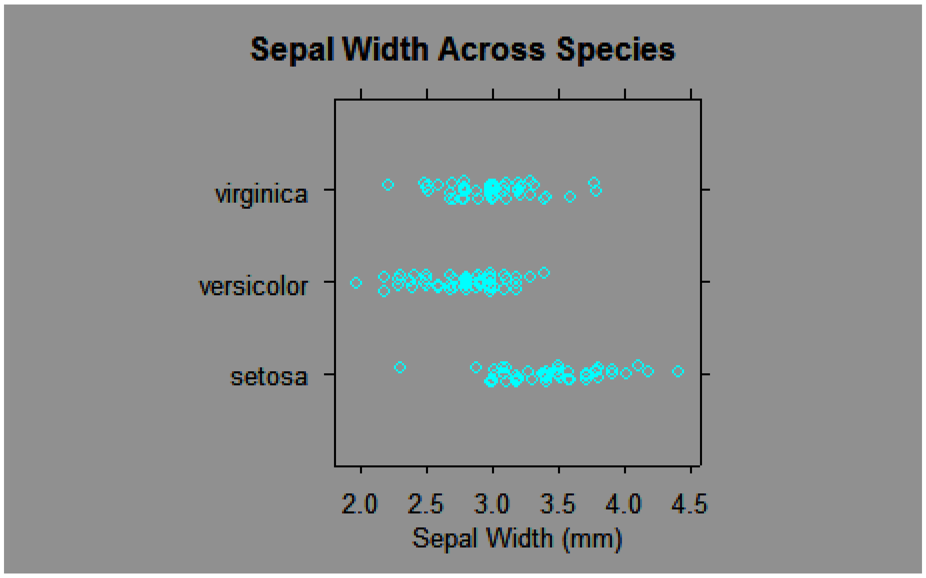
```
> bwplot(Species~Sepal.Width,data=iris,xlab="Sepal Width
(mm)",main="Boxplots of Iris Sepal Width")
```



Strip Plots

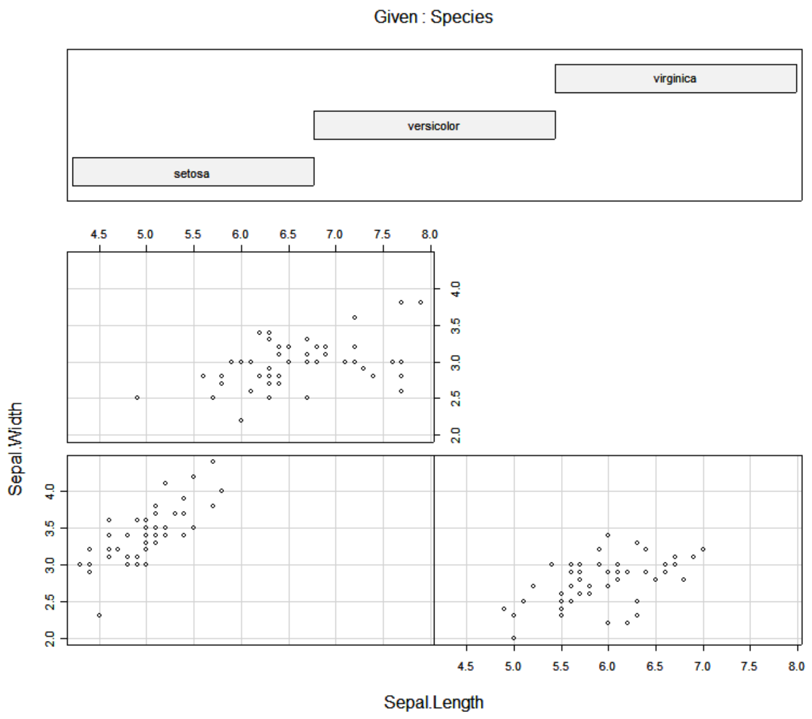
Basic Form: `stripplot(GROUP ~ VARIATE)`

```
stripplot(Species ~ jitter(Sepal.Width), data = iris, aspect = 1,
jitter=T,xlab="Sepal Width (mm)",main="Sepal Width Across Species")
```

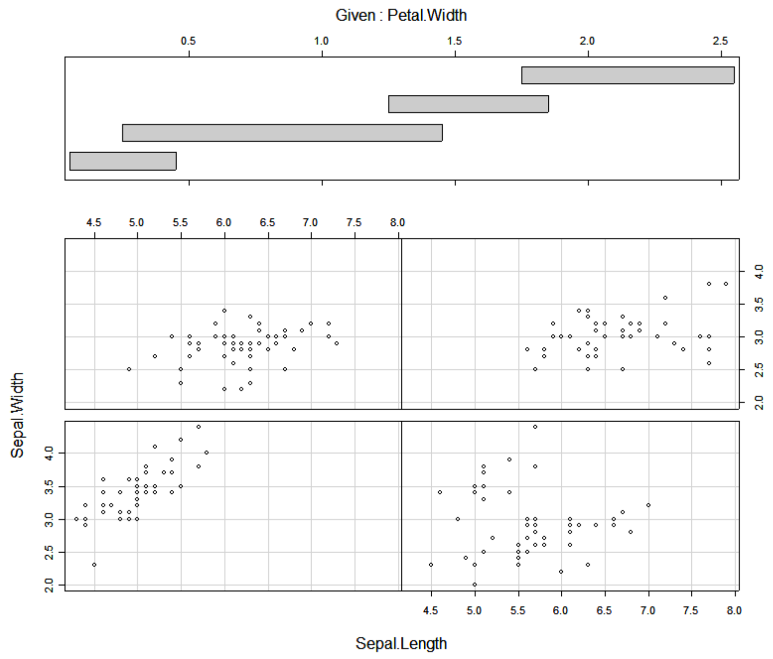


Another function that can be used to create conditional plots is the `coplot` function that is part of the R base package. Here are some examples of its use, again we start with Fisher's iris data.

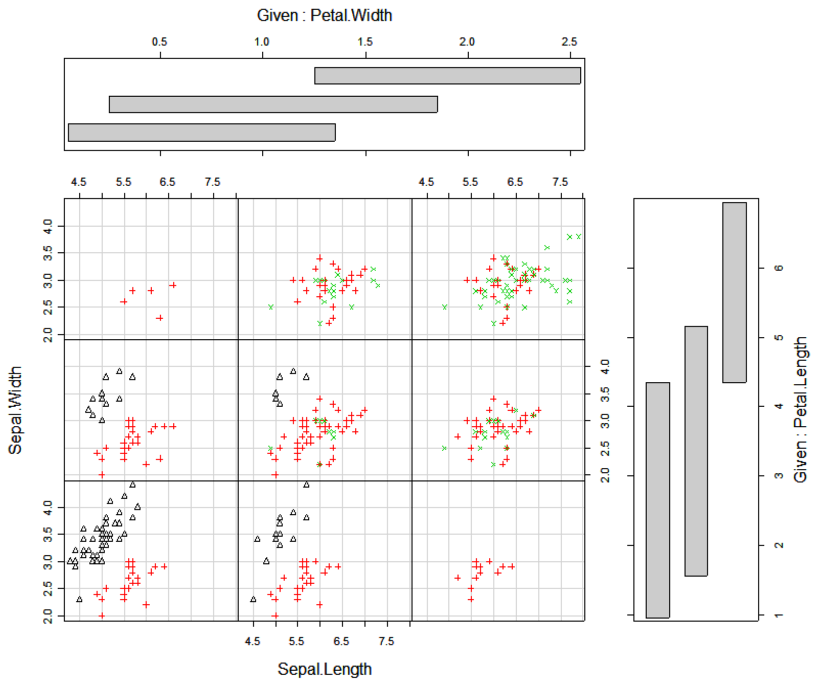
```
> coplot(Sepal.Width~Sepal.Length|Species)
```



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



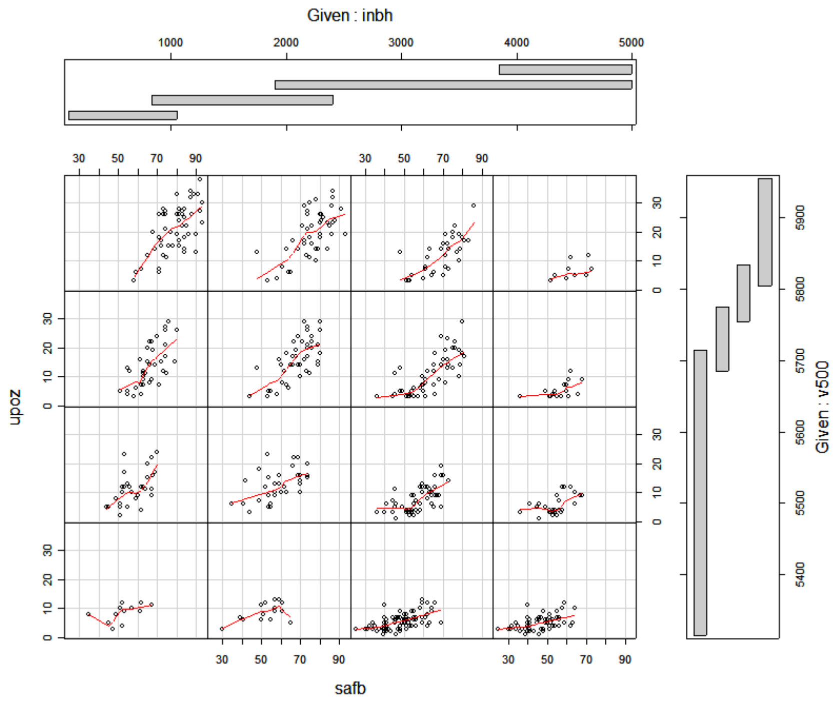
```
>coplot(Sepal.Width~Sepal.Length|Petal.Width*Petal.Length,number=c(3,3)
,overlap=.5,col=as.numeric(Species),pch=as.numeric(Species)+1)
```



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

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,...))
```



End Coplots/Trellised Graphics

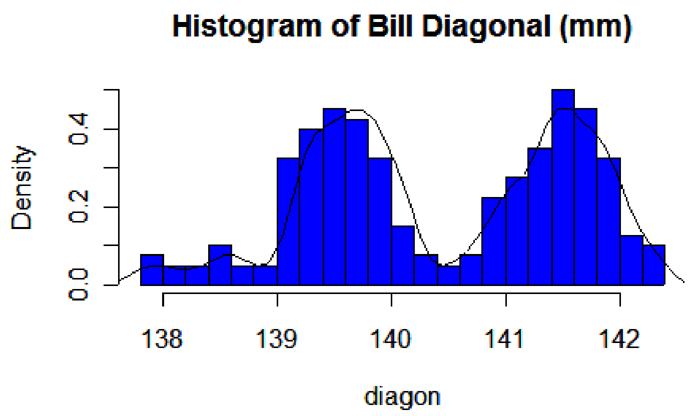
Mores Density Estimates (`library(ash)`)

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. 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.

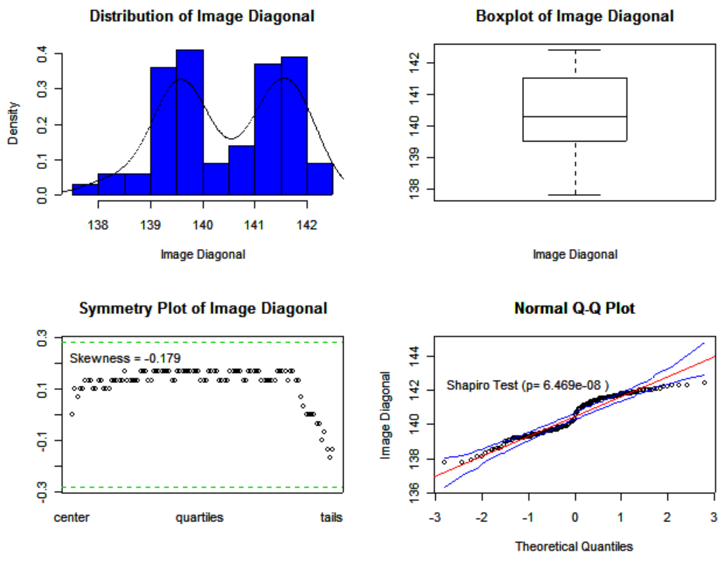
1-D ASH estimates

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

```
> lines(d1)
```

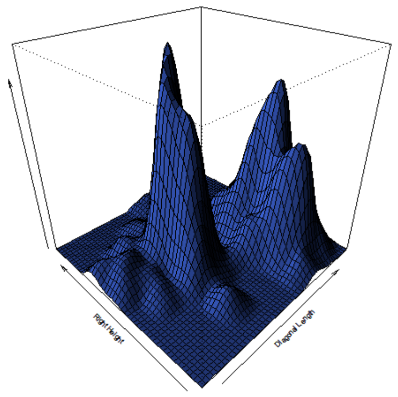


```
> statplot(diagon,xname="Image Diagonal")
```



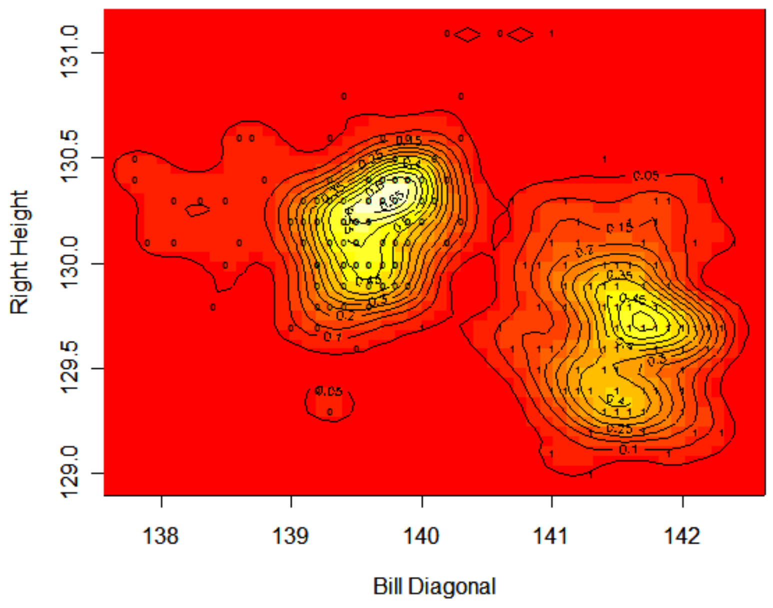
2-D Density Estimate Viewed as Wire Frame Surface

```
> 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")
```



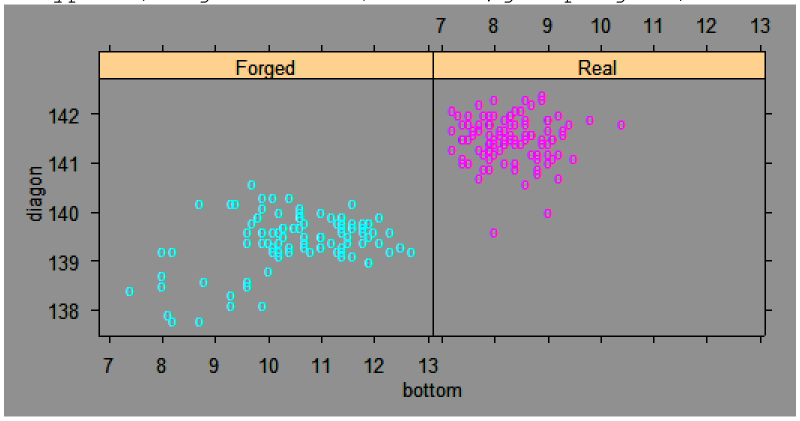
2-D Density Estimate Viewed as Image/Contour Map

```
> 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)
```



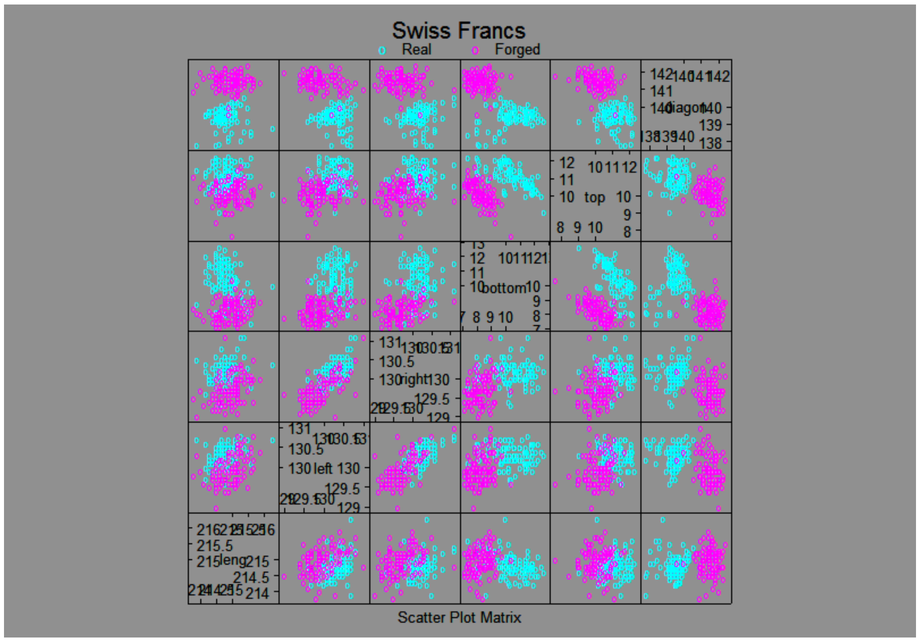
Trellis Display

```
> Genuine <- genu
> xyplot(diagon~bottom|Genuine,groups=genu)
> Genuine[genu==0] <- "Forged"
> Genuine[genu==1] <- "Real"
> xyplot(diagon~bottom|Genuine,groups=genu)
```



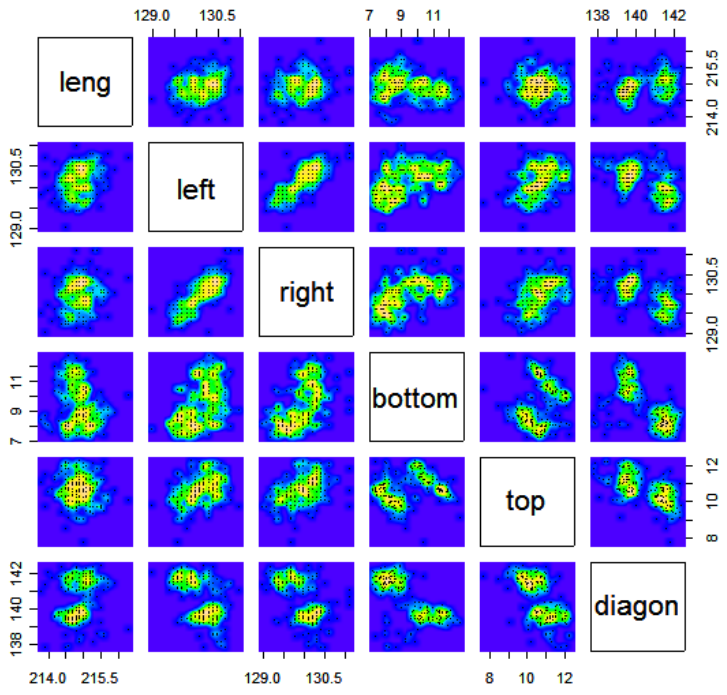
Scatterplot Matrix Using `splom`

```
splom(~Swiss[,2:7],groups=Genuine,cex=.6,panel=panel.superpose,
+  key=list(title="Swiss Francs",columns=2,
+  points=list(pch=super.sym$pch[1:2],col=super.sym$col[1:2]),
+  text=list(unique(Genuine))))
```

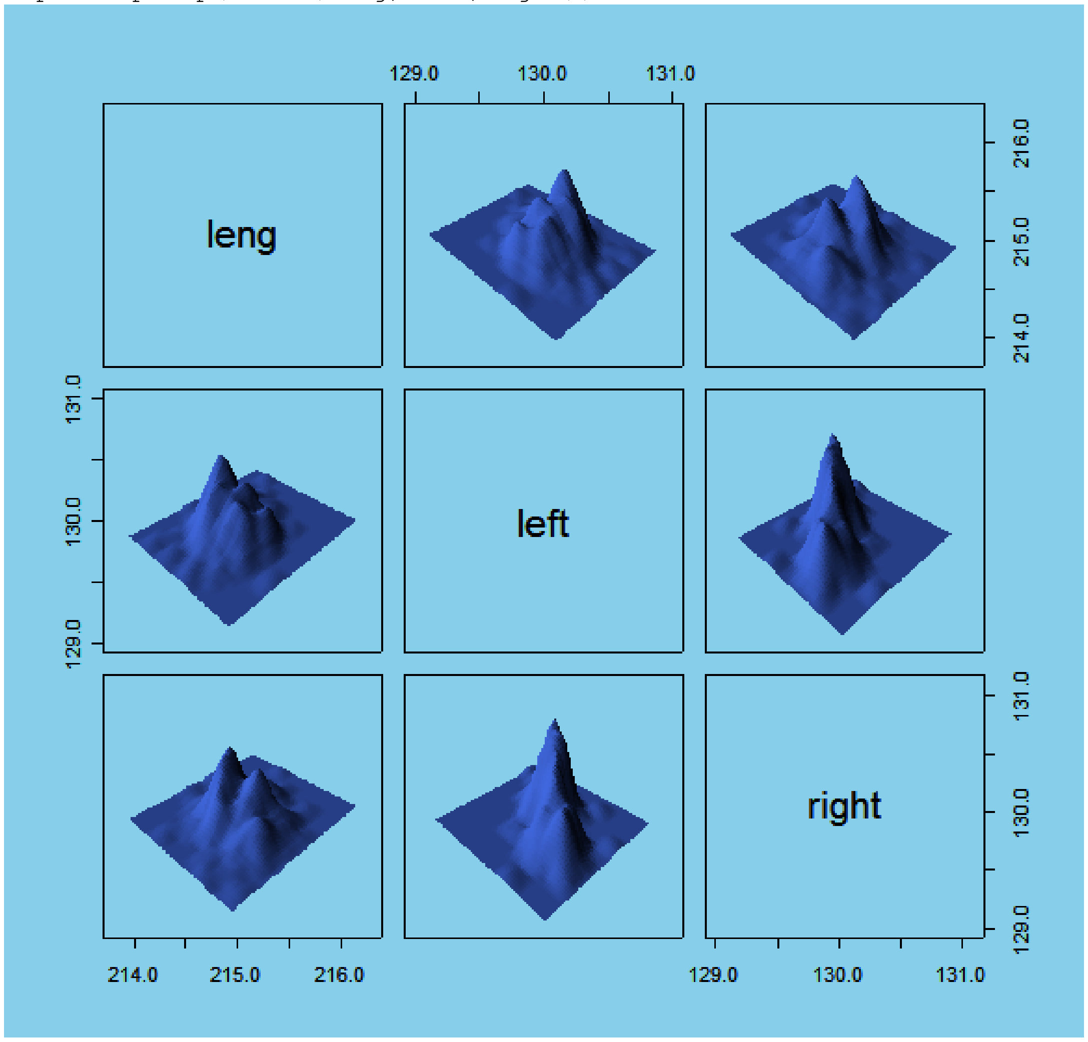


Using Panel Functions with the Standard Scatterplot Matrix Command `pairs`

```
> 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))
```


leng
left
right