**R Basic Data Visualization**

R - Pie Charts

R - Bar Charts

R – Boxplots

R – Histograms

R - Line Graphs

R – Scatterplots

**Pie Charts**

Pie charts are not recommended in the R documentation, and their features are somewhat limited. The authors recommend bar or dot plots over pie charts because people are able to judge length more accurately than volume. Pie charts are created with the function **pie(***x***, labels=)** where *x* is a non-negative numeric vector indicating the area of each slice and labels= notes a character vector of names for the slices.

**Simple Pie Chart**

# Simple Pie Chart  
slices <- c(10, 12,4, 16, 8)  
lbls <- c("US", "UK", "Australia", "Germany", "France")  
pie(slices, labels = lbls, main="Pie Chart of Countries")



## Pie Chart with Annotated Percentages

# Pie Chart with Percentages  
slices <- c(10, 12, 4, 16, 8)   
lbls <- c("US", "UK", "Australia", "Germany", "France")

pct <- round(slices/sum(slices)\*100)  
lbls <- paste(lbls, pct) # add percents to labels   
lbls <- paste(lbls,"%",sep="") # ad % to labels 

pie(slices,labels = lbls, col=rainbow(length(lbls)),  
   main="Pie Chart of Countries")



## 3D Pie Chart

The **pie3D( )** function in the plotrix package provides 3D exploded pie charts.

# 3D Exploded Pie Chart

install.packagte(“plotrix”)  
library(plotrix)

slices <- c(10, 12, 4, 16, 8)   
lbls <- c("US", "UK", "Australia", "Germany", "France")

pie3D(slices,labels=lbls,explode=0.1,  
   main="Pie Chart of Countries ")



## Creating Annotated Pies from a data frame

# Pie Chart from data frame with Appended Sample Sizes  
mytable <- table(iris$Species)  
lbls <- paste(names(mytable), "\n", mytable, sep="")  
pie(mytable, labels = lbls,   
   main="Pie Chart of Species\n (with sample sizes)")

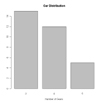


# Bar Plots

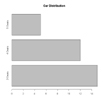
Create barplots with the **barplot(**height**)** function, where height is a vector or matrix. If **height is a vector**, the values determine the heights of the bars in the plot. If **height is a matrix** and the option **beside=FALSE** then each bar of the plot corresponds to a column of height, with the values in the column giving the heights of stacked “sub-bars”. If **height is a matrix** and **beside=TRUE**, then the values in each column are juxtaposed rather than stacked. Include option **names.arg=(**character vector**)**to label the bars. The option **horiz=TRUE** to createa a horizontal barplot.

## Simple Bar Plot

# Simple Bar Plot   
counts <- table(mtcars$gear)  
barplot(counts, main="Car Distribution",   
   xlab="Number of Gears")



# Simple Horizontal Bar Plot with Added Labels   
counts <- table(mtcars$gear)  
barplot(counts, main="Car Distribution", horiz=TRUE,  
  names.arg=c("3 Gears", "4 Gears", "5 Gears"))



## Stacked Bar Plot

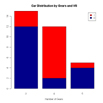
# Stacked Bar Plot with Colors and Legend  
counts <- table(mtcars$vs, mtcars$gear)

X1 <- c()

X2 <- c()

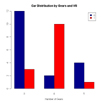
Df = data.frame(x1,x2)

barplot(counts, main="Car Distribution by Gears and VS",  
  xlab="Number of Gears", col=c("darkblue","red"),  
  legend = rownames(counts))



## Grouped Bar Plot

# Grouped Bar Plot  
counts <- table(mtcars$vs, mtcars$gear)  
barplot(counts, main="Car Distribution by Gears and VS",  
  xlab="Number of Gears", col=c("darkblue","red"),  
  legend = rownames(counts), beside=TRUE)



## Notes

Bar plots need not be based on counts or frequencies. You can create bar plots that represent means, medians, standard deviations, etc. Use the aggregate( ) function and pass the results to the barplot( ) function.

By default, the categorical axis line is suppressed. Include the option **axis.lty=1** to draw it.

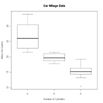
With many bars, bar labels may start to overlap. You can decrease the font size using the **cex.names =** option. Values smaller than one will shrink the size of the label. Additionally, you can use graphical parameters such as the following to help text spacing:

# Fitting Labels   
par(las=2) # make label text perpendicular to axis  
par(mar=c(5,8,4,2)) # increase y-axis margin.  
  
counts <- table(mtcars$gear)  
barplot(counts, main="Car Distribution", horiz=TRUE, names.arg=c("3 Gears", "4 Gears", "5   Gears"), cex.names=0.8)

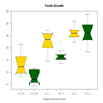
# Boxplots

Boxplots can be created for individual variables or for variables by group. The format is **boxplot(**x**, data=)**, where x is a formula and **data=** denotes the data frame providing the data. An example of a **formula**is y~group where a separate boxplot for numeric variable y is generated for each value of group. Add **varwidth=TRUE** to make boxplot widths proportional to the square root of the samples sizes. Add **horizontal=TRUE** to reverse the axis orientation.

# Boxplot of MPG by Car Cylinders   
boxplot(mpg~cyl,data=mtcars, main="Car Milage Data",   
   xlab="Number of Cylinders", ylab="Miles Per Gallon")



# Notched Boxplot of Tooth Growth Against 2 Crossed Factors  
# boxes colored for ease of interpretation   
boxplot(len~supp\*dose, data=ToothGrowth, notch=TRUE,   
  col=(c("gold","darkgreen")),  
  main="Tooth Growth", xlab="Suppliment and Dose")



In the**notched boxplot**, if two boxes' notches do not overlap this is ‘strong evidence’ their medians differ (Chambers et al., 1983, p. 62).

**Colors recycle**. In the example above, if I had listed 6 colors, each box would have its own color. Earl F. Glynn has created an easy to use list of colors is PDF format.

## Other Options

The **boxplot.matrix( )** function in the sfsmisc package draws a boxplot for each column (row) in a matrix. The **boxplot.n( )** function in the **gplots** package annotates each boxplot with its sample size. The bplot( ) function in the Rlab package offers many more options controlling the positioning and labeling of boxes in the output.

## Violin Plots

A violin plot is a combination of a boxplot and a kernel density plot. They can be created using the vioplot( ) function from vioplot package.

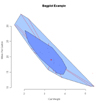
# Violin Plots  
library(vioplot)  
x1 <- mtcars$mpg[mtcars$cyl==4]  
x2 <- mtcars$mpg[mtcars$cyl==6]  
x3 <- mtcars$mpg[mtcars$cyl==8]  
vioplot(x1, x2, x3, names=c("4 cyl", "6 cyl", "8 cyl"),   
   col="gold")  
title("Violin Plots of Miles Per Gallon")



## Bagplot - A 2D Boxplot Extension

The **bagplot(x, y)** function in the aplpack package provides a bivariate version of the univariate boxplot. The bag contains 50% of all points. The bivariate median is approximated. The fence separates points in the fence from points outside. Outliers are displayed.

# Example of a Bagplot  
library(aplpack)  
attach(mtcars)  
bagplot(wt,mpg, xlab="Car Weight", ylab="Miles Per Gallon",  
  main="Bagplot Example")

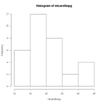


# Histograms and Density Plots

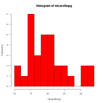
## Histograms

You can create histograms with the function **hist(**x**)** where x is a numeric vector of values to be plotted. The option **freq=FALSE** plots probability densities instead of frequencies. The option **breaks=** controls the number of bins.

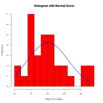
# Simple Histogram  
hist(mtcars$mpg)



# Colored Histogram with Different Number of Bins  
hist(mtcars$mpg, breaks=12, col="red")



# Add a Normal Curve (Thanks to Peter Dalgaard)  
x <- mtcars$mpg   
h<-hist(x, breaks=10, col="red", xlab="Miles Per Gallon",   
   main="Histogram with Normal Curve")   
xfit<-seq(min(x),max(x),length=40)   
yfit<-dnorm(xfit,mean=mean(x),sd=sd(x))   
yfit <- yfit\*diff(h$mids[1:2])\*length(x)   
lines(xfit, yfit, col="blue", lwd=2)



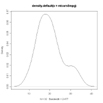
Histograms can be a poor method for determining the shape of a distribution because it is so strongly affected by the number of bins used.

To practice making a density plot with the hist() function, try this exercise.

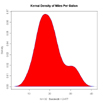
## Kernel Density Plots

Kernal density plots are usually a much more effective way to view the distribution of a variable. Create the plot using **plot(density(**x**))**where x is a numeric vector.

# Kernel Density Plot  
d <- density(mtcars$mpg) # returns the density data   
plot(d) # plots the results



# Filled Density Plot  
d <- density(mtcars$mpg)  
plot(d, main="Kernel Density of Miles Per Gallon")  
polygon(d, col="red", border="blue")

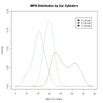


## Comparing Groups VIA Kernal Density

The **sm.density.compare( )**function in the **sm**package allows you to superimpose the kernal density plots of two or more groups. The format is **sm.density.compare(**x**,**factor) where x is a numeric vector and factor is the grouping variable.

# Compare MPG distributions for cars with   
# 4,6, or 8 cylinders  
library(sm)  
attach(mtcars)  
  
# create value labels   
cyl.f <- factor(cyl, levels= c(4,6,8),  
  labels = c("4 cylinder", "6 cylinder", "8 cylinder"))   
  
# plot densities   
sm.density.compare(mpg, cyl, xlab="Miles Per Gallon")  
title(main="MPG Distribution by Car Cylinders")  
  
# add legend via mouse click

colfill<-c(2:(2+length(levels(cyl.f))))   
legend(locator(1), levels(cyl.f), fill=colfill)



**Line Charts**

Overview

Line charts are created with the function **lines(***x***,***y***, type=)** where *x* and *y* are numeric vectors of (x,y) points to connect. **type=** can take the following values:

|  |  |
| --- | --- |
| **type** | **Description** |
| **p** | Points |
| **l** | Lines |
| **o** | overplotted points and lines |
| **b, c** | points (empty if "c") joined by lines |
| **s, S** | stair steps |
| **h** | histogram-like vertical lines |
| **n** | does not produce any points or lines |

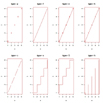
The **lines( )** function *adds* information to a graph. It can not produce a graph on its own. Usually it follows a **plot(***x***,***y***)** command that produces a graph.

By default, **plot( )**plots the (x,y) points. Use the **type="n"** option in the **plot( )** command, to create the graph with axes, titles, etc., but *without* plotting the points.

Example

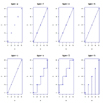
In the following code each of the **type=** options is applied to the same dataset. The **plot( )**command sets up the graph, but ***does not*** plot the points.

x <- c(1:5); y <- x # create some data   
par(pch=22, col="red") # plotting symbol and color   
par(mfrow=c(2,4)) # all plots on one page   
opts = c("p","l","o","b","c","s","S","h")   
for(i in 1:length(opts)){   
  heading = paste("type=",opts[i])   
  plot(x, y, type="n", main=heading)   
  lines(x, y, type=opts[i])   
}

[](https://www.statmethods.net/graphs/images/lines0.png)

Next, we demonstrate each of the **type=** options when **plot( )** sets up the graph and ***does***plot the points.

x <- c(1:5); y <- x # create some data  
par(pch=22, col="blue") # plotting symbol and color   
par(mfrow=c(2,4)) # all plots on one page   
opts = c("p","l","o","b","c","s","S","h")   
for(i in 1:length(opts){   
  heading = paste("type=",opts[i])   
  plot(x, y, main=heading)   
  lines(x, y, type=opts[i])   
}



As you can see, the **type="c"** option only looks different from the **type="b"** option if the plotting of points is suppressed in the **plot( )** command.

To demonstrate the creation of a more complex line chart, let's plot the growth of 5 orange trees over time. Each tree will have its own distinctive line. The data come from the dataset **Orange**.

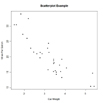
# Create Line Chart  
  
# convert factor to numeric for convenience   
Orange$Tree <- as.numeric(Orange$Tree)   
ntrees <- max(Orange$Tree)  
  
# get the range for the x and y axis   
xrange <- range(Orange$age)   
yrange <- range(Orange$circumference)   
  
# set up the plot   
plot(xrange, yrange, type="n", xlab="Age (days)",  
   ylab="Circumference (mm)" )   
colors <- rainbow(ntrees)   
linetype <- c(1:ntrees)   
plotchar <- seq(18,18+ntrees,1)  
  
# add lines   
for (i in 1:ntrees) {   
  tree <- subset(Orange, Tree==i)   
  lines(tree$age, tree$circumference, type="b", lwd=1.5,  
    lty=linetype[i], col=colors[i], pch=plotchar[i])   
}   
  
# add a title and subtitle   
title("Tree Growth", "example of line plot")  
  
# add a legend   
legend(xrange[1], yrange[2], 1:ntrees, cex=0.8, col=colors,  
   pch=plotchar, lty=linetype, title="Tree")

# Scatterplots

## Simple Scatterplot

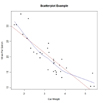
There are **many** ways to create a scatterplot in R. The basic function is **plot(**x**,**y**)**, where x and y are numeric vectors denoting the (x,y) points to plot.

# Simple Scatterplot  
attach(mtcars)  
plot(wt, mpg, main="Scatterplot Example",   
   xlab="Car Weight ", ylab="Miles Per Gallon ", pch=19)



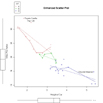
(To practice making a simple scatterplot, try this interactive example from DataCamp.)

# Add fit lines  
abline(lm(mpg~wt), col="red") # regression line (y~x)   
lines(lowess(wt,mpg), col="blue") # lowess line (x,y)



The **scatterplot( )** function in the car package offers many enhanced features, including fit lines, marginal box plots, conditioning on a factor, and interactive point identification. Each of these features is optional.

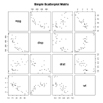
# Enhanced Scatterplot of MPG vs. Weight   
# by Number of Car Cylinders   
library(car)   
scatterplot(mpg ~ wt | cyl, data=mtcars,   
   xlab="Weight of Car", ylab="Miles Per Gallon",   
   main="Enhanced Scatter Plot",   
   labels=row.names(mtcars))



## Scatterplot Matrices

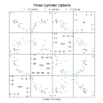
There are at least 4 useful functions for creating scatterplot matrices. Analysts must love scatterplot matrices!

# Basic Scatterplot Matrix  
pairs(~mpg+disp+drat+wt,data=mtcars,   
   main="Simple Scatterplot Matrix")



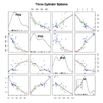
The lattice package provides options to condition the scatterplot matrix on a factor.

# Scatterplot Matrices from the lattice Package   
library(lattice)  
splom(mtcars[c(1,3,5,6)], groups=cyl, data=mtcars,  
   panel=panel.superpose,   
   key=list(title="Three Cylinder Options",  
   columns=3,  
   points=list(pch=super.sym$pch[1:3],  
   col=super.sym$col[1:3]),  
   text=list(c("4 Cylinder","6 Cylinder","8 Cylinder"))))



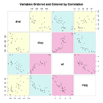
The car package can condition the scatterplot matrix on a factor, and optionally include lowess and linear best fit lines, and boxplot, densities, or histograms in the principal diagonal, as well as rug plots in the margins of the cells.

# Scatterplot Matrices from the car Package  
library(car)  
scatterplot.matrix(~mpg+disp+drat+wt|cyl, data=mtcars,  
   main="Three Cylinder Options")



The gclus package provides options to rearrange the variables so that those with higher correlations are closer to the principal diagonal. It can also color code the cells to reflect the size of the correlations.

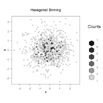
# Scatterplot Matrices from the glus Package   
library(gclus)  
dta <- mtcars[c(1,3,5,6)] # get data   
dta.r <- abs(cor(dta)) # get correlations  
dta.col <- dmat.color(dta.r) # get colors  
# reorder variables so those with highest correlation  
# are closest to the diagonal  
dta.o <- order.single(dta.r)   
cpairs(dta, dta.o, panel.colors=dta.col, gap=.5,  
main="Variables Ordered and Colored by Correlation" )



## High Density Scatterplots

When there are many data points and significant overlap, scatterplots become less useful. There are several approaches that be used when this occurs. The **hexbin(x, y)**function in the hexbin package provides bivariate binning into hexagonal cells (it looks better than it sounds).

# High Density Scatterplot with Binning  
library(hexbin)  
x <- rnorm(1000)  
y <- rnorm(1000)  
bin<-hexbin(x, y, xbins=50)   
plot(bin, main="Hexagonal Binning")



Another option for a scatterplot with significant point overlap is the **sunflowerplot**. See **help(sunflowerplot)**for details.

Finally, you can save the scatterplot in **PDF** format and use color transparency to allow points that overlap to show through (this idea comes from B.S. Everrit in HSAUR).

# High Density Scatterplot with Color Transparency   
pdf("c:/scatterplot.pdf")   
x <- rnorm(1000)  
y <- rnorm(1000)   
plot(x,y, main="PDF Scatterplot Example", col=rgb(0,100,0,50,maxColorValue=255), pch=16)  
dev.off()



Note: You can use the **col2rgb( )**function to get the rbg values for R colors. For example, **col2rgb("**darkgreen**")** yeilds r=0, g=100, b=0. Then add the alpha transparency level as the 4th number in the color vector. A value of zero means fully transparent. See **help(rgb)**for more information.

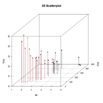
## 3D Scatterplots

You can create a 3D scatterplot with the scatterplot3d package. Use the function **scatterplot3d(**x**,** y**,** z).

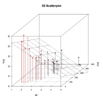
# 3D Scatterplot  
library(scatterplot3d)  
attach(mtcars)  
scatterplot3d(wt,disp,mpg, main="3D Scatterplot")



# 3D Scatterplot with Coloring and Vertical Drop Lines  
library(scatterplot3d)   
attach(mtcars)   
scatterplot3d(wt,disp,mpg, pch=16, highlight.3d=TRUE,  
  type="h", main="3D Scatterplot")



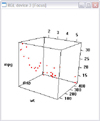
# 3D Scatterplot with Coloring and Vertical Lines  
# and Regression Plane   
library(scatterplot3d)   
attach(mtcars)   
s3d <-scatterplot3d(wt,disp,mpg, pch=16, highlight.3d=TRUE,  
  type="h", main="3D Scatterplot")  
fit <- lm(mpg ~ wt+disp)   
s3d$plane3d(fit)



### Spinning 3D Scatterplots

You can also create an interactive 3D scatterplot using the **plot3D(**x**,** y**,** z**)**function in the rgl package. It creates a spinning 3D scatterplot that can be rotated with the mouse. The first three arguments are the **x, y, and z** numeric vectors representing points. **col=** and size= control the color and size of the points respectively.

# Spinning 3d Scatterplot  
library(rgl)  
  
plot3d(wt, disp, mpg, col="red", size=3)



You can perform a similar function with the **scatter3d(**x**,**y**,**z**)** in the **Rcmdr**package.

# Another Spinning 3d Scatterplot  
library(Rcmdr)  
attach(mtcars)  
scatter3d(wt, disp, mpg)

