```
##  THE R BOOK

objects() #to see what variables you have created.
search() #to see which libraries and dataframes are attached.
rm(list=ls()) #remove all variables

sum(x<5) # the total counts of data < 5.
sum(x[x<5]) #the total sum of data < 5.

rep(1:4,2)              # [1] 1 2 3 4 1 2 3 4
rep(1:4, each=2)        #[1] 1 1 2 2 3 3 4 4
rep(1:4,each=2,times=3) #[1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
rep(1:4,1:4)            #[1] 1 2 2 3 3 3 4 4 4 4
rep(1:4,c(4,1,4,2))     #[1] 1 1 1 1 2 3 3 3 3 4 4

seq(a,b,along=x) #to get the same length as x, whitout know the step.

rank(); sort(); order()       #the following example illustrate the differences.
(view=data.frame(houses$Price,ranks,sorted,ordered))
     #put the function in a parenthese resulting the print of
     #the results on the screen, which saves an extra line of code.
   houses.Price ranks sorted ordered
1           325  12.0     95       9
2           201  10.0    101       6
3           157   5.0    117      10
4           162   6.0    121      12
5           164   7.0    157       3
6           101   2.0    162       4
7           211  11.0    164       5
8           188   8.5    188       8
9            95   1.0    188      11
10          117   3.0    201       2
11          188   8.5    211       7
12          121   4.0    325       1
with(houses,Location[order(Price)])
 [1] Reading     Staines     Winkfield   Newbury     Bracknell   Camberley
 [7] Bagshot     Maidenhead  Warfield    Sunninghill Windsor     Ascot

rownames(x)=rownames(x,do.NULL=FALSE,prefix="Trail.") #Trail1,Trail2,...Trailn.

match(vector1,vector2) #where do the values in the 2nd vector appear in the 1st vector?

var.test(a,b)#Fisher's F test of Variance Ratio

ifelse()  #ifelse(y<0,-1,1): if y<0, then -1, otherwise 1.

#significant star
options(show.signif.stars=FALSE) #not show the significant star

apply() #used for appling functions to the rows or columns of metrices or dataframes.
apply(x,1,mean) #rows
apply(x,2,mean) #columns
sapply() # apply a function to a vector
sapply(3:6,seq) #generate a list of sequences from 1:3 up to 1:6
lapply() #apply a function to each of the elements of a list in turn.
lapply(x,length) #get the length of every component in the list.

rle(x) #looking for runs of numbers within vectors,"run length encoding" .
(possion=rpois(30,0.7)) #[1] 1 1 0 0 4 1 0 0 1 2 0 0 0 0 2 0 0 1 2 0 1 0 1 2 0 1
rle(possion)
```

```
  Run Length Encoding
  lengths: int [1:20] 2 2 1 1 2 1 1 4 1 2 ...
  values : num [1:20] 1 0 4 1 0 1 2 0 2 0 ...
max(rle(possion)[[1]])    #[1] 4
which(rle(possion)[[1]]==4)     #[1]  8 20
rle(possion)[[2]][c(8,20)]     #[1] 0 0
length(rle(possion)[[2]])       #[1]  20

####################
SAVE
####################
nbnumbers=rnbinom(1000,size=1,mu=1.2)
write(nbnumbers,"nbnumbers.txt",1)    #1 means one column, the default is 5 cols.
(nbtable=table(nbnumbers))
 nbnumbers
   0   1   2   3   4   5   6   7   8   9  10
 454 255 124  75  35  30  11   6   6   3   1
write.table(nbtable,"table.txt",col.names=F,row.names=F)
           #save both the counts and their frequencies in adjacent cols.
write.table(unclass(nbtable),"table.txt",col.names=F,row.names=F)
           #only save the frequencies as a single column.

#write a vector in R and then pasti into Excel
writeClipboard(as.character(factor.nale)) # go into Excel, then Ctrl+V in one col.
writeClipboard(as.character(numeric.variables)) #go to Excel, Ctrl+V

#writing an Excel readable file from R (from a dataframe)
write.table(data,"clipboard",sep="\t",col.names=NA) #then go to Excel, paste.

##most numbers are rounded to 53 binary digits accuracy.

###pattern matching
## the start of the character string is denoted by ^,
   #and the end of the character string is denoted by $.
## dot . as anything character.
wf=read.table("worldfloras.txt",header=T)
names(wf)  #[1] "Country""Latitude""Area" "Population" "Flora" "Endemism" "Continent"
grep() #returns the subscript indicating which character strings within a vector
       #of character strings contained a special character.
as.vector(Country[grep("R",as.character(Country))])
          # all the country names containing "R"
as.vector(Country[grep("^R",as.character(Country))])
          # all the country names starting with "R"
as.vector(Country[grep(" R",as.character(Country))])
          # blank R, all the country's second names starting with "R"
as.vector(Country[grep(" ",as.character(Country))])
          # blank, all the country names have two names.
as.vector(Country[grep("y$",as.character(Country))])
          # all the country names ended with "y" .
as.vector(Country[grep("[C-E]",as.character(Country))])
          # all the country names containing "C:E"
as.vector(Country[-grep("[a-t]$",as.character(Country))])
          # all the country names not ended with "a:t".
as.vector(Country[grep("^.y",as.character(Country))])
          # all the country names' second character is "y"
as.vector(Country[grep("^..y",as.character(Country))])
          # all the country names' third character is "y"
as.vector(Country[grep("^.{5}y",as.character(Country))])
          # all the country names' sixth character is "y".
as.vector(Country[grep("^.{,4}$",as.character(Country))])
```

```
          # all the country names has less than 4 characters. {,4}:repeat up to 4.

###substituting text within character strings
text=c("foot","elbow","hand")
sub("o","O",text)   #[1] "fOot"  "elbOw" "hand"
             ##sub only substitude the first character within one character string.
gsub("o","O",text)   #[1] "fOOt"  "elbOw"  "hand"
             ##gsub substitude all characters in the charater string.
gsub("^.","O",text)  #[1] "Ooot"  "Olbow"  "hand"
grep("o",text) #[1] 1  2
text[grep("o",text)] #[1] "foot"  "elbow"

###location of the pattern within a vector of character strings using regexpr
regexpr("o",text)  # if the character has no such letter, return -1.
   [1]  2  4 -1
   attr(,"match.length")
   [1]  1  1 -1
   attr(,"useBytes")
   [1] TRUE

#####Dates and times in R     two basic classes of time: POSIXct, POSIXlt.
Sys.time()    #[1] "2011-12-29 23:38:54 CST"
date()        #[1] "Thu Dec 29 23:46:25 2011"
unclass(Sys.time())   # [1] 1325224110 POSIXct the number of seconds since 1 Jan1970.
time=as.POSIXlt(Sys.time())   # [1] "2011-12-29 23:51:36 CST"
unlist(unclass(time))
    # Sunday is wday 0, January is scored as month 0. years are scored as post-1900.
      sec       min      hour      mday       mon      year      wday      yday     isds
      # is daylight savings time in operation?
 36.68441  51.00000  23.00000  29.00000  11.00000 111.00000   4.00000 362.00000   0.0000
time1=as.POSIXlt(Sys.time())
time1-time   ## time+number; time-number;time1-time2; time1 "logical operation" time2

difftime("2005-10-21","2003-8-15")    #Time difference of 798 days
as.numeric(difftime("2005-10-21","2003-8-15")) #[1] 798
as.difftime(c("0:3:20","11:23:15")) #convert character strings into difftime objects.
    Time differences in mins
    [1]   3.333333 683.250000

##############################################################################
Data input
##############################################################################
x=c()
x=scan() #can also copy a column of Excel and then paste
x=read.table("name.txt",header=T)
x=read.delim("name.txt")  # do not need to type "header=T"
data=read.table(file.choose(),header=T)  # choose files.
file.exists("") #check whether the file exists

data()  #built-in data sets.
try(data(package="MASS"))   #find the names of data sets in a package.

length(scan("rt.txt",sep="\n"))    # get the number of lines of the data
strsplit(phrase,split="") #string-split.
readLines("data.txt",n=-1)
            #read each line from a file, n=-1 means read to the end of the file.

##############################################################################
Dataframes
##############################################################################
```

```
All the values of the same variable must go in the same column.
data.frame(y,x,z) # the sequences
as.data.frame() #e.g. y=rpois(1500,1.5); table(y); as.data.frame(table(y))
short=as.data.frame(table(y))
long=as.data.frame(lapply(short,function(x)rep(x,short$Freq)))
            #anonymous function to generate the repeats of each row by the freq
long[,1]

worms=read.delim("worms.txt")
with(worms,by(worms,Vegetation,mean))
                        # SUMMARY of the dataframe on the basis of factor levels.
aggregate(worms[,c(2,3,5,7)],by=list(veg=worms$Vegetation,d=worms$Damp),mean)
#### Using subscripts
### Ordering dataframe
worms[sample(1:20,8),]#random sample some rows with replacement. Default is replace=F
worms[order(worms$Slope),]   #if ascending order, use rev()
worms[order(worms$Vegetation,worms$Worm.density),]
     #first order by Vegetation, then in the same veg type, order by second variable.
worms[order(worms$Vegetation,-worms$Worm.density),]
     # different direction for worm.density. Minus only work for numerical variables.
worms[order(-rank(worms$Vegetation),-worms$Worm.density),]
     #for factor levels,use the rank function to make the lavels numerical.
worms[order(worms$Vegetation,worms$Worm.density),c(4,7,5,3)]
   Vegetation Worm.density Soil.pH Slope
8      Arable            4     4.8     0
18     Arable            5     4.5     2
2      Arable            7     5.2     2
14  Grassland            0     3.8     6
12  Grassland            1     4.1     1
19  Grassland            1     3.5     1
3   Grassland            2     4.3     3
6   Grassland            2     3.9     2
13  Grassland            2     4.0     2
7   Grassland            3     4.2     3
1   Grassland            4     4.1    11
10  Grassland            7     5.0     4
## using logical conditions to select rows from the dataframe.
worms[worms$Damp==T,]
worms[!worms$Damp==F,]
worms[-which(Damp==F),]
worms[!duplicated(worms$Vegetation),]
       # eliminate pseudoreplication. only one row for each vegetation type.
dataframe[duplicated(dataframe),]
       # view the rows that are dulicates in a dataframe (if any)
unique() #eliminating duplicate rows from a dataframe
worms[-(6:15),]  # Drop some rows.
worms[!(Vegetation=="Grassland"),]
worms[Worm.density>median(Worm.density)&Soil.pH<5.2,] # condition 1 AND condition 2
worms[,sapply(worms,is.numeric)]
       #to extract all cols that contain numbers from the dataframe
worms[,sapply(worms,is.factor)]
## Omitting rows containing missing valuues, NA
na.omit(dataframe)
complete.cases(dataframe) # test for the presence of missing values across a dataframe.
dataframe[complete.cases(dataframe),]
                        # the dataframe without missing values. same as na.omit
apply(apply(dataframe,2,is.na),2,sum)  #sum up the missing values in each variable.
  Field.Name         Area        Slope   Vegetation      Soil.pH         Damp Worm.densi
           0            1            1            0            1            1
worms[,grep("S",names(worms))] #select cols name contained "S"
```

```
## using the match function in dataframes
herbicides=read.delim("herbicides.txt")
        Type Herbicide
1   Woodland  Fusilade
2    Conifer  Weedwipe
3     Arable  Twinspan
4       Hill  Weedwipe
5    Bracken  Fusilade
6      Scrub  Weedwipe
7  Grassland  Allclear
8      Chalk  Vanquish
9     Meadow  Propinol
10      Lawn  Vanquish
11   Orchard  Fusilade
12     Verge  Allclear
worms$hb=herbicides$Herbicide[match(worms$Vegetation,herbicides$Type)]
                                          #each vegetation type's herbicide type
## merge dataframes
merge(frame1,frame2,all=T) #with NA inserted when missing values.
                          #Default only contain those rows complete in both frames.
merge(x,y,by.x=c("name","name1"),by.y=c("name2","name3"))
                     #name and name2 are the same variables, and name1 and name3

#############################################################################
Graphics
#############################################################################
plot(x,y) # Cartesian
plot(y~x) #formula
data1=read.delim("scatter1.txt"); data1=read.delim("scatter2.txt")
plot(c(data1$xv,data2$xv2),c(data1$ys,data2$ys2),xlab="x",ylab="y",xlim=c(0,100),
     ylim=c(0,80),type="n")
      #if plot seperately, the axis scale will be only from the first dataset.
points(data1$xv,data1$ys,col="red")
      #col used by plot: 1: black(default),2:red,3:green,4:blue,5:pale blue,6:purple
points(data2$xv2,data2$ys2,col="blue")
abline(lm(ys~xv)); abline(lm(data2$ys2~data2$xv2))
legend(locator(1),c("treatment","control"),pch=c(1,1),col=c(2,4))# locator(1)
     #allow you to select exactly where on the plot the legend box should be placed.
##pch
plot(0:10,0:10,type="n",xlab="",ylab="")
k=-1
for(i in c(2,5,8)){
  for (j in 0:9){
  k=k+1
  points(i,j,pch=k,cex=2)}}

data=read.delim("pgr.txt")  #names: [1] "FR"  "hay" "pH"
plot(hay,pH)
text(hay,pH,labels=round(FR,2),pos=1,offset=0.5,cex=0.7)
plot(hay,pH,pch=16,col=ifelse(FR>median(FR),"red","green"))
###Drawing mathematical functions
curve(x^3-3*x,-2,2)

###adding other shapes to a plot
plot(0:10,0:10,xlab="",ylab="",xaxt="n",yaxt="n",type="n")
rect(6,6,9,9)
arrows(1,1,3,8)
arrows(1,9,5,9,code=3)    #double arrow angle=30 by default
arrows(4,1,4,6,code=3,angle=90)
```

```r
## function to draw by cursor
click.draw=function(){
    coos=c(unlist(locator(1)),unlist(locator(1)))
    arrows(coos[1],coos[2],coos[3],coos[4])}  #rect()
click.draw()
#locator() is a list.
locator(1) # $x  [1] 0.9157503 $y  [1] 6.858191
unlist(locator(1)) #        x         y
                   #        0.8456143 6.7510908
locations=locator(6)  #then click 6 points on the graphy
polygon(locations,col="lavender")  #polygon automatically closed the shape,
                                   #drawing a line from the last point to the first.

##fill area with polygon
x=seq(-3,3,0.01)
y=dnorm(x)
plot(x,y,type="l")
polygon(c(x[x<=-1],-1),c(y[x<=-1],y[x==-3]),col="red")
                    #the point(-1,y[x==-3])draw the polygon down to the x axis.

info=read.table("plotfit.txt",header=T)
plot(x,y,xlim=c(0,100),ylim=c(0,1.05*max(y)),pch=16)
model=nls(y~a*x*exp(-b*x),start=list(a=500,b=0.05))# nls:non-linear least squares.
                    #need to provide an initial gusstimate for the two parameters.
# from the model, a=480,b=0.047
xv=0:100       #100 points to calculate and plot the smoothed values of y.
lines(xv,predict(model,list(x=xv)),lty=2)
          #predict(). the first argument is a fitted model, second is a dataframe.
yv=480*xv*exp(-0.047*xv)
lines(xv,yv)

##fitting non-parametric curves through a scatterplot
# lowess (a non-parametric curve fitter)
# loess (a modelling tool)
# gam (fits generalized additive models)
# lm
plot(x,y)
lines(lowess(x,y)) # lowess

model=loess(y~x)
xv=0:50; yv=predict(model,data.frame(x=xv))  # loess
lines(xv,yv)

library(mgcv)
model=gam(y~s(x))       # gam
xv=0:50; yv=predict(model,list(x=xv))
lines(xv,yv)

model=lm(y~x+I(x^2)+I(x^3))    # lm cubic
xv=0:50; yv=predict(model,list(x=xv))
lines(xv,yv)

## joining the dots...
sm=read.table("smoothing.txt",header=T)
sequence=order(sm$x)    ensure that the points on the x axis are ordered!
plot(x[sequence],y[sequence])
lines(x[sequence],y[sequence])

## Categorical explanatory variables
# In R, for categorical exp variables use boxplot by default in plot.
```

```r
#categorical variables are factors with two or more levels.
weather=read.table("SilwoodWeather.txt",header=T) #names: upper, lower,rain, month,yr
month=factor(month)
plot(month,upper)
boxplot(month,upper,notch=T)
# box-and-whisker plot: the horizontal line is the median,
    #the upper and lower of box: 75th, 25th perctile
# whiskers: either the maximun/minimum value or 1.5 IQR (75th-25th).
    #Outliers can show skewness also.
# notch=+-1.58IQR/sqrt(n) roughly 95% Confident intervals of median,
    #based on normal distribution and same n and ...

data=read.table("box.txt",header=T)
index=order(tapply(response,fact,mean))
                          #rank the mean values, so we can see diffrences easily.
ordered=factor(rep(index,rep(20,8)))
boxplot(response~ordered,notch=T,names=as.character(index))
model=aov(response~factor(fact))
plot(TukeyHSD(model))

#####Plots for single samples
# histograms to show a frequency distribution
# index plots to show the values of y in sequence
# time-series plots
# compositional plots like pie diagrams

# overlaying histograms with smooth density functions
y=rnbinom(158,mu=1.5,size=1)
bks=-0.5:max(y)+0.5
hist(y,bks,main="")
xv=0:11
yv=dnbinom(xv,size=mean(y)^2/(var(y)-mean(y)),mu=mean(y))
lines(xv,yv*158)

# Density estimation for continuous variables
# the rule of thumb for bandwidth is b = (max(x)-min(x))/(2*(1+log(length(x),bas
library(MASS)
attach(faithful)
par(mfrow=c(1,2))
hist(eruptions,15,freq=F,main="",col=27)
lines(density(eruptions,width=0.6,n=200))
truehist(eruptions,nbins=15,col=27)    # truehist
lines(density(eruptions,n=200))       # default width = b

# index plot
# useful for single csamples, plot tkes a single argument which is a continuous
# and plot the values on the y axis, with the x coordinate determined by the pos
    # of the number in the vector.
response=read.delim("das.txt")
plot(response$y)
which(response$y>20)

# time series plots
ts.plot()
plot.ts()

#pie charts
data=read.csv("piedata.csv") # names: names amounts
pie(data$amounts,labels=as.character(data$names))
```

```r
# the stripchart function
# for very small sample size data, which is not suitble for box-and whisker
# plot all individual points whitin a group
data(OrchardSprays)
with(OrchardSprays, stripchart(decrease~treatment,vertical=T,log="y"))

######## plots with multiple variables
#pairs  for a matrix of scatterplots of every variable against every other
#coplot for conditioning plots where y is plotted against x for different values
#xyplot where a set of panel plots is produced

# pairs
ozones=read.table("ozone.data.txt",header=T)  #names: rad temp wind ozone
with(ozones,pairs(ozone,panel=panel.smooth))
                #  panel=panel.smooth adda a non-parametric smoother to the plot

# coplot
with(ozones,coplot(ozone~wind|temp,panel=panel.smooth))

#interction plots
yields=read.table("splityield.txt",header=T)
with(yields, interaction.plot(fertilizer,irrigation,yield))
     # the first argument: x axis; second:trace or group variable; third: response
# special plots
library(lattice)
xyplot(y~x1|x2)

plot.design(y~x1*x2*x3) # default is the plot mean, can add 'fun="sd" '

#bubble plot (explantroy1,explantory2,response)
bubble.plot=function(xv,yv,rv,bs=0.1){
r=rv/max(rv)
yscale=max(yv)-min(yv)
xscale=max(xv)-min(xv)
plot(xv,yv,type="n",xlab=deparse(substitute(xv)),ylab=deparse(substitute(yv)))
for(i in 1:length(xv)) bubble(xv[i],yv[i],r[i],bs,xscale,yscale) }

bubble=function(x,y,r,bubble.size,xscale,yscale){
theta=seq(0,2*pi,pi/200)
yv=r*sin(theta)*bubble.size*yscale
xv=r*cos(theta)*bubble.size*xscale
lines(x+xv,y+yv)}

dd=read.table("pgr.txt",header=T)
with(dd, bubble.plot(hay,pH,FR))

#plots with many identical values. esp counting numbers
numbers=read.table("longdata.txt",header=T)
with(numbers,sunflowerplot(xlong,ylong,col=27))

#############################################################################
changing the look of graphics
#############################################################################
# many of the changes that you want to make to the look of your graphics will in
# the use of the graphic parameters function, par. other changes, however, can b
# made through alterations to the arguments to high-level functions such as plot
# points,lines, axis,title and text.

#shading
dd=read.table("F:\\therbook\\pgr.txt",header=T)
```

```
attach(dd)
plot(hay,pH)
x=hay[FR>5]
y=pH[FR>5]
polygon(x[chull(x,y)],y[chull(x,y)],col="red")
x=hay[FR>10]
y=pH[FR>10]
polygon(x[chull(x,y)],y[chull(x,y)],col="green")
x=hay[FR>20]
y=pH[FR>20]
polygon(x[chull(x,y)],y[chull(x,y)],density=10,angle=90,col="blue")
polygon(x[chull(x,y)],y[chull(x,y)],density=10,angle=0,col="blue")
points(hay,pH,pch=16)

# Logarithmic axes: You can transform the variables inside the plot function
#(e.g. plot(log(y)~x)) or you can plot the untransformed variables on
#logarithmically scaled axes (e.g. log=``x" , log=``y" , log=``xy" ).

#Axis labels containing subscripts and superscripts, you need to use expression
#function. plot(1:10, 1:10, ylab=expression(r^2), xlab=expression(x[i]), type=``
#axis labels
plot(1:10,1:10,xlab=expression(x[i]),ylab=expression(r^2),type="n")
par(family="sans")    #fonts: family="sans"(default),"serif","mono","symbol"
text(5,8,"this is the default font")
par(family="serif")
text(5,6,"this is the serif font")
par(family="mono")
text(5,4,"this is the mono font")
par(family="sans")
 cd=0.64
text(locator(1),as.expression(substitute(r^2==cd,list(cd=cd))))
text(locator(1),expression(r^2==0.64))
text(locator(1),expression(chi^2==0.64))
text(10,2,substitute(chi^2=="0.68"))
text(locator(1),expression(paste(frac(1,sigma*sqrt(2*pi))," ",
     e^{frac(-(x-mu)^2,2*sigma^2)})))
# normal density formula
text(locator(1),expression(hat(y) %+-% se))  #%+-%

# mathmematical symbols on plots
x=seq(-4,4,len=101)
plot(x,sin(x),type="l", xaxt="n",xlab=expression(paste("phase angle",phi)),
    ylab=expression("sin"*phi))
axis(1, at=c(-pi,-pi/2,0,pi/2,pi),lab=expression(-pi,-pi/2,0,pi/2,pi))
text(-pi/2,0.5,substitute(chi^2=="24.5"))

#phase planes
plot(c(0,1),c(0,1),xlab="",ylab="",xaxt="n",yaxt="n",type="n")
abline(0.8,-1.5);abline(0.6,-0.8,lty=2)
axis(1,at=0.805,lab=expression(1/alpha[21])) #x axis
axis(1,at=0.56,lab=expression(1/alpha[11]))
axis(2,at=0.86,lab=expression(1/alpha[12]),las=1) #y axis
axis(2,at=0.63,lab=expression(1/alpha[22]),las=1)
text(0.05,0.85,expression(paste(frac("d N"[1],"dt"),"=0")))
text(0.78,0.07,expression(paste(frac("d N"[2],"dt"),"=0")))
arrows(-0.02,0.72,0.05,0.72,length=0.1)
arrows(-0.02,0.72,-0.02,0.65,length=0.1)
arrows(-0.02,0.72,0.05,0.65,length=0.1)

### trellis plots
```

```r
library(lattice)  # need lattice package
xyplot(y~x|subject)  # the most commonly use trellis plot. panles are default drawn
  #starting from the bottom left-hand corner, going right and then up, unless
  #as.table=TRUE, in which case panels are drawn from the top left-hand corner, go
  #right then down. the base R graphics functions like lines will not work in a la
  #panel function.
results=read.table("fertilizer.txt",header=T) #"root" "week" "plant" "fertilizer"
xyplot(root~week|plant,pch=16)
xyplot(root~week|plant,panel=function(x,y){    #use panel function o make more changes
                       panel.xyplot(x,y,pch=16)
                       panel.abline(lm(y~x))})
bwplot(y~x|subject)  # box and whisker plot
barchart(yield~variety|site, data=barley,groups=year,layout=c(2,3),stack=T,
         scales=list(x=list(rot=45)),col=c(2,3)) #scales to rotate the long tabels
ee=equal.count(ethanol$E,number=9,overlap=1/4)
xyplot(NOx~C|ee,data=ethanol,layout=c(9,1),panel=function(x,y){
                       panel.grid(h=-1,v=2)
                       panel.xyplot(x,y,pch=16)
                       panel.abline(lm(y~x))})
# histograms
months=month.abb  # or month.name
datas=read.delim("SilwoodWeather.txt") #"upper" "lower" "rain"  "month" "yr"
attach(datas)
wet=(rain>0)
wd=as.vector(tapply(wet,list(datas$yr,datas$month),sum))
mos=factor(as.vector(tapply(datas$month,list(datas$yr,datas$month),mean)))
histogram(~wd|mos,type="count",xlab="rainy days",breaks=-.5:28.5,
          strip=strip.custom(factor.levels=months))

xyplot(decrease~treatment, OrchardSprays, groups=rowpos,type="a",
       auto.key=list(space="right",points=F,lines=T))
bwplot(decrease ~ treatment, OrchardSprays, groups = rowpos,
       panel = "panel.superpose", #each group be drawn in a different colour
       panel.groups = "panel.linejoin",#dots joined by lines for each member
       xlab = "treatment",              #of the group
       key = list(lines = Rows(trellis.par.get("superpose.line"),
                  c(1:7, 1)),
                  text = list(lab = as.character(unique(OrchardSprays$rowpos))),
                  columns = 4, title = "Row position"))

# 3-D plots
data=read.delim("pgr.txt")  #"FR"  "hay" "pH"
install.packages("akima")
library(akima)
zz=interp(hay,pH,FR) #to implement bivariate interpolation onto a grid for irregularly
# spaced input data. 1st and 2nd: explanatory variables, 3rd: response variable
#four functions: contour, filled.contour, image or persp.
# colors. heat.colors; topo.colors
image(zz,col=topo.colors(12),xlab="biomass",ylab="ph")
contour(zz,add=T)
filled.contour(zz,col=topo.colors(24))
persp(zz,xlab="biomass",ylab="ph",zlab="festuca rubra",theta=30,phi=30,col="lightblue")
#theta gives the azimuthal direction,phi gives the colatitude.
# 3-D images of mathematical functions by the outer function without using inter
x=seq(0,10,.1);y=seq(0,10,.1)
func=function(x,y)3*x*exp(.1*x)*sin(y*exp(-.5*x))
image(x,y,outer(x,y,func))
contour(x,y,outer(x,y,func),add=T)

wireframe(volcano,shade=T,aspect=c(61/87,.4),screen=list(z=-120,x=-45),
```

```
          light.source=c(0,0,10),distance=.2,
          shade.colors=function(irr,ref,height,w=.5) grey(w*irr+(1-w)*(1-(1-ref)^0.4)))

n=50;tx=matrix(seq(-pi,pi,len=2*n),2*n,n)
ty=matrix(seq(-pi,pi,len=n)/2,2*n,n,byrow=T)
xx=cos(tx)*cos(ty); yy=sin(tx)*cos(ty);zz=sin(ty);zzz=zz;zzz[,1:12*4]=NA
wireframe(zzz~xx*yy,shade=T,light.source=c(3,3,3))

#To inspect the current values of any of the graphics parameters (par),
#type the name of the option in double quotes.
par("usr")# to see the current limits of the x and y axes. x-min, x-max, y-min, y-max.
par("mar") # to see the sizes of the margins.
default.parameters=par(no.readonly=T)
par()...
par(default.parameters)

par(adj=0) #adj=0: left adjustification, =0.5 center, =1, right
text(..., adj=c(1,0)) # vary justification in x and y directions independently.
par(ann=F) #remove annotation of graphs, labels and main titles.
par(ask=T) # delay moving on to the next in a series of plot.
par(bg="cornsilk") #backgroud colour, the default is "transparent"

#axis
plot(1:10,10:1,type="n",axes=F,xlab="",ylab="") #axes=F to draw graph with no axes.
axis(1,1:10,LETTERS[1:10],col.axis="blue")
axis(2,1:10,letters[10:1],col.axis="red")
axis(3,lwd=3,col.axis="green")
axis(4,at=c(2,5,8),labels=c("one","two","three"))
#box around plots, bty
par(mfrow=c(4,2))
plot(1:10,10:1);plot(1:10,10:1,bty="n");plot(1:10,10:1,bty="]")
plot(1:10,10:1,bty="c");plot(1:10,10:1,bty="u");plot(1:10,10:1,bty="7")
plot(1:10,10:1,bty="l")  ;par(mfrow=c(1,1))
#size of plotting symbols using the character expansion function, cex
plot(0:10,0:10,type="n",xlab="",ylab="")
for (i in 1:10) points(2,i,cex=i)
for (i in 1:10) points(6,i,cex=(10+(2*i)))

#colour specification
colors()  # to see all 657 colours available in R
#rrggbb
palette()[i]
#creat your own colors in palette
palette(c("wheat1","wheat2","beige","bisque1"));pie(1:4,col=palette())
palette("default") #reset the palette back to the default use.
pie(rep(1,12),col=gray(seq(.1,.8,length=12)),main="gray")
pie(rep(1,12),col=rainbow(12),main="rainbow")
pie(rep(1,12),col=terrain.colors(12),main="terrain.colors")
pie(rep(1,12),col=heat.colors(12),main="heat.colors")

#color and font
plot(1:10,1:10,xlab="x axis labels",ylab="y axis labels",pch=16,col="orange",
     col.lab="green4",col.axis="blue",col.main="red",main="title",
     col.sub="navy",sub="subtitle", las=1,font.axis=3,font.lab=2,font.main=4,
     font.sub=3)

#foreground colours, fg. change the color of such things as axes and boxes aroun
#plots use the fg
par(mfrow=c(2,2))
plot(1:10,1:10,xlab="x axis labels",ylab="y axis labels")
```

```r
plot(1:10,1:10,xlab="x axis labels",ylab="y axis labels",fg="blue")
plot(1:10,1:10,xlab="x axis labels",ylab="y axis labels",fg="red")
plot(1:10,1:10,xlab="x axis labels",ylab="y axis labels",fg="green")

x=rnorm(1000);hist(x,col="lavender",main="")  #color with histograms
#changing the shape of the plotting region, plt
par(plt=c(0.15,0.94,0.3,0.7));plot()
#locating multiple graphs in non-standard layouts using fig
par(fig=c(0,.5,0,.5));plot(1:10,1:10)  #c(xmin,xmax,ymin,ymax). [0,1]propotrtion
par(fig=c(.5,1,.5,1),new=T);plot(1:10,1:10)#just like plot, will clean,need new=T
#two graphs with a common x scale but different y scales using fig
par(fig=c(0,1,.5,1));par(mar=c(0,5,2,2));plot(x,y,xlab="",xaxt="n")
par(fig=c(0,1,0,.5),new=T);par(mar=c(5,5,0,2));plot(x,y)

# layout function
layout(matrix,width,heights,respect=FALSE)
#use 0 to indicate locations where you do not want to put a graph.
nf=layout(matrix(c(2,0,1,3),2,2,byrow=T),c(3,1),c(1,3),T)
layout.show(nf)   #barplot(y,horiz=T)

#creating and controlling multiple screens on a single device
screen(1) #to choose which screen to draw in.
erase.screen() #to clear a single screen
close.screen()# all=TRUE to close all
fig.mat=c(0,1,.7,1,0,.5,0,.7,.5,1,.35,.7,.5,1,0,.35)#creat matrix each row
                                                    #descript the figs of each screen
fig.mat=matrix(fig.mat,nrow=4,byrow=T)
split.screen(fig.mat)
screen(1);plot()
screen(2);plot()...

#orientation of numbers on the tick marks, las
#shapes for the ends of lines, lend
par(mfrow=c(3,1))
plot(1:10,1:10,type="n",axes=F,ann=F)
lines(c(2,9),c(5,5),lwd=38);text(5,1,"rounded ends")
par(lend="square")
plot(1:10,1:10,type="n",axes=F,ann=F)
lines(c(2,9),c(5,5),lwd=38);text(5,1,"aquare ends")
par(lend="butt")
plot(1:10,1:10,type="n",axes=F,ann=F)
lines(c(2,9),c(5,5),lwd=38);text(5,1,"butt ends")
#two graphs on the same plot
plot();par(new=T);plot()
#outer margins. There is an area outside the margins of the plotting area called
par(oma=c(0,0,0,0))
#packing graphs closer together
par(mfrow=c(3,3));par(mar=c(0.2,.2,.2,.2)) #0.2 lines looks best for tightly plots.
par(oma=c(5,5,1,1))
for(i in 1:9) plot(sort(runif(100)),sort(runif(100)),xaxt="n",yaxt="n")
title(xlab="time",ylab="distance",outer=T,cex.lab=2)
#outer=T in title to write the titles in the outer margin.

#character rotation, srt. "srt"="string rotation" counter-clockwise rotation
srt=(degrees)
plot(1:10,1:10,type="n");
for(i in 1:10)text(i,i, LETTERS[i],srt=(i*20))

#rotating the axis labels
  #when you have long labels it is good to rotate 45 degrees
```

```
spending=read.delim("spending.txt") #spend, country
attach(spending)
par(mar=c(7,4,4,2)+.1)
xvals=barplot(spend,ylab="spending")
text(xvals,par("usr")[3]-0.25,srt=45,adj=1,labels=country,xpd=T)

# tick marks on the axes
tcl=-0.5 #the default setting, negative values put the tick marks outside the box.
  #tcl gibes the length of tick marks as a fraction of the height of a line of tex
tck=NA #tck=0 means no tick marks, tck=1 means fill the whole frame (makes a grid)
par(mfrow=c(2,2))
plot(1:10,1:10,type="b",xlab="",ylab="")
plot(1:10,1:10,type="b",xlab="",ylab="",tck=1)
plot(1:10,1:10,type="b",xlab="",ylab="",tck=0)
plot(1:10,1:10,type="b",xlab="",ylab="",tck=-0.3)

#axis styles
axis  #select one of the four sides of the plot to work with.
xaxs  #intervals for the tich marks. style "r"(regular) first extend the data range
      #by 4% then finds an axis with pretty labels that fits within the range.
      #style "i"(internal) only find an axis with pretty labels.
xaxt  #suppress production of the axis with xaxt="n" when you want to specify your
      #own kind of axes with different locations for tick marks and/or diff labels.



################################################################################
Tables
################################################################################
convey details: tables. show effects: graphics
data=read.delim("Daphnia.txt")
attach(data)
tapply(response, explantory,function,na.rm=F)
tapply(Growth.rate,list(Daphnia,Detergent),mean) #1st variable:row; 2nd: column
  BrandA   BrandB   BrandC   BrandD
Clone1 2.732227 2.929140 3.071335 2.626797
Clone2 3.919002 4.402931 4.772805 5.213745
Clone3 5.003268 4.698062 4.019397 2.834151
tapply(Growth.rate,list(Daphnia,Detergent,Water),mean) #1st variable:row; 2nd: column
 # the result is a list. 3rd: list
ftable(tapply(Growth.rate,list(Daphnia,Detergent,Water),mean)) #better layout ftable()
               Tyne     Wear
Clone1 BrandA  2.811265 2.653189
       BrandB  2.775903 3.082377
       BrandC  3.287529 2.855142
       BrandD  2.597192 2.656403
Clone2 BrandA  3.307634 4.530371
       BrandB  4.191188 4.614673
       BrandC  3.620532 5.925078
       BrandD  4.105651 6.321838
Clone3 BrandA  4.866524 5.140011
       BrandB  4.766258 4.629867
       BrandC  4.534902 3.503892
       BrandD  3.365766 2.302537

table(x): #table the values and their frequences.

cells=rnbinom(10000,size=.63,prob=0.63/1.83)
gender=rep(c("female","male"),c(5000,5000))
table(cells,gender)
```

```
      gender
cells female male
   0    2583 2581
   1    1083 1051
   2     534  556
   3     319  313
tapply(cells,gender,mean)

# expanding a table into a dataframe
count.table=read.delim("tabledata.txt")
  count    sex   age   condition
1    12   male young     healthy
2     7   male   old     healthy
3     9 female young     healthy
4     8 female   old     healthy
5     6   male young parasitized
6     7   male   old parasitized
7     8 female young parasitized
8     5 female   old parasitized
tableframe=as.data.frame(lapply(count.table,function(x)rep(x,count.table$count)))
tableframe=tableframe[,-1]
   sex   age   condition
1    male young     healthy
2    male young     healthy
3    male young     healthy
4    male young     healthy
5    male young     healthy
...
61 female   old parasitized
62 female   old parasitized
table(tableframe) #converting from a dataframe to a table
as.data.frame(table(tableframe))
# calculating tables of proportions
count=matrix(c(2,2,4,3,1,4,2,0,1,5,3,3),nrow=4)
prop.table(matrix.name,margins) #margins: 1 for row, 2 for column
colSums(prop.table(matrix.name,margins))
scale(count)
 #scale: the mean of values whitin a column will be zero; can also get the sd of
#     [,1]      [,2]      [,3]
#[1,] -0.7833495 -0.439155 -1.224745
#[2,] -0.7833495  1.317465  1.224745
#[3,]  1.3055824  0.146385  0.000000
#[4,]  0.2611165 -1.024695  0.000000
#attr(,"scaled:center")
#[1] 2.75 1.75 3.00
#attr(,"scaled:scale")    ## this is the sd of cols
#[1] 0.9574271 1.7078251 1.6329932

# to generate tables of combinations of factor levels.
expand.grid(height=seq(60,80,5),weight=seq(100,300,50),sex=c("male","female"))
height weight    sex
1      60    100   male
2      65    100   male
3      70    100   male
4      75    100   male
5      80    100   male
6      60    150   male
7      65    150   male
...
45     80    250 female
```

```
46     60    300 female
47     65    300 female
48     70    300 female
49     75    300 female
50     80    300 female

data=read.table("parasites.txt",header=T)
#1   splendens
#2   knowlesii
#3    vulgaris
#4   knowlesii
#5     viridis
...
#150 splendens
names(data)="parasite"
vulgaris=factor(1*(parasite=="vulgaris"))
                              # (parasite=="vulgaris") is logical, *1:numerical
model.matrix(~parasite-1)
#  parasiteknowlesii parasitekochii parasitesplendens parasiteviridis
#1                 0              0                 1               0
#2                 1              0                 0               0
#3                 0              0                 0               0
#4                 1              0                 0               0
#...
#150               0              0                 1               0

##############################################################################################
Mathematics
##############################################################################################
#Draw smooth functions in R you need to generate a series of 100 or more regular
#spaced x values between min(x) and max(x).
# Gamma function
t=seq(.2,4,.01)
plot(t,gamma(t),type="l")
abline(h=1,lty=2)

#asymptotic function : y=ax/(1+bx)   x=0,y=0. x-infinite,y=a/b
#in ecology, it is called Holling's disc equation and shows predator feeding rat
#a function of prey density. also y=a(1-e^(-bx)) x=0,y=0. x-infinite,y=a
# parameter estimate in asymptotic functions, use the Reciprocal transformation:
# 1/y=(1+bx)/ax    1/y=1/ax+b/a
#lwt Y=1/y, X=1/x,A=1/a,C=b/a  then, Y=AX+C

#Sigmoid (S-shaped) functions
#the simplest S-shaped function is the two-parameter logistic, where,for 0<=y<=1
#y=e^(a+bx)/(1+e^(a+bx))
# three parameter logistic function: y=a/(1+be^(-cx))
# the intercept is a/(1+b), the asymptotic value is a and the initial slope is c

#normal distribution
# y=exp(-|x|^m)  when m=2, this is the normal distribution
#f(z)=(1/2*pi)*exp(-(z^2)/2)
means=numeric(10000)
for(i in 1:10000){means[i]=mean(runif(5)*10)}
hist(means,ylim=c(0,1600))
x=seq(0,10,.1);y=dnorm(x,mean(means),sd(means))*5000  #*5000 to get the same height
lines(x,y)

### Dice and normal distribution
par(mfrow=c(2,2))
```

```r
hist(sample(1:6,10000,replace=T),breaks=.5:6.5,main="",xlab="one dice")
a=sample(1:6,replace=T,10000)
b=sample(1:6,replace=T,10000);c=sample(1:6,replace=T,10000)
d=sample(1:6,replace=T,10000);e=sample(1:6,replace=T,10000)
hist(a+b,breaks=1.5:12.5,main="",xlab="two dice")
hist(a+b+c,breaks=2.5:18.5,main="",xlab="two dice")
hist(a+b+c+d+e,breaks=4.5:30.5,main="",xlab="two dice")
lines(seq(1,30,.1),dnorm(seq(1,30,.1),mean(a+b+c+d+e),sd(a+b+c+d+e))*10000)

par(mfrow=c(2,2))
curve(dnorm,-3,3,xlab="z",ylab="prob density",main="density")
curve(pnorm,-3,3,xlab="z",ylab="probility",main="cum prob")
curve(qnorm,0,1,xlab="p",ylab="quantile",main="quantiles")
hist(rnorm(1000),xlab="z",ylab="frequence",main="random numbers")

# hypergeometric distribution
dhyper(q,w,r,n)#q: a vector of counts of good ball in a sample;
               #w:good ball;r:bad ball;n:sample size
rhyper(m,w,r,n) # m:the number of hypergeo distribution random numbers to be generated.

x=0:12;freq=c(131,55,21,14,6,6,2,0,0,0,0,2,1)
barplot(freq,names=as.character(x),ylab="frequency",xlab="spores")

################## matrix
a=matrix(c(1,0,4,2,-1,1),nrow=3)
b=matrix(c(1,-1,2,1,1,0),nrow=2)
###matrix multiplication m*n dim matrix multiple n*p matrix = m*p matrix
a11 a12        b11  b12     a11b11+a12b21    a11b12+a12b22    c11  c12
          *               =                                 =
a21 a22        b21  b22     a21b11+a21b21    a21b12+a21b22    c21  c22

a%*%b  # In R, the symbol for matrix multiplication is %*%

#diagonals of matrices
ym=diag(1,3,3)
diag(ym)=1:3
diag(1:4,4,4)
m=cbind(x=1:5,y=rnorm(5));var(m) #a variance-covariance matrix

#determinant of matrix
a b
c d  =ad-bc
det(ym) #the matrix must have the same number of rows and columns

#inverse of matrix
library(MASS)
ginv(a)
ginv(ginv(a))



### solving systems of linear equations using matrix
3x+4y=12     # 3  4     x     12
x+2y=8       # 1  2  *  y  =   8
A=matrix(c(3,1,4,2),2,2) #coefficients
Y=matrix(c(12,8),2,1)    #known values
solve(A,Y)
      [,1]
[1,]   -4
```

```
[2,]    6

# calculus
D(expression(log(x)),"x")  #derivation

#Integrals
integrate(dnorm,0,Inf)
ff=function(x){1/((x+1)*sqrt(x))}
integrate(ff,0,Inf)
x=seq(0,10,.1)
plot(x,ff(x),type="l")

#############################################################################################
Classical tests
#############################################################################################
####### single samples: mean values? different from expect? normality?
summary()
fivenum()  # Tukey's five-number summary.

## normality
data=read.delim("das.txt")
plot(data$y)  #index plot hist(),boxplot()
qqnorm(data$y)
qqline(data$y,lty=2)
shapiro.test() # formal normality test

t.test(x)

## test
wilcox.test(x,mu=)  #non-parameterical test

# bootstrap test
B = 10000
n = length(salmon$mass)
mass.boot = apply(matrix(sample(salmon$mass, size = n * B, replace = T),
  nrow = B, ncol = n), 1, mean)
quantile(mass.boot, c(0.025, 0.975))

a=numeric(10000)
for(i in 1:10000) a[i]=mean(sample(x,replace=T))
quantile(a,c(0.025, 0.975))

## skew
m3=sum((x-mean(x))^3)/length(x)
s3=sqrt(var(x))^3
skew=m3/s3
se=sqrt(6/length(x))

## kurtosis
m4=m3=sum((x-mean(x))^4)/length(x)
s4=var(x)^2
kurtosis=(m4/s4)-3
se=sqrt(24/length(x))

###### two samples
 # comparing two variances (Fisher's F test, var.test)
   #for multiple samples you can choose between the Bartlett test and the
   #Fligner-Killeen test.
 # comparing two sample means with normal errors (t.test)
 # comapring two means with non-normal errors(wilcoxon's rank test,wilcox.test)
```

```r
 # comparing two proportions (the binomial test, prop.test)
 # correlating two variables (Pearson's or Spearman's rank correlation, cor.test)
 # testing for independence of two variables in contingency table(chisq.test,
    #fisher.test)

#test of homogeneity of variances
var.test(x,y) #do not need to compare variances first to put the bigger as numerator.
              #the variable name that came first in the alphabet on top as numerator.
              #this test is sensitive to outliers, so use it witht care.
refs=read.delim("refuge.txt")
bartlett.test(B[-31],T[-31])  #fisher F and bartlett sensitive to outliers.
fligner.test(B[-31],T[-31]) #not sensitive to outliers. It is a non-parametric test.
fligner.test(x,y,z)  #of the many tests for homo of variances, it is the nmost
                     #robust against departures from normality.

# comapreing two means: two sample student's t test; Wilcoxon's rank-sum test
# t test: independent,constant variances,normal distribution of errors
# wilcoxon: independent,not normal distribution of errors
t.test(x,y)
t.test(y~x) #x has two levels. the data frame has two columns,one for x and one for y.
wilcox.test(x,y) #typically, the t test will give the LOWER p-value.
#paired t test
t.test(x,y,paired=T)
# the sign test
binom.test(k,n) # null hypothesis: p=0.5
#binomial test to compare two proportions
prop.test(c(k1,k2),c(n1,n2))

#contingency tables. In statisics, contingencies are all the events that
  #could possibly happen.
#the assumption is they are independent!
chisq.test(table.name)
chisq.test(c(10,3,2,6)) #Chi-squared test for given probabilities.
                        #H0:p=0.25 for each cell
chisq.test(c(10,3,2,6),p=c(.2,.2,.3,.3))
die.count=table(ceiling(runif(100,0,6))) #estimate 100 times die
chisq.test(die.count) #whether this is a fair die? Ho:p=1/6 for each count (1:6)
fisher.test(x) #x is a matrix 2*2,with small expect values.

#correlation and covariance
r=cor(x,y)=cov(x,y)/sqrt(var(x)*var(y))
cor(dataframe) #reture a matrix of correlation
cor(vector1,vector2) #return a single number.
cor.test(x,y,method=c("pearson", "kendall", "spearman"))
                                          #k and s are non-parametric test

#scale-dependent correlations
productivity=read.delim("productivity.txt")
plot(x,y,ylab="mamal no.",xlab="productivity")
xyplot(y~x|f, panel=function(x,y){panel.xyplot(x,y,pch=16)
panel.abline(lm(y~x))})  #library(lattice)

#Kolmogorov-Smirnov test
#1, are two sample distribution the same? or are they significantly different fr
   #another in one or more ways?
#2,does a particular sample distribution arises from a particular
    #hypothesized distribution?
ks.test(x,y)

#power analysis for experiment design
```

```r
# rule of thumb:alpha=0.05, beta=0.2,power=0.8
#power.t.test  power calculation for one- and two-sample t test
#power.prop.test   power calculation two-sample test for proportions
#power.anova.test   power calculation for balanced one-way anova tests
 power.t.test() # see ?power.t.test
 power.t.test(delta=2,sd=5,type="one.sample",alternative="one.sided",power=0.8)

 # bootstrap
install.packages("boot")
library(boot)
data=read.delim("skewdata.txt")   #names: values
mymean=function(values,i)mean(values[i])
                    #the function is vital.write mean(values[i]) not mean(values)
myboot=boot(values,mymean,10000)   #(data,statistic,replicates)
boot.ci(myboot)

#############################################################################################
Statistical models
#############################################################################################
xnames=paste("x",1:25,sep="")
model.formula=as.formula(paste("y~",paste(xnames,collapse="+")))

###Box-Cox transformation
#The idea is to find the power transformation, \lambda (lambda), that maximizes
  #the likelihood
# when a specified set of explanatory variables is
  #fitted to\frac{y^{\lambda}-1}{\lambda}
# as the response. For the case \lambda=0, the Box-Cox trnsformation is
   #defined as log(y).
data=read.delim("timber.txt") # "volume" "girth"  "height"
library(MASS)
boxcox(volume~girth+height)
boxcox(volume~log(girth)+log(height))
boxcox(volume~girth+height,lambda=seq(.1,.6,.01))

#Optional arguments in model-fitting functions: subset, weights, data,
   #offset, na.action.
data=read.delim("ipomopsis.txt")
attach(data)
model=lm(Fruit~Root, subset=(Grazing=="Grazed"))
model1=lm(Fruit~Grazing,weights=Root)
model2=lm(Fruit~Grazing*Root,na.action=na.fail) #na.action=na.omit

#Akaike's Information Criterion (AIC) is known in the statistics trade as a pena
#log-likelihood. If you have a model for which log-likelihood value can be obtai
#then AIC=-2* log-likelihood+2(p+1), where p   is the number of parameters in th
#and 1 is added for the estimated variance (you can call this another parameter
#you wanted to). When comparing two models, the smaller the AIC, the better the
AIC(model1,model2)

## extracting informtion from model objects
#by name:
coef(model); fitted();resid(); effects();vcov()#variance-covariance matrix
#by list subscripts
summary(model)[[1]];... ;summary(model)[[11]]
summary(model)[[4]][[1]];... ;summary(model)[[8]]
# using $
model$coef; model$fitted; model$resid; model$df

summary.lm(model)
```

```r
summary.aov(model)[[1]][[1]]

##########regression
model1=lm(y~x) ; abline(model1)
model2=lm(y~x+I(x^2)); xv=0:90; yv=predict(model2,list(x=xv));lines(xv,yv)
anova(model1,model2) # to check whether there are singnificant improvement.

# serial correlation in the residuals
install.packages("car")
model=lm(response~explanatory)
durbin.watson(model)  # to check whether evidence of serial correlation in residuals.
data.ellipse(explanatory,response)

###the multiple regression model
oz=read.table("ozone.data.txt",header=T)  # "rad"   "temp"  "wind"  "ozone"
pairs(oz,panel=panel.smooth)
# a good way to tackle a multiple regression problem is using non-parametric smo
# in a generalized additive model like this:
library(mgcv);par(mfrow=c(2,2))
model=gam(ozone~s(rad)+s(temp)+s(wind))
plot(model);par(,frow=c(1,1))

library(tree)
model=tree(ozone~.,data=oz)
plot(model)
text(model)

model1=lm(oz$ozone~temp*wind*rad+I(rad^2)+I(temp^2)+I(wind^2))
model2=update(model1,~.-temp:wind:rad)
model3=update(model2,~.-temp:rad);...
# or
model10=step(model1)  # model simplification

################ analysis of variance
x=1:15
y=rep(c(10,12,14),each=5)
plot(x,y,ylim=c(5,16),pch=(15+(x>5)+(x>10)))  #different value different pch

# first, check the homogeneity of variances
bartlett.test(formula,data)  #parametric
kruskal.test(formula,data)  #non-parametric

## effect sizes
plot.design(y~x)

# plots for interpreeting one-wat ANOVA
# box-and-whisker plots
# barplots with error bars.
comp=read.delim("competition.txt") #"biomass"  "clipping" has 5 levels
plot(clipping,biomass)   # box-and-whisker
model=aov(biomass~clipping)
summary(model)  # mean sq of error = s^2, se=sqrt(s^2/n)=28.75, n=6
se=rep(28.75,5)
lables=as.character(levels(clipping))
ybar=as.vector(tapply(biomass,clipping,mean))

error.bars=function(yv,z,nn){
xv=barplot(yv,ylim=c(0,(max(yv)+max(z))),names=nn,ylab=deparse(substitute(yv)))
g=(max(xv)-min(xv))/50
for(i in 1:length(xv)){
```

```
lines(c(xv[i],xv[i]),c(yv[i]+z[i],yv[i]-z[i]))
lines(c(xv[i]-g,xv[i]+g),c(yv[i]+z[i],yv[i]+z[i]))
lines(c(xv[i]-g,xv[i]+g),c(yv[i]-z[i],yv[i]-z[i]))}}

error.bars(ybar,se,lables)   # 1 standard error can not seesig diff from overlap or not
lsd=qt(0.975,10)*sqrt(2*4961/6)#t*se of the difference=least significant difference(LSD)
ladbars=rep(lsd,5)/2
error.bars(ybar,ladbars,lables)
                      #LSD  bars, overlap means non-sig diff,overlap means sig diff

### factorial experiment
weight=read.delim("growth.txt")
attach(weight)
barplot(tapply(gain,list(diet,supplement),mean),beside=T,ylim=c(0,30),col=rainbow(3))
   # the second factor in the list appears as groups of bars from left to roght in
   # alphabetical order by factor level. the first factor in the list appears as
   # three levels
   # within each group of bars in alphabetical order by factor level.
labs=c("barley","oats","wheat")
legend(locator(1),labs,fill=rainbow(3))

### multiple comparisonss
model=aov(y~x); summary(model);plot(model)
TukeyHSD(model)
plot(TukeyHSD(model))
```