Basic statistics

April 10, 2017

Contents

In this Chapter we'll review R functions for generating basic descriptive and inferential statistics.

1 Descriptive statistics

1.1 Methodology

Arithmetic mean of a single sample

The mean is the sum of the numbers $\sum y$ divided by the number of numbers $n = \sum 1$ (summing over the number of numbers in the vector called y).

```
arithmetic.mean <- function(x) sum(x)/length(x)
x<-c(1:100)
arithmetic.mean(x)

## [1] 50.5

mean(x)

## [1] 50.5

salary=c(2000,2100,2200,2300,2350,2450,2500,2700,2900,2850,3500,3800,2600,
        3000,3300,3200,4000,3100,4200)
mean(salary)

## [1] 2897.368

mean(salary,trim=0.2)

## [1] 2826.923

mean(salary,trim=0.5)

## [1] 2850
```

Median of a single sample

The median (or 50th percentile) is the middle value of the sorted values of a vector of numbers.

```
sort(x)[ceiling(length(x)/2)]

## [1] 50
```

There is slight hitch here, of course, because if the vector contains an even number of numbers, then there is no middle value. The logic here is that we need to work out the arithmetic average of the two values of y on either side of the middle. The question now arises as to how we know, in general, whether the vector y contains an odd or an even number of numbers, so that we can decide which of the two methods to use.

```
med <- function(x) {
odd.even <- length(x)%%2
if (odd.even == 0) (sort(x)[length(x)/2]+sort(x)[1+ length(x)/2])/2
else
```

```
sort(x)[ceiling(length(x)/2)]
}
med(x)
```

```
## [1] 50.5
```

```
median(x)
```

```
## [1] 50.5
```

You could write the same function in a single (long) line by using ifelse instead of if.

```
med <- function(x) ifelse(length(x)%%2==1, sort(x)[ceiling(length(x)/2)],
       (sort(x)[length(x)/2]+sort(x)[1+ length(x)/2])/2 )
salary=c(2000,2100,2200,2300,2350,2450,2500,2700,2900,2850,3500,3800,2600,
         3000,3300,3200,4000,3100,4200)
mean(salary)
```

```
## [1] 2897.368
```

```
med(salary)
```

```
## [1] 2850
```

```
salary=c(2000,2100,2200,2300,2350,2450,2500,2700,2900,2850,3500,3800,2600,
         3000,3300,3200,4000,3100,15000)
mean(salary)
```

```
## [1] 3465.789
```

```
med(salary)
```

```
## [1] 2850
```

Geometric mean

For processes that change multiplicatively rather than additively, neither the arithmetic mean nor the median is an ideal measure of central tendency. Under these conditions, the appropriate measure is the geometric mean.

$$\hat{y} = \sqrt[n]{\prod y}$$

Harmonic mean

Consider the following problem. An elephant has a territory which is a square of side 2 km. Each morning, the elephant walks the boundary of this territory. He begins the day at a sedate pace, walking the first side of the territory at a speed of 1 km/hr. On the second side, he has sped up to 2 km/hr. By the third side he has accelerated to an impressive 4 km/hr, but this so wears him out, that he has to return on the final side at a sluggish 1 km/hr. So what is his average speed over the ground? You might say he travelled at 1, 2, 4 and 1 km/hr so the average speed is $(1 + 2 + 4 + 1)/4 = 8/4 = 2$ km/hr. But that is wrong. Can you see how to work out the right answer? Recall that velocity is defined as distance travelled divided by time taken. The distance travelled is easy: it is just $4 \times 2 = 8$ km. The time taken is a bit harder. The first edge was 2 km long, and travelling at 1 km/hr this

must have taken 2 hr. The second edge was 2 km long, and travelling at 2 km/hr this must have taken 1 hr. The third edge was 2 km long and travelling at 4 km/hr this must have taken 0.5 hr. The final edge was 2 km long and travelling at 1 km/hr this must have taken 2 hr. So the total time taken was $2 + 1 + 0.5 + 2 = 5.5$ hr. So the average speed is not 2 km/hr but $8/5.5 = 1.4545$ km/hr.

$$\tilde{y} = \frac{1}{(\sum(1/y))/n}$$

```
harmonic <- function (x) 1/mean(1/x)
harmonic(c(1,2,4,1))
```

```
## [1] 1.454545
```

Variance

A measure of variability is perhaps the most important quantity in statistical analysis. The greater the variability in the data, the greater will be our uncertainty in the values of parameters estimated from the data, and the less will be our ability to distinguish between competing hypotheses about the data.

The variance of a sample is measured as a function of 'the sum of the squares of the difference between the data and the arithmetic mean'. This important quantity is called the 'sum of squares':

$$SS = \sum(y - \bar{y})^2$$

To complete our calculation of the variance we need the degrees of freedom (d.f.). This important concept in statistics is defined as follows:

$$d.f. = n-k$$

which is the sample size, n, minus the number of parameters, k, estimated from the data. For the variance, we have estimated one parameter from the data, $\bar{y}$, and so there are $n - 1$ degrees of freedom. In a linear regression, we estimate two parameters from the data, the slope and the intercept, and so there are $n - 2$ degrees of freedom in a regression analysis.

We always calculate variance as

$$\text{variance} = \frac{\text{sum of squares}}{\text{degrees of freedom}}$$

```
y <- c(13,7,5,12,9,15,6,11,9,7,12)
variance <- function(x) sum((x - mean(x))^2)/(length(x)-1)
variance(y)
```

```
## [1] 10.25455
```

```
var(y)
```

```
## [1] 10.25455
```

Exercise

Write a funtion to calculate mean, variance, standard deviation, median, variable coefficient, skewness and kurtosis of data.

Variance ratio test

How do we know if two variances are significantly different from one another? Write a funtion to calculate statistic of Variance ratio test.

1.2 Basic method

First, we'll look at measures of central tendency, variability, and distribution shape for continuous variables. For illustrative purposes, we'll use several of the variables from the Motor Trend Car Road Tests (mtcars) dataset you first saw in chapter 1. Our focus will be on miles per gallon (mpg), horsepower (hp), and weight (wt). install.packages("pastecs"), install.packages("Hmisc")

```r
vars <- c("mpg", "hp", "wt")
head(mtcars[vars])

##                    mpg  hp    wt
## Mazda RX4         21.0 110 2.620
## Mazda RX4 Wag     21.0 110 2.875
## Datsun 710        22.8  93 2.320
## Hornet 4 Drive    21.4 110 3.215
## Hornet Sportabout 18.7 175 3.440
## Valiant           18.1 105 3.460

summary(mtcars[vars])

##      mpg              hp              wt
## Min.   :10.40   Min.   : 52.0   Min.   :1.513
## 1st Qu.:15.43   1st Qu.: 96.5   1st Qu.:2.581
## Median :19.20   Median :123.0   Median :3.325
## Mean   :20.09   Mean   :146.7   Mean   :3.217
## 3rd Qu.:22.80   3rd Qu.:180.0   3rd Qu.:3.610
## Max.   :33.90   Max.   :335.0   Max.   :5.424

library(Hmisc)

## Warning:  package 'ggplot2' was built under R version 3.3.3

describe(mtcars[vars])

## mtcars[vars]
##
##  3  Variables      32  Observations
## ---------------------------------------------------------------------------
## mpg
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##       32        0       25    0.999    20.09    6.796    12.00    14.34
##      .25      .50      .75      .90      .95
##    15.43    19.20    22.80    30.09    31.30
##
## lowest : 10.4 13.3 14.3 14.7 15.0, highest: 26.0 27.3 30.4 32.4 33.9
## ---------------------------------------------------------------------------
## hp
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##       32        0       22    0.997    146.7    77.04    63.65    66.00
```

```
##      .25      .50      .75      .90      .95
##    96.50   123.00   180.00   243.50   253.55
##
## lowest :  52  62  65  66  91, highest: 215 230 245 264 335
## ------------------------------------------------------------------------
## wt
##          n  missing distinct     Info     Mean      Gmd      .05      .10
##         32        0       29    0.999    3.217    1.089    1.736    1.956
##      .25      .50      .75      .90      .95
##    2.581    3.325    3.610    4.048    5.293
##
## lowest : 1.513 1.615 1.835 1.935 2.140, highest: 3.845 4.070 5.250 5.345 5.424
## ------------------------------------------------------------------------
```

```r
library(pastecs)
stat.desc(mtcars[vars])
```

```
##                         mpg           hp           wt
## nbr.val         32.0000000   32.0000000   32.0000000
## nbr.null         0.0000000    0.0000000    0.0000000
## nbr.na           0.0000000    0.0000000    0.0000000
## min             10.4000000   52.0000000    1.5130000
## max             33.9000000  335.0000000    5.4240000
## range           23.5000000  283.0000000    3.9110000
## sum            642.9000000 4694.0000000  102.9520000
## median          19.2000000  123.0000000    3.3250000
## mean            20.0906250  146.6875000    3.2172500
## SE.mean          1.0654240   12.1203173    0.1729685
## CI.mean.0.95     2.1729465   24.7195501    0.3527715
## var             36.3241028 4700.8669355    0.9573790
## std.dev          6.0269481   68.5628685    0.9784574
## coef.var         0.2999881    0.4674077    0.3041285
```

1.3   Descriptive statistics by group

When comparing groups of individuals or observations, the focus is usually on the descriptive statistics of each group, rather than the total sample.

```r
aggregate(mtcars[vars], by = list(am = mtcars$am), mean)
```

```
##   am      mpg       hp       wt
## 1  0 17.14737 160.2632 3.768895
## 2  1 24.39231 126.8462 2.411000
```

```r
aggregate(mtcars[vars], by = list(am = mtcars$am), sd)
```

```
##   am      mpg       hp        wt
## 1  0 3.833966 53.90820 0.7774001
## 2  1 6.166504 84.06232 0.6169816
```

Unfortunately, aggregate() only allows you to use single value functions such as mean, standard deviation, and the like in each call. It won't return several statistics at once. install.packages("psych")

```
library(psych)
describeBy(mtcars[vars], mtcars$am)

## $`0`
##      vars  n   mean     sd median trimmed   mad   min    max  range  skew
## mpg     1 19  17.15   3.83  17.30   17.12  3.11 10.40  24.40  14.00  0.01
## hp      2 19 160.26  53.91 175.00  161.06 77.10 62.00 245.00 183.00 -0.01
## wt      3 19   3.77   0.78   3.52    3.75  0.45  2.46   5.42   2.96  0.98
##      kurtosis    se
## mpg     -0.80  0.88
## hp      -1.21 12.37
## wt       0.14  0.18
##
## $`1`
##      vars  n   mean     sd median trimmed   mad   min    max  range skew
## mpg     1 13  24.39   6.17  22.80   24.38  6.67 15.00  33.90  18.90 0.05
## hp      2 13 126.85  84.06 109.00  114.73 63.75 52.00 335.00 283.00 1.36
## wt      3 13   2.41   0.62   2.32    2.39  0.68  1.51   3.57   2.06 0.21
##      kurtosis    se
## mpg     -1.46  1.71
## hp       0.56 23.31
## wt      -1.17  0.17
##
## attr(,"call")
## by.data.frame(data = x, INDICES = group, FUN = describe, type = type)
```

1.4   An advanced example

The describe.by() function doesn't allow you to specify an arbitrary function, so you can use the following command.

```
mystats <- function(x, na.omit = FALSE) {
if (na.omit)  x <- x[!is.na(x)]
m <- mean(x)
n <- length(x)
s <- sd(x)
skew <- sum((x - m)^3/s^3)/n
kurt <- sum((x - m)^4/s^4)/n - 3
return(c(n = n, mean = m, stdev = s, skew = skew, kurtosis = kurt))
}
library(reshape)
dfm <- melt(mtcars, measure.vars = c("mpg", "hp", "wt"), id.vars = c("am", "cyl"))
cast(dfm, am + cyl + variable ~ ., mystats)

##   am cyl variable n       mean       stdev         skew  kurtosis
## 1  0   4      mpg 3  22.900000   1.4525839  6.851657e-02 -2.3333333
## 2  0   4       hp 3  84.666667  19.6553640 -3.804217e-01 -2.3333333
## 3  0   4       wt 3   2.935000   0.4075230 -3.807326e-01 -2.3333333
## 4  0   6      mpg 4  19.125000   1.6317169  4.817573e-01 -1.9076891
## 5  0   6       hp 4 115.250000   9.1787799 -9.395461e-02 -2.3303515
## 6  0   6       wt 4   3.388750   0.1162164 -7.349455e-01 -1.6967507
```

```
## 7    0   8      mpg 12   15.050000  2.7743959 -2.843325e-01 -0.9635443
## 8    0   8       hp 12 194.166667 33.3598379  2.785849e-01 -1.4385375
## 9    0   8       wt 12    4.104083  0.7683069  8.542070e-01 -1.1433587
## 10   1   4      mpg  8   28.075000  4.4838599 -2.078592e-01 -1.6584985
## 11   1   4       hp  8   81.875000 22.6554156  1.373411e-01 -1.8105587
## 12   1   4       wt  8    2.042250  0.4093485  3.488929e-01 -1.1451272
## 13   1   6      mpg  3   20.566667  0.7505553 -3.849002e-01 -2.3333333
## 14   1   6       hp  3 131.666667 37.5277675  3.849002e-01 -2.3333333
## 15   1   6       wt  3    2.755000  0.1281601 -1.154378e-01 -2.3333333
## 16   1   8      mpg  2   15.400000  0.5656854  0.000000e+00 -2.7500000
## 17   1   8       hp  2 299.500000 50.2045815  0.000000e+00 -2.7500000
## 18   1   8       wt  2    3.370000  0.2828427 -1.193490e-15 -2.7500000
```

or install.packages("doBy")

```r
#summaryBy(formula, data=dataframe, FUN=function)
library(doBy)
summaryBy(mpg+hp+wt~am, data=mtcars, FUN=mystats)
```

```
##   am mpg.n mpg.mean mpg.stdev   mpg.skew mpg.kurtosis hp.n  hp.mean
## 1  0    19 17.14737  3.833966 0.01395038   -0.8031783   19 160.2632
## 2  1    13 24.39231  6.166504 0.05256118   -1.4553520   13 126.8462
##   hp.stdev     hp.skew hp.kurtosis wt.n  wt.mean  wt.stdev   wt.skew
## 1 53.90820 -0.01422519  -1.2096973   19 3.768895 0.7774001 0.9759294
## 2 84.06232  1.35988586   0.5634635   13 2.411000 0.6169816 0.2103128
##   wt.kurtosis
## 1   0.1415676
## 2  -1.1737358
```

1.5   Describe the data using a Graph

```r
COM <- function (x)   {
    par(mfrow=c(2,2))                   # Combine four graph
    hist(x)                             # histogram
    dotchart(x)                         # dotchart
    boxplot(x,horizontal=T)             # boxplot
    qqnorm(x);qqline(x)                 # normal
    par(mfrow=c(1,1))                   # reset the enviroment
  }
x <- rnorm(100)
COM(x)
```

Histogram of x

Normal Q–Q Plot
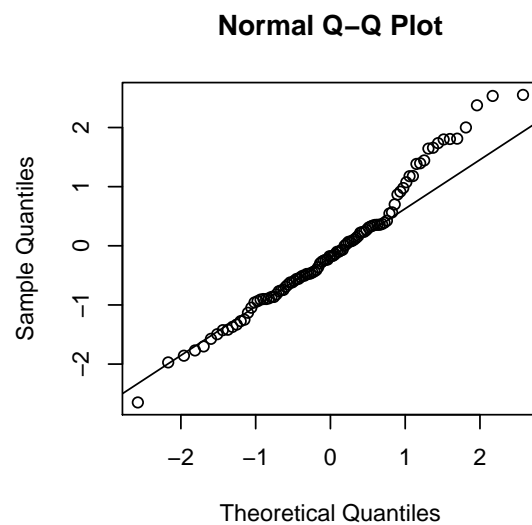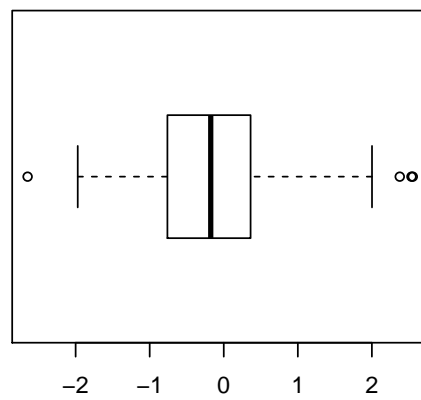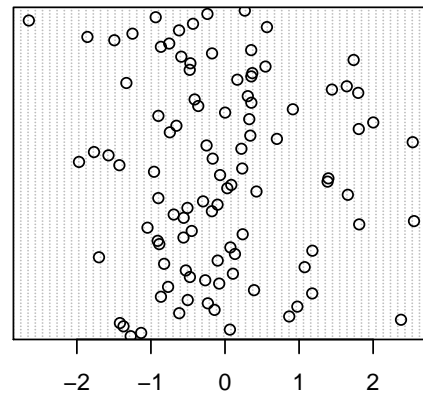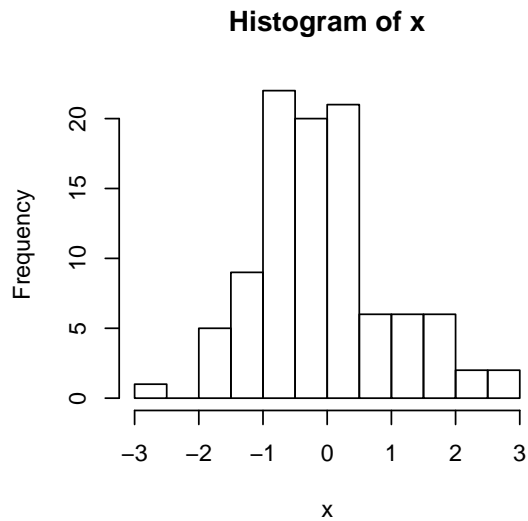
1.6   Frequency tables

One way tables

You can generate simple frequency counts using the table() function.

```
library("vcd")
```

```
## Loading required package:  grid
```

```
mytable <- with(Arthritis, table(Improved))
mytable
```

```
## Improved
##   None   Some Marked
##     42     14     28
```

```
prop.table(mytable)

## Improved
##      None      Some    Marked
## 0.5000000 0.1666667 0.3333333

prop.table(mytable)*100

## Improved
##     None     Some   Marked
## 50.00000 16.66667 33.33333
```

Two way tables

For two-way tables, the format for the table() function is mytable <- table(A, B), where A is the row variable, and B is the column variable.

```
mytable <- with(Arthritis, table(Treatment,Improved))
mytable

##           Improved
## Treatment None Some Marked
##    Placebo   29    7      7
##    Treated   13    7     21

mytable <- xtabs(~ Treatment+Improved, data=Arthritis)
mytable

##           Improved
## Treatment None Some Marked
##    Placebo   29    7      7
##    Treated   13    7     21

DF <- as.data.frame(UCBAdmissions)
head(DF)

##       Admit Gender Dept Freq
## 1 Admitted    Male    A  512
## 2 Rejected    Male    A  313
## 3 Admitted  Female    A   89
## 4 Rejected  Female    A   19
## 5 Admitted    Male    B  353
## 6 Rejected    Male    B  207

xtabs(Freq ~ Gender + Admit, DF)

##          Admit
## Gender    Admitted Rejected
##    Male       1198     1493
##    Female      557     1278
```

You can generate marginal frequencies using the addmargins() functions.

```
addmargins(mytable)

##          Improved
## Treatment None Some Marked Sum
##   Placebo   29    7      7  43
##   Treated   13    7     21  41
##   Sum       42   14     28  84

addmargins(prop.table(mytable))

##          Improved
## Treatment       None       Some     Marked       Sum
##   Placebo 0.34523810 0.08333333 0.08333333 0.51190476
##   Treated 0.15476190 0.08333333 0.25000000 0.48809524
##   Sum     0.50000000 0.16666667 0.33333333 1.00000000
```

NOTE: The table() function ignores missing values (NAs) by default. To include NA as a valid category in the frequency counts, include the table option useNA="ifany".

1.7   Correlations

Correlation coefficients are used to describe relationships among quantitative variables. The sign ± indicates the direction of the relationship (positive or inverse) and the magnitude indicates the strength of the relationship (ranging from 0 for no relationship to 1 for a perfectly predictable relationship).

R can produce a variety of correlation coefficients, including Pearson, Spearman, Kendall, partial, polychoric, and polyserial. The cor() function produces all three correlation coefficients, whereas the cov() function provides covariances.

```
#cor(x, use= , method= )
cor(1:10, 2:11)

## [1] 1

states<- state.x77[,1:5]
cov(states)

##                 Population        Income   Illiteracy     Life Exp      Murder
## Population 19931683.7588 571229.7796  292.8679592 -407.8424612 5663.523714
## Income        571229.7796 377573.3061 -163.7020408  280.6631837 -521.894286
## Illiteracy       292.8680   -163.7020    0.3715306   -0.4815122    1.581776
## Life Exp        -407.8425    280.6632   -0.4815122    1.8020204   -3.869480
## Murder          5663.5237   -521.8943    1.5817755   -3.8694804   13.627465

cor(states)

##             Population     Income Illiteracy    Life Exp      Murder
## Population  1.00000000  0.2082276  0.1076224 -0.06805195  0.3436428
## Income      0.20822756  1.0000000 -0.4370752  0.34025534 -0.2300776
## Illiteracy  0.10762237 -0.4370752  1.0000000 -0.58847793  0.7029752
## Life Exp   -0.06805195  0.3402553 -0.5884779  1.00000000 -0.7808458
## Murder      0.34364275 -0.2300776  0.7029752 -0.78084575  1.0000000

cor(states, method="spearman")
```

```
##              Population     Income Illiteracy   Life Exp     Murder
## Population   1.0000000  0.1246098  0.3130496 -0.1040171  0.3457401
## Income       0.1246098  1.0000000 -0.3145948  0.3241050 -0.2174623
## Illiteracy   0.3130496 -0.3145948  1.0000000 -0.5553735  0.6723592
## Life Exp    -0.1040171  0.3241050 -0.5553735  1.0000000 -0.7802406
## Murder       0.3457401 -0.2174623  0.6723592 -0.7802406  1.0000000
```

The default options are use="everything" and method="pearson". Notice that you get square matrices by default (all variables crossed with all other variables). You can also produce nonsquare matrices.

```
x <- states[,c("Population", "Income", "Illiteracy")]
y <- states[,c("Life Exp", "Murder")]
cor(x,y)

##               Life Exp     Murder
## Population -0.06805195  0.3436428
## Income      0.34025534 -0.2300776
## Illiteracy -0.58847793  0.7029752
```

Testing correlations for significance

The corr.test() function provided in the psych package allows you to testing correlations for significance. install.packages("psych")

```
library(psych)
corr.test(states, use="complete")

## Call:corr.test(x = states, use = "complete")
## Correlation matrix
##            Population Income Illiteracy Life Exp Murder
## Population       1.00   0.21       0.11    -0.07   0.34
## Income           0.21   1.00      -0.44     0.34  -0.23
## Illiteracy       0.11  -0.44       1.00    -0.59   0.70
## Life Exp        -0.07   0.34      -0.59     1.00  -0.78
## Murder           0.34  -0.23       0.70    -0.78   1.00
## Sample Size
## [1] 50
## Probability values (Entries above the diagonal are adjusted for multiple tests.)
##            Population Income Illiteracy Life Exp Murder
## Population       0.00   0.44       0.91     0.91   0.09
## Income           0.15   0.00       0.01     0.09   0.43
## Illiteracy       0.46   0.00       0.00     0.00   0.00
## Life Exp         0.64   0.02       0.00     0.00   0.00
## Murder           0.01   0.11       0.00     0.00   0.00
##
##  To see confidence intervals of the correlations, print with the short=FALSE option
```

The use= options can be "pairwise" or "complete" (for pairwise or listwise deletion of missing values, respectively). The method= option is "pearson" (the default), "spearman", or "kendall".

Partial correlations

A partial correlation is a correlation between two quantitative variables, controlling for one or more other quantitative variables. You can use the pcor() function in the ggm package to provide partial correlation coefficients. install.packages("ggm")

```
#pcor(u, S)
library(ggm)

## Loading required package:  igraph
##
## Attaching package:  'igraph'
## The following objects are masked from 'package:stats':
##
##    decompose, spectrum
## The following object is masked from 'package:base':
##
##    union
##
## Attaching package:  'ggm'
## The following object is masked from 'package:igraph':
##
##    pa
## The following object is masked from 'package:Hmisc':
##
##    rcorr

pcor(c(1,5,2,3,4), cov(states))

## [1] 0.4476116
```

where u is a vector of numbers, with the first two numbers the indices of the variables to be correlated, and the remaining numbers the indices of the conditioning variables (that is, the variables being partialed out). S is the covariance matrix among the variables.

Another example

install.packages("MSG")

```
library("MSG")
par(mfrow = c(1, 2), pch = 20, ann = FALSE, mar = c(2, 2, 0.5, 0.2))
plot(BinormCircle, col = rgb(1, 0, 0))
plot(BinormCircle, col = rgb(1, 0, 0, alpha = 0.01))
attach(BinormCircle)
cor(V1,V2)
detach(BinormCircle)
```

1.8   OUTPUT IN MULTI-FORMAT

As anything with R, there are many ways of exporting output into nice tables. Stargazer will automatically recognize the type of object, and will produce the appropriate output. In the case of data frames, it will display summary statistics .

```r
#install.packages("stargazer")
library(stargazer)
mydata <- mtcars
stargazer(mydata, type = "text", title="Descriptive statistics", digits=1, out="table1.txt")
#Use this option if you want the variables in columns
stargazer(mydata, type = "text", title="Descriptive statistics",
          digits=1, out="table1.txt", flip=TRUE)
```

The table will be saved in the working directory with whatever name you write in the out option. You can open this file with any word processor. Use "flip" option if you want the variables in columns.

Use the option covariate.labels to replace variable names with variable labels. Must be in same order as in the dataset.

```r
stargazer(mydata, type = "text", title="Descriptive statistics", digits=1, out="table1.txt",
          covariate.labels=c("Miles/(US)gallon","No. of cylinders","Displacement (cu.in.)",
                             "Gross horsepower","Rear axle ratio","Weight (lb/1000)",
                             "1/4 mile time","V/S","Transmission (0=auto, 1=manual)",
                             "Number of forward gears","Number of carburetors"))
#selected variables
stargazer(mydata[c("mpg","hp","drat")], type = "text",
          title="Descriptive statistics/selected variables", digits=1, out="table2.txt")
#by group stargazer(subset(mydata[c("mpg","hp","drat")], mydata$am==0),
          title="Automatic transmission", type = "text", digits=1, out="table3.txt")
```

The default setting produces LATEX code, the additional alternatives are:

- Output as text, which allows a quick view of results

- Output as html, which produce editable tables for Word documents.

```r
#Latex stargazer(mydata, title="Descriptive statistics", digits=1, out="table1.tex",
          covariate.labels=c("Miles/(US)gallon","No. of cylinders","Displacement (cu.in.)",
                             "Gross horsepower","Rear axle ratio","Weight (lb/1000)",
                             "1/4 mile time","V/S","Transmission (0=auto, 1=manual)",
                             "Number of forward gears","Number of carburetors"))
#HTML stargazer(mydata, type="html", title="Descriptive statistics", digits=1, out="table1.html",
          covariate.labels=c("Miles/(US)gallon","No. of cylinders","Displacement (cu.in.)",
                             "Gross horsepower","Rear axle ratio","Weight (lb/1000)",
                             "1/4 mile time","V/S","Transmission (0=auto, 1=manual)",
                             "Number of forward gears","Number of carburetors"))
```
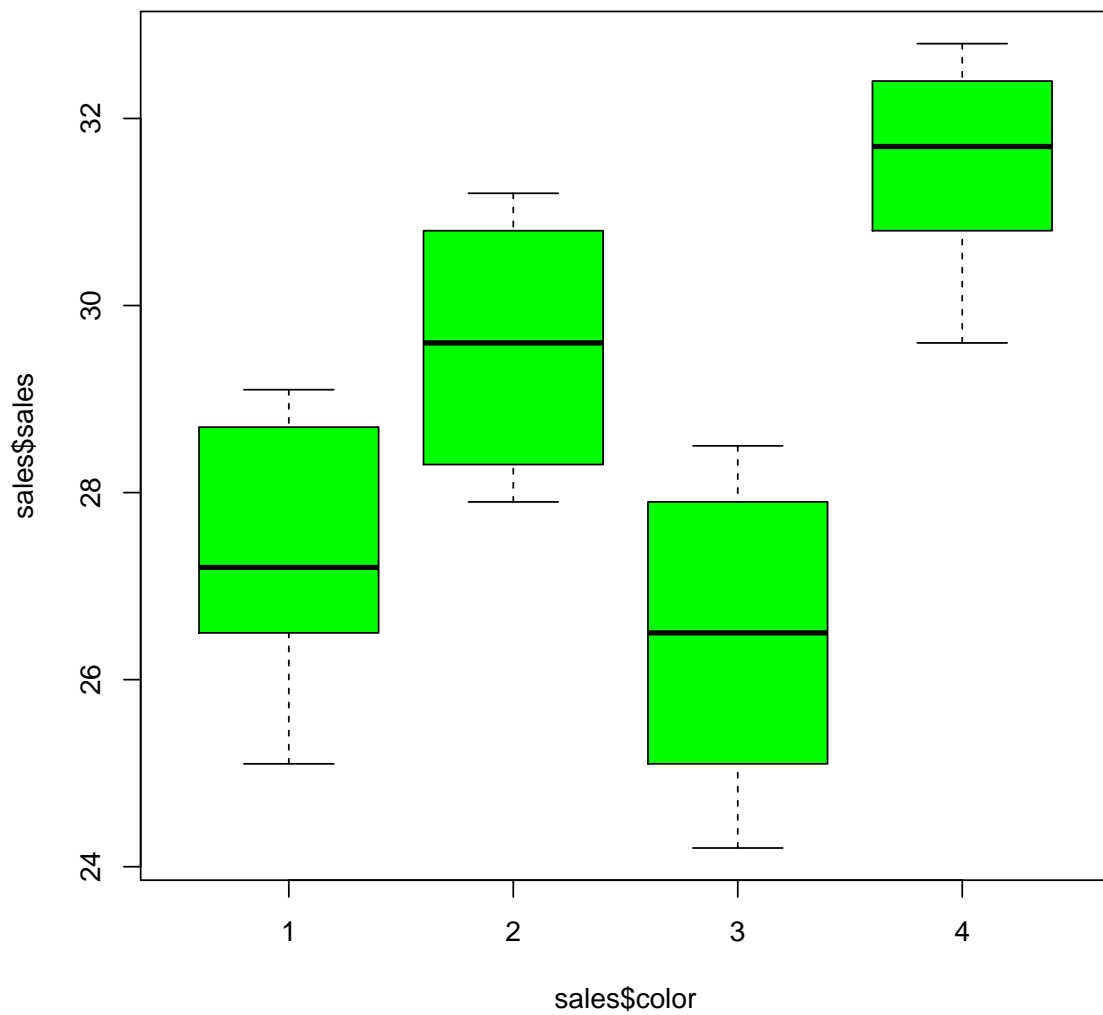
2 Analysis of variance (ANOVA)

2.1 One-way ANOVA

In a one-way ANOVA, you're interested in comparing the dependent variable means of two or more groups defined by a categorical grouping factor.

```r
library("foreign")
sales<-read.dta("sales.dta")
class(sales$color)
```

```
## [1] "integer"
```

```
sales$color<-factor(sales$color)
plot(sales$sales ~ sales$color,col="green")
```



```
aggregate(sales$sales, by=list(sales$color), FUN=mean)
```

```
##   Group.1     x
## 1       1 27.32
## 2       2 29.56
## 3       3 26.44
## 4       4 31.46
```
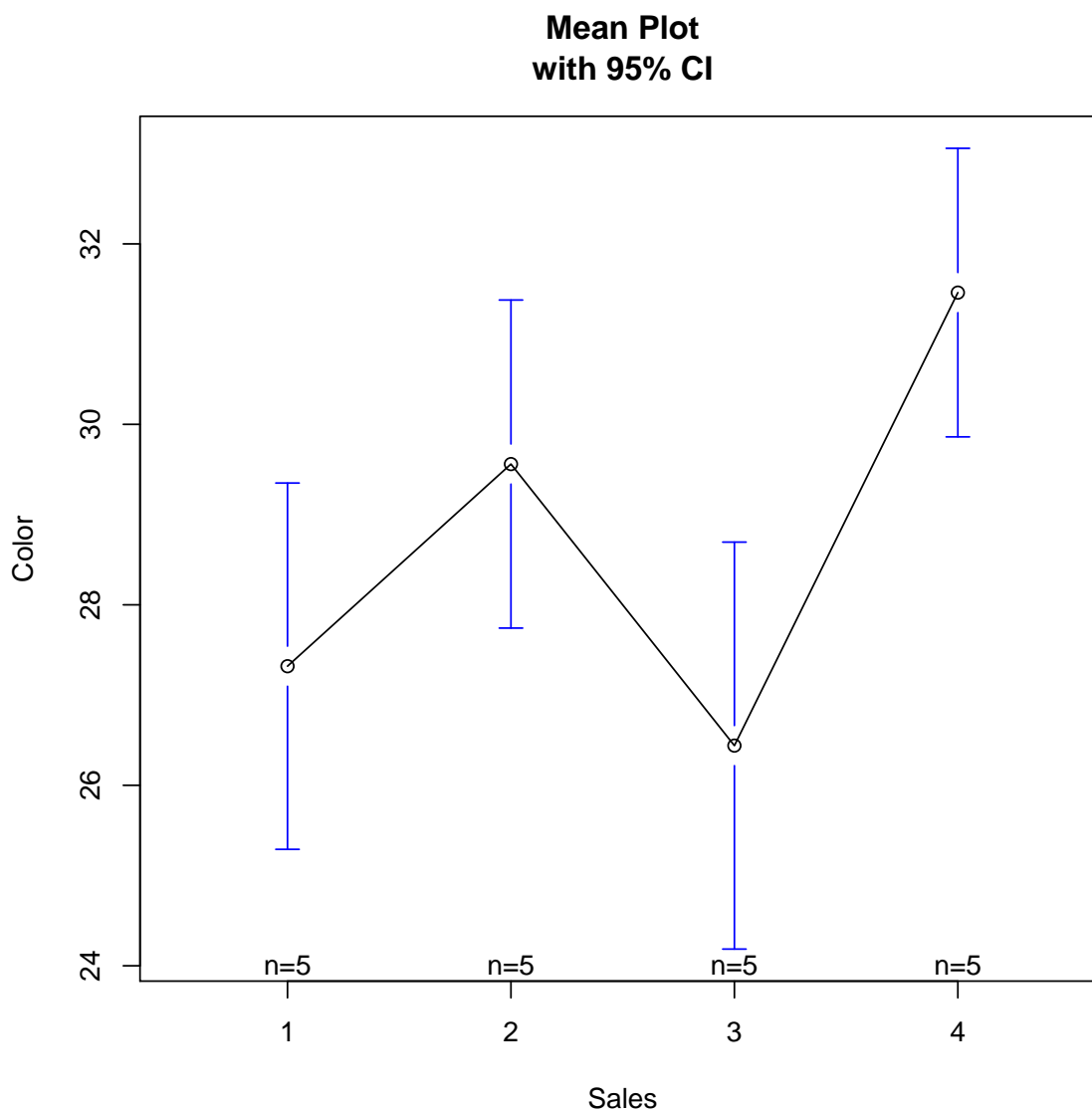
```
fit<-aov(sales~color,data=sales)
summary(fit)
```

```
##              Df Sum Sq Mean Sq F value   Pr(>F)
## color        3  76.85  25.615   10.49 0.000466 ***
## Residuals   16  39.08   2.443
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

library(gplots)

##
## Attaching package:  'gplots'
## The following object is masked from 'package:stats':
##
##     lowess

plotmeans(sales$sales ~ sales$color, xlab="Sales", ylab="Color",
          main="Mean Plot\nwith 95% CI")
```

Mean Plot
with 95% CI

```r
bartlett.test(sales$sales ~ sales$color)

##
##  Bartlett test of homogeneity of variances
##
## data:  sales$sales by sales$color
## Bartlett's K-squared = 0.46564, df = 3, p-value = 0.9264
```

Bartlett's test indicates that the variances in the five groups don't differ significantly (p = 0.93).

2.2   Two-way factorial ANOVA

In a two-way factorial ANOVA, subjects are assigned to groups that are formed from the cross-classification of two factors. This example uses the ToothGrowth dataset in the base installation to

demonstrate a two-way between-groups ANOVA. Sixty guinea pigs are randomly assigned to receive one of three levels of ascorbic acid (0.5, 1, or 2mg), and one of two delivery methods (orange juice or Vitamin C), under the restriction that each treatment combination has 10 guinea pigs. The dependent variable is tooth length.

```
attach(ToothGrowth)
table(supp, dose)

##      dose
## supp 0.5  1   2
##   OJ  10 10 10
##   VC  10 10 10

aggregate(len, by = list(supp, dose), FUN = mean)

##   Group.1 Group.2     x
## 1      OJ     0.5 13.23
## 2      VC     0.5  7.98
## 3      OJ     1.0 22.70
## 4      VC     1.0 16.77
## 5      OJ     2.0 26.06
## 6      VC     2.0 26.14

aggregate(len, by = list(supp, dose), FUN = sd)

##   Group.1 Group.2        x
## 1      OJ     0.5 4.459709
## 2      VC     0.5 2.746634
## 3      OJ     1.0 3.910953
## 4      VC     1.0 2.515309
## 5      OJ     2.0 2.655058
## 6      VC     2.0 4.797731

fit <- aov(len ~ supp * dose)
summary(fit)

##              Df Sum Sq Mean Sq F value   Pr(>F)
## supp          1  205.4   205.4  12.317 0.000894 ***
## dose          1 2224.3  2224.3 133.415  < 2e-16 ***
## supp:dose     1   88.9    88.9   5.333 0.024631 *
## Residuals    56  933.6    16.7
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The table statement indicates that you have a balanced design (equal sample sizes in each cell of the design), and the aggregate statements provide the cell means and standard deviations. The ANOVA table provided by the summary() function indicates that both main effects (supp and dose) and the interaction between these factors are significant. You can visualize the results in several ways.
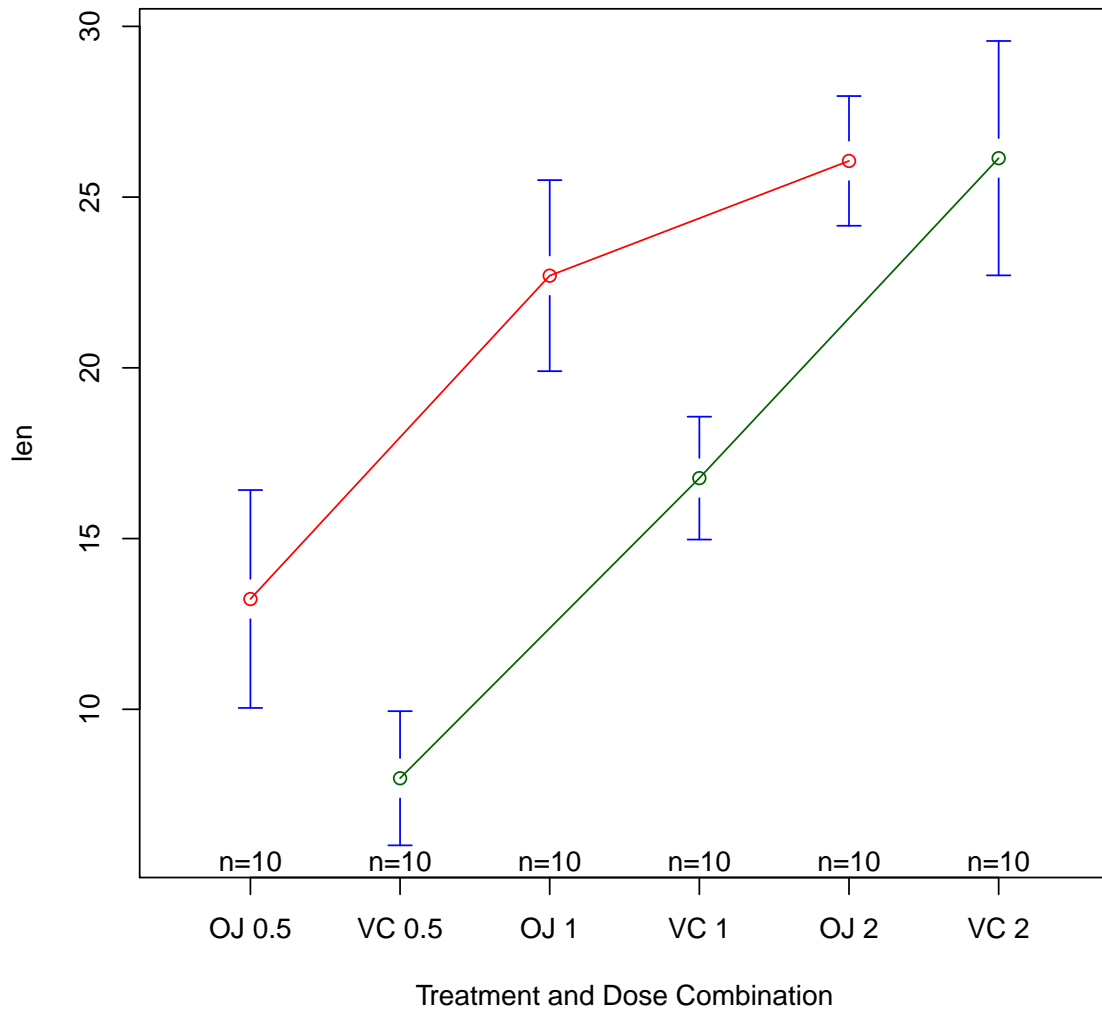
```
interaction.plot(dose, supp, len, type = "b", col = c("red","blue"), pch = c(16, 18),
                 main = "Interaction between Dose and Supplement Type")
```

Interaction between Dose and Supplement Type
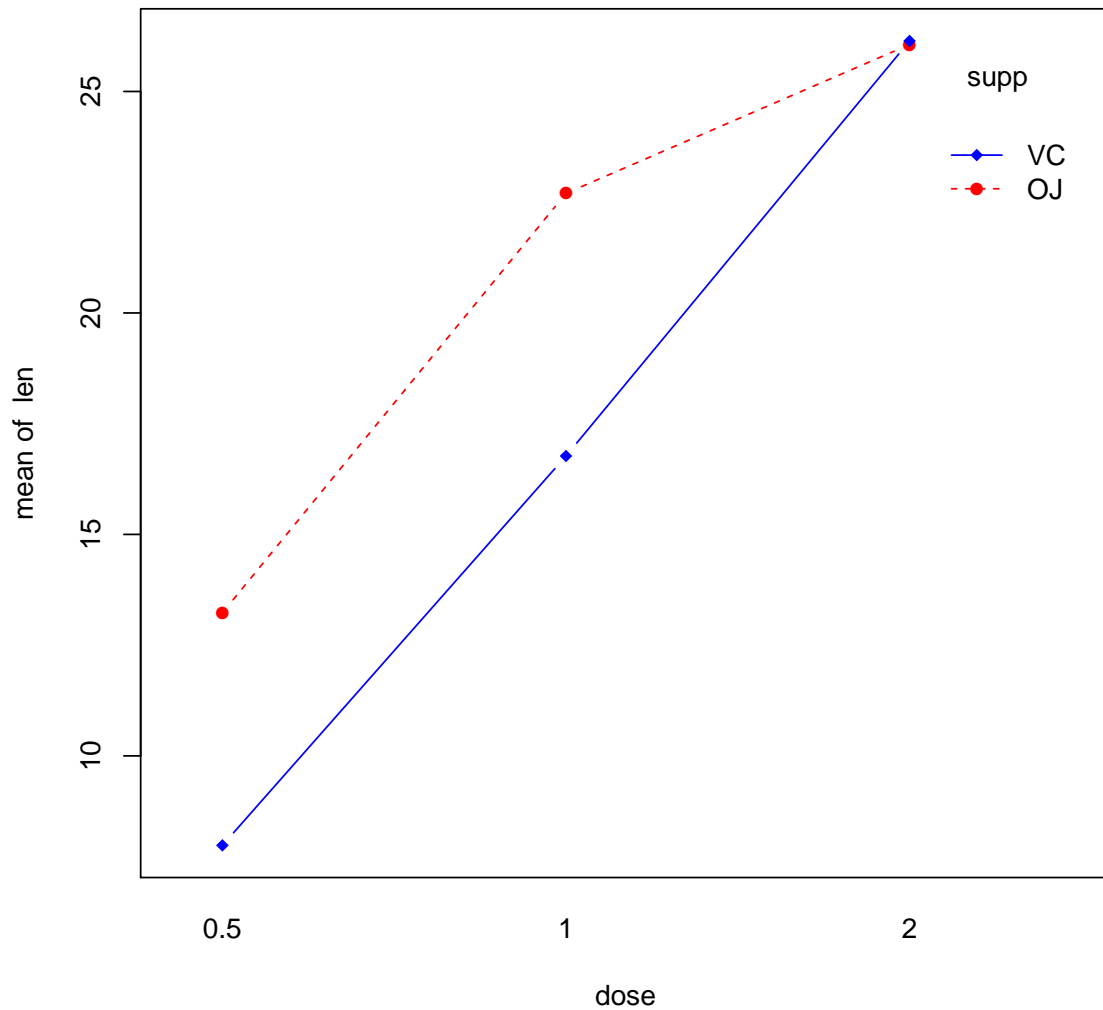
```r
library(gplots)
plotmeans(len ~ interaction(supp, dose, sep = " "),
          connect = list(c(1, 3, 5), c(2, 4, 6)),
          col = c("red", "darkgreen"),
          main = "Interaction Plot with 95% CIs",
          xlab = "Treatment and Dose Combination")
```

Interaction Plot with 95% CIs


```
library(HH)

## Warning:  package 'HH' was built under R version 3.3.3
## Loading required package:  latticeExtra
## Loading required package:  RColorBrewer
##
## Attaching package:  'latticeExtra'
## The following object is masked from 'package:vcd':
##
##     rootogram
## The following object is masked from 'package:ggplot2':
##
##     layer
## Loading required package:  multcomp
```

```
## Loading required package:  mvtnorm
## Loading required package:  TH.data
## Loading required package:  MASS
## Warning:  package 'MASS' was built under R version 3.3.3
##
## Attaching package:  'TH.data'
## The following object is masked from 'package:MASS':
##
##    geyser
## Loading required package:  gridExtra
##
## Attaching package:  'gridExtra'
## The following object is masked from 'package:Hmisc':
##
##    combine
##
## Attaching package:  'HH'
## The following object is masked from 'package:gplots':
##
##    residplot
## The following object is masked from 'package:vcd':
##
##    odds
## The following object is masked from 'package:psych':
##
##    logit
## The following object is masked from 'package:boot':
##
##    logit

interaction2wt(len ~ supp * dose)
```

len: main effects and 2–way interactions
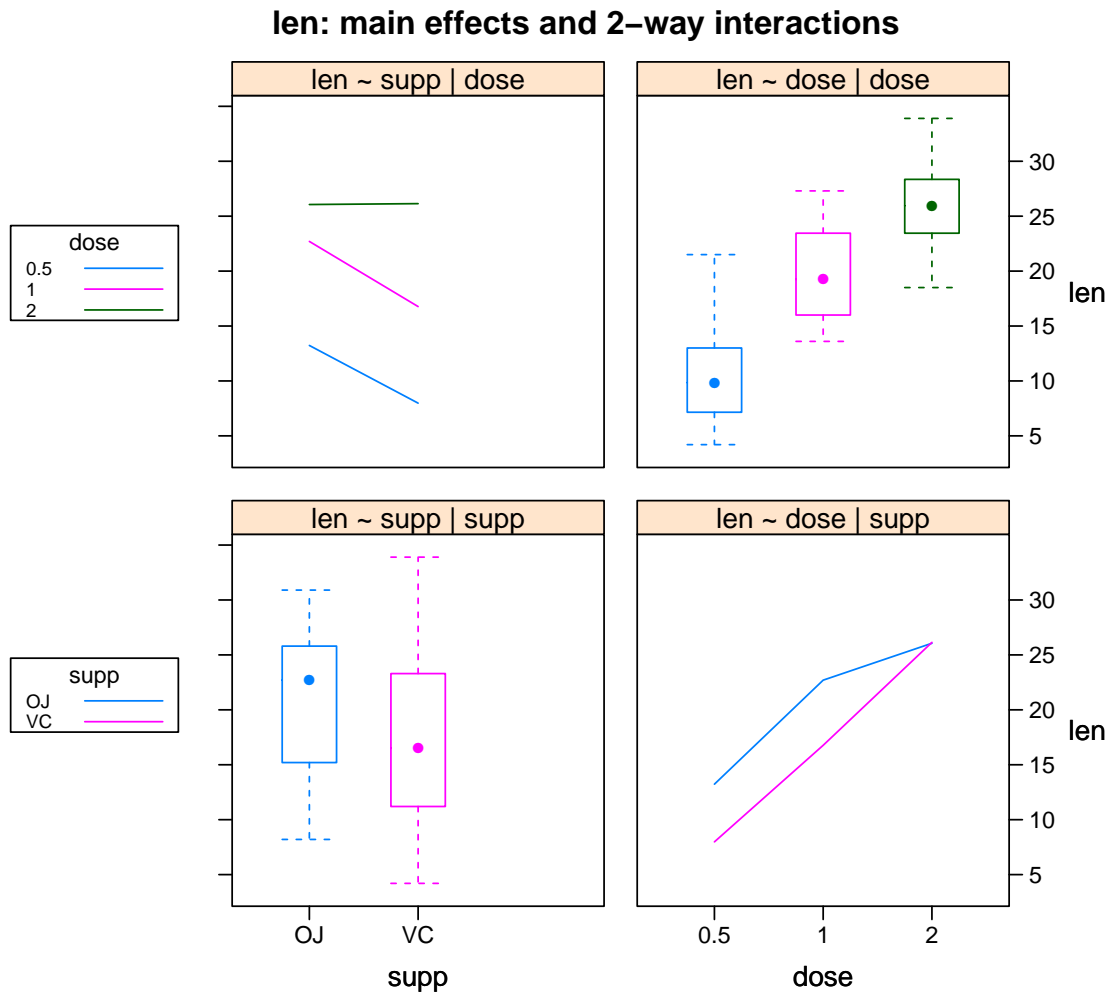
All three graphs indicate that tooth growth increases with the dose of ascorbic acid for both orange juice and Vitamin C. For the 0.5 and 1mg doses, orange juice produced more tooth growth than Vitamin C. For 2mg of ascorbic acid, both delivery methods produced identical growth.

3   Probability functions

Random Sampling

The sample function draws randomly from a specified set of (scalar) objects allowing you to sample from arbitrary distributions.

```
set.seed(1)
sample(1:10, 4)

## [1] 3 4 5 7
```

```
sample(1:10, 4)

## [1] 3 9 8 5

sample(letters, 5)

## [1] "q" "b" "e" "x" "p"

sample(1:10) ## permutation

##  [1]  4  7 10  6  9  2  8  3  1  5

sample(1:10)

##  [1]  2  3  4  1  9  5 10  8  6  7

sample(1:10, replace = TRUE) ## Sample w/replacement

##  [1] 2 9 7 8 2 8 5 9 7 8
```

In R, probability functions take the form [dpqr]distribution_abbreviation(). where the first letter refers to the aspect of the distribution returned:

d = density

**p** = distribution function

q = quantile function

r = random generation (random deviates)

To see how these work, let's look at functions related to the normal distribution. If you don't specify a mean and a standard deviation, the standard normal distribution is assumed (mean=0, sd=1).

```
par(mfrow=c(1,2))
x <- pretty(c(-3,3), 60)
y <- dnorm(x)
z <- pnorm(x)
plot(x, y, type = "l", xlab = "Normal Deviate", ylab = "Density", yaxs = "i" )
plot(x, z, type = "l", xlab = "Normal Deviate", ylab = "Distribution", yaxs = "i" )
```
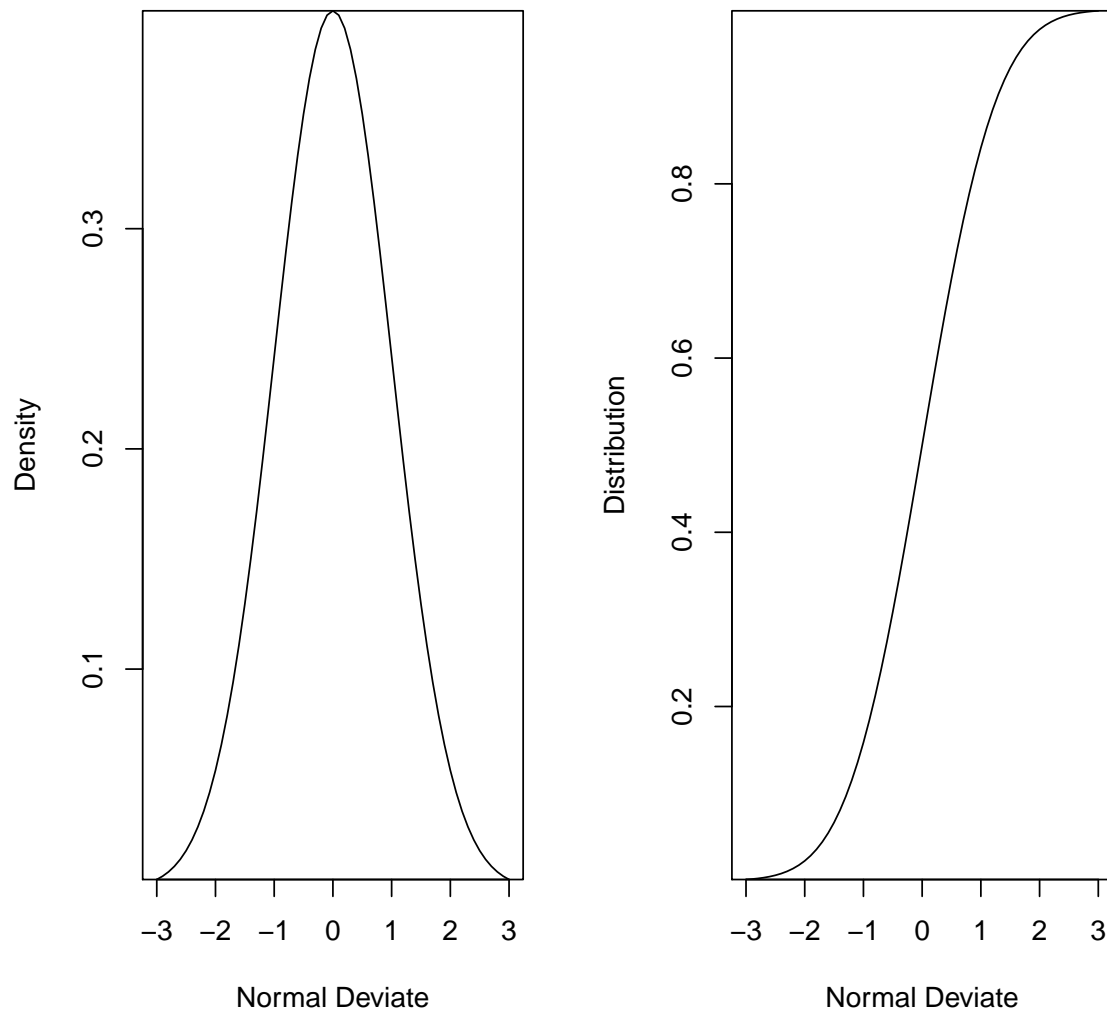
What is the area under the standard normal curve to the left of z=1.96?

```
pnorm(1.96)
```

```
## [1] 0.9750021
```

What is the value of the 90th percentile of a normal distribution with a mean of 500 and a standard deviation of 100?

```
qnorm(.95)
```

```
## [1] 1.644854
```

```
qnorm(.9, mean=500, sd=100)
```

```
## [1] 628.1552
```

Generate 50 random normal deviates with a mean of 50 and a standard deviation of 10.

```
rnorm(50, mean=50, sd=10)
```

```
##  [1] 51.33336 58.04190 49.42893 55.03608 60.85769 43.09046 37.15401
##  [8] 50.46726 47.64293 44.57112 45.66690 43.50528 57.26751 61.51912
## [15] 59.92160 45.70487 62.38304 47.20654 67.57903 55.60746 45.47216
## [22] 41.67957 38.33429 39.34409 34.36218 61.56537 58.32047 47.72671
## [29] 52.66137 46.23297 74.41365 42.04661 49.45123 52.50141 56.18243
## [36] 48.27376 27.76100 37.36386 53.58729 49.88955 40.59351 48.84175
## [43] 41.85031 52.42263 35.74902 53.65941 52.48413 50.65288 50.19156
## [50] 52.57338
```

Setting the seed for random number generation

Each time you generate pseudo-random deviates, a different seed, and therefore different results, are produced. To make your results reproducible, you can specify the seed explicitly, using the set.seed() function . An example is given in the next listing. Here, the runif() function is used to generate pseudo-random numbers from a uniform distribution on the interval 0 to 1.

```
runif(5)
```

```
## [1] 0.2581659 0.7293096 0.4525708 0.1751268 0.7466983
```

```
runif(5)
```

```
## [1] 0.1049876 0.8645449 0.6146450 0.5571595 0.3287773
```

```
set.seed(1234)
runif(5)
```

```
## [1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154
```

```
set.seed(1234)
runif(5)
```

```
## [1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154
```

Generating multivariate normal data

```
library(MASS)
options(digits=3)
set.seed(1234)
mean <- c(230.7, 146.7, 3.6)
sigma <- matrix( c(15360.8, 6721.2, -47.1, 6721.2, 4700.9, -16.5,
        -47.1,  -16.5,   0.3), nrow=3, ncol=3)
mydata <- mvrnorm(500, mean, sigma)
mydata <- as.data.frame(mydata)
names(mydata) <- c("y", "x1", "x2")
dim(mydata)
```

```
## [1] 500    3

head(mydata, n=10)

##          y     x1   x2
## 1    98.8  41.3 3.43
## 2   244.5 205.2 3.80
## 3   375.7 186.7 2.51
## 4   -59.2  11.2 4.71
## 5   313.0 111.0 3.45
## 6   288.8 185.1 2.72
## 7   134.8 165.0 4.39
## 8   171.7  97.4 3.64
## 9   167.2 101.0 3.50
## 10  121.1  94.5 4.10
```

Generating Poisson data

```
rpois(10, 1)

##  [1] 3 0 0 0 0 0 0 2 1 2

rpois(10, 2)

##  [1] 1 4 3 1 0 1 1 4 1 1

rpois(10, 20)

##  [1] 18 17 27 19 11 33 24 25 11 11

ppois(2, 2) ## Cumulative distribution Pr(x <= 2)

## [1] 0.677

ppois(4, 2) ## Pr(x <= 4)

## [1] 0.947

ppois(6, 2) ## Pr(x <= 6)

## [1] 0.995
```

Generating Random Numbers From a Linear Model

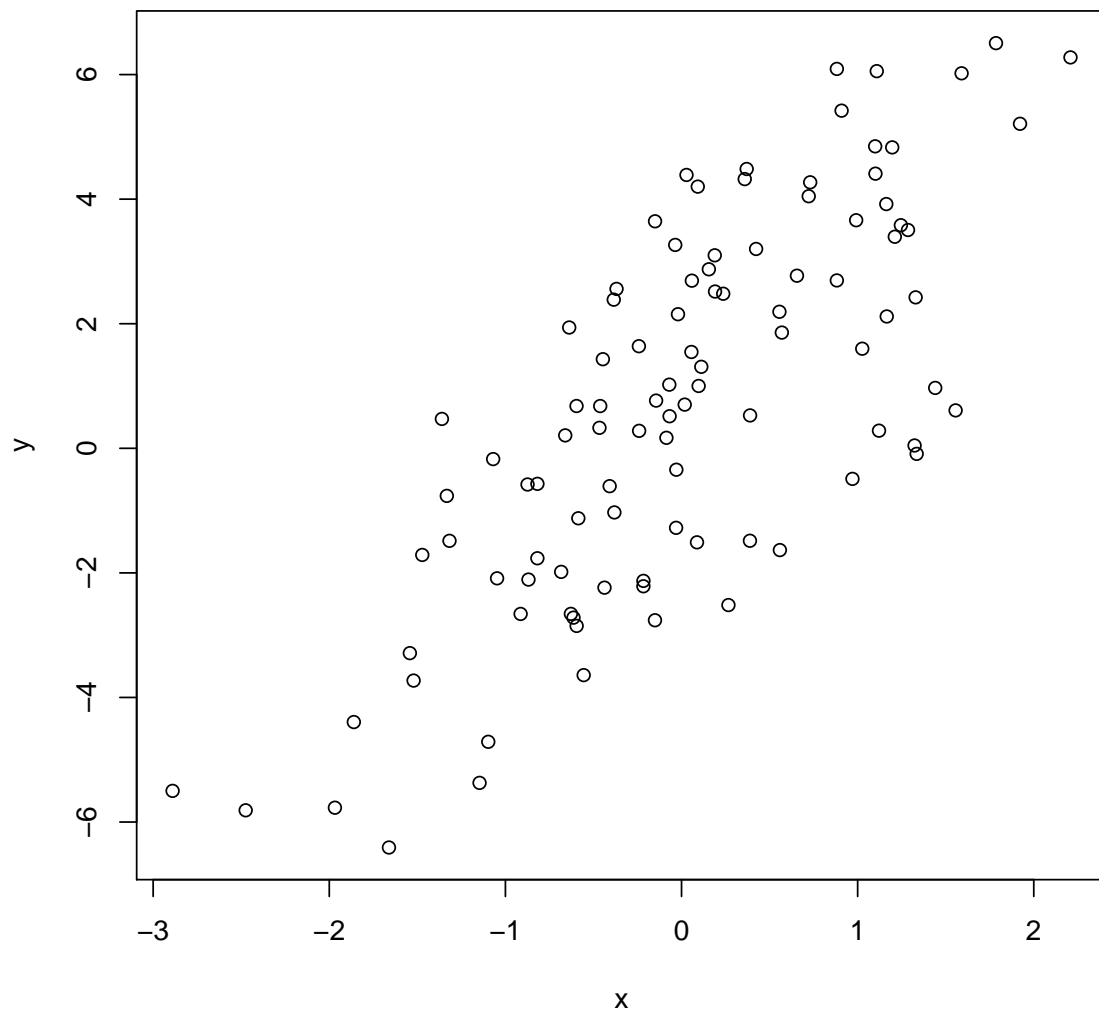Suppose we want to simulate from the following linear model

$$y = \beta_0 + \beta_1 x + \varepsilon$$

where $\varepsilon \sim N(0, 2^2)$. Assume $x \sim N(0, 1^2)$, $\beta_0 = 0.5$, and $\beta_1 = 2$.

```
set.seed(20)
x <- rnorm(100)
e <- rnorm(100, 0, 2)
y <- 0.5 + 2 * x + e
summary(y)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -6.41   -1.54    0.68    0.69    2.93    6.51

plot(x, y)
```


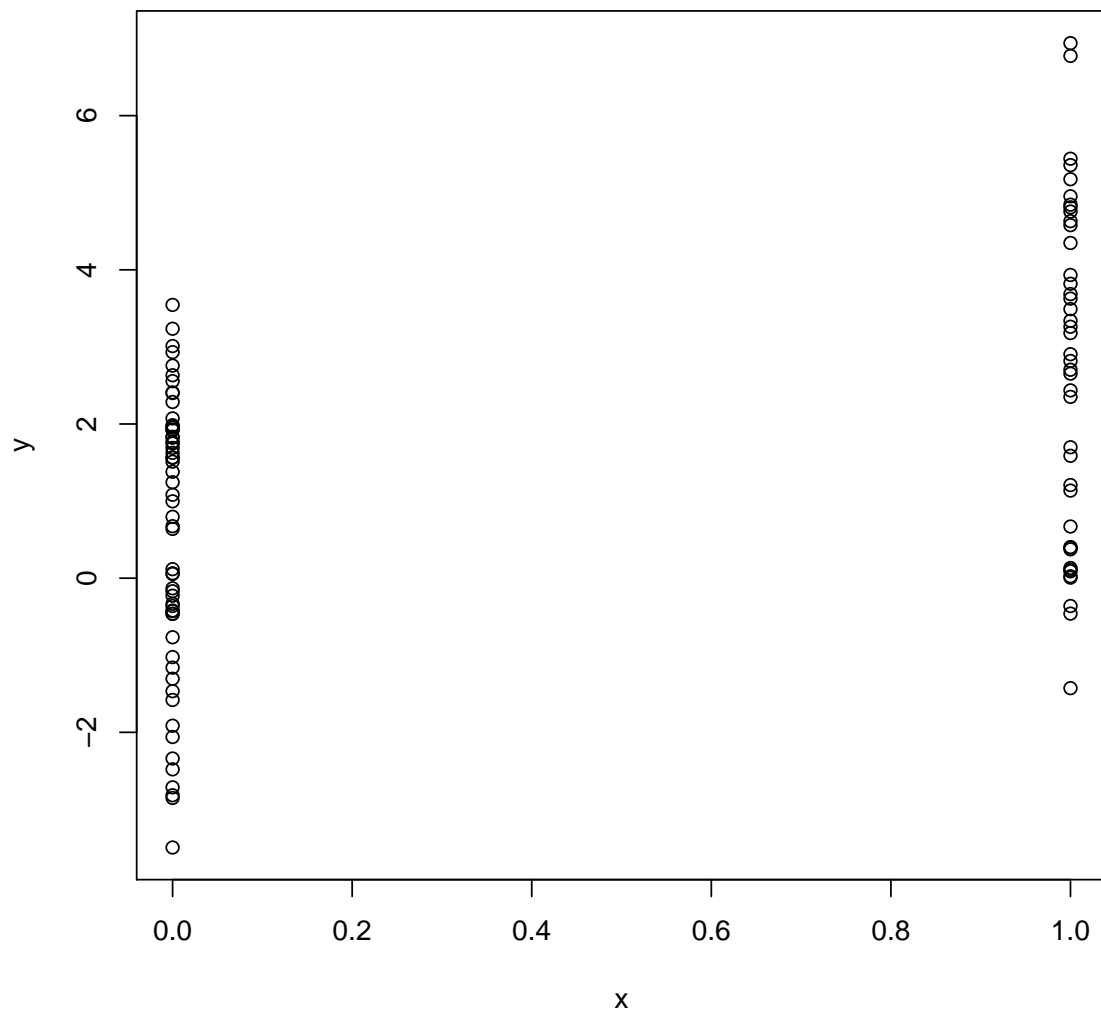What if x is binary?

```
set.seed(10)
x <- rbinom(100, 1, 0.5)
e <- rnorm(100, 0, 2)
y <- 0.5 + 2 * x + e
summary(y)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -3.49   -0.14    1.58    1.43    2.84    6.94

plot(x, y)
```

Testing normality

The simplest test of normality (and in many ways the best) is the 'quantile–quantile plot'. This plots the ranked samples from our distribution against a similar number of ranked quantiles taken from a normal distribution. If our sample is normally distributed then the line will be straight. Departures from normality show up as various sorts of non-linearity (e.g. S-shapes or banana shapes). The functions you need are qqnorm and qqline (quantile–quantile plot against a normal distribution)

```r
y <- rnorm(5000)
shapiro.test(y)

##
##  Shapiro-Wilk normality test
##
## data:  y
## W = 1, p-value = 0.8

qqnorm(y)
qqline(y,lty=2)
```

Normal Q–Q Plot
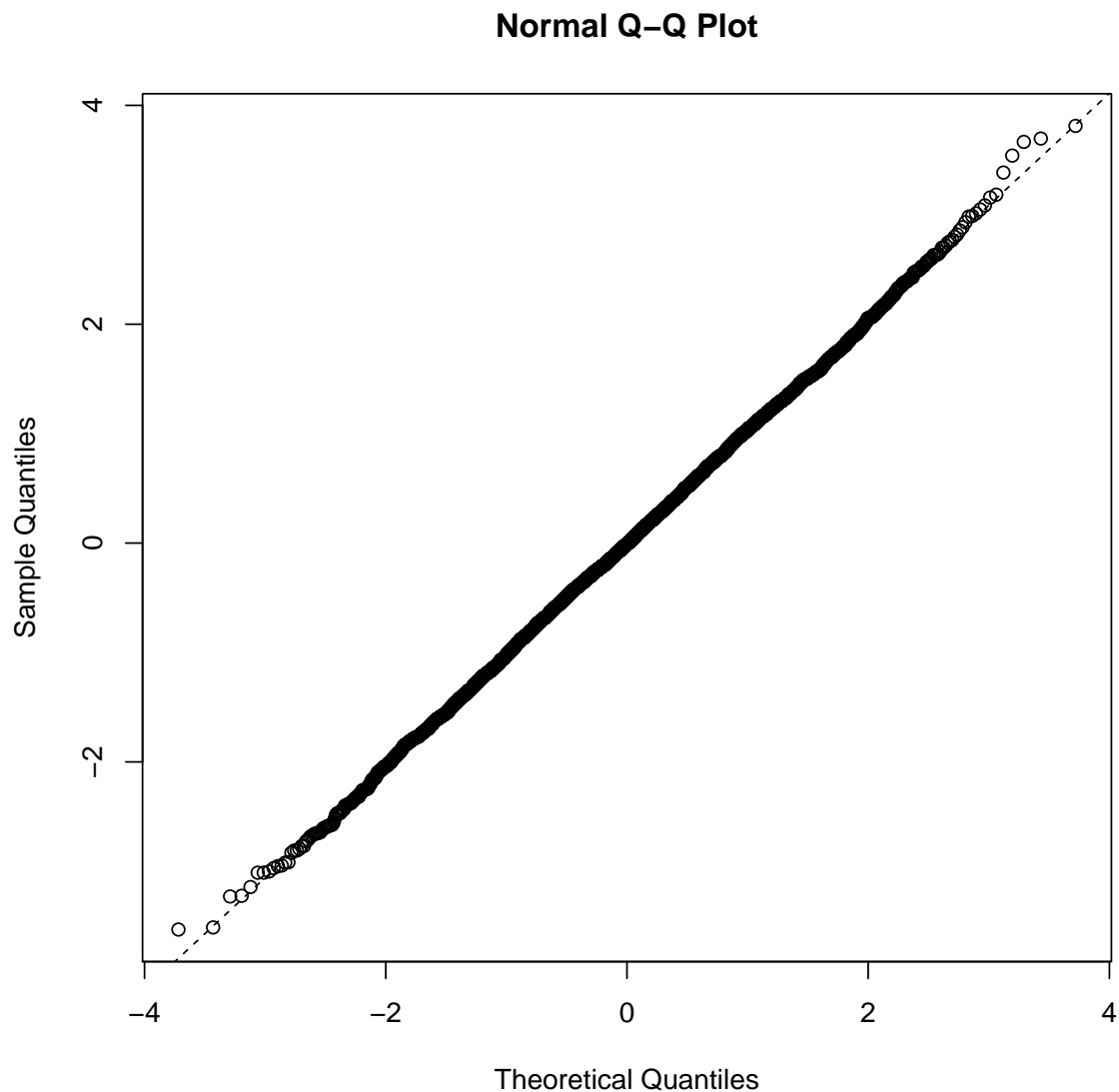
4   Simulation

4.1   Buffon's needle problem

In mathematics, Buffon's needle problem is a question first posed in the 18th century by Georges-Louis Leclerc, Comte de Buffon:

Suppose we have a floor made of parallel strips of wood, each the same width, and we drop a needle onto the floor. What is the probability that the needle will lie across a line between two strips?

Buffon's needle was the earliest problem in geometric probability to be solved; it can be solved using integral geometry. The solution, in the case where the needle length is not greater than the width of the strips, can be used to design a Monte Carlo method for approximating the number π, although that was not the original motivation for de Buffon's question.

With the position random variables defined in this way, we clearly have $0 \leq x \leq d/2$ and $0 \leq \psi \leq \pi$.
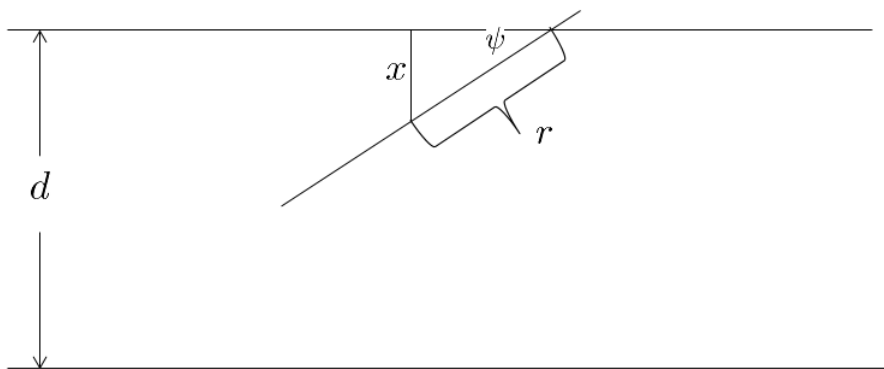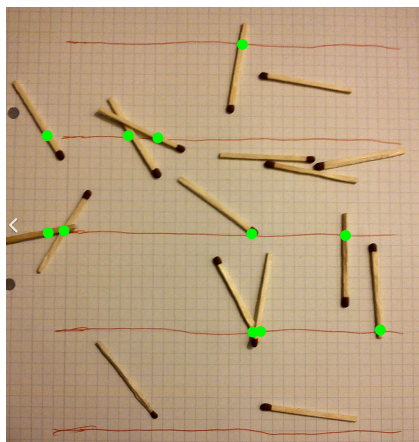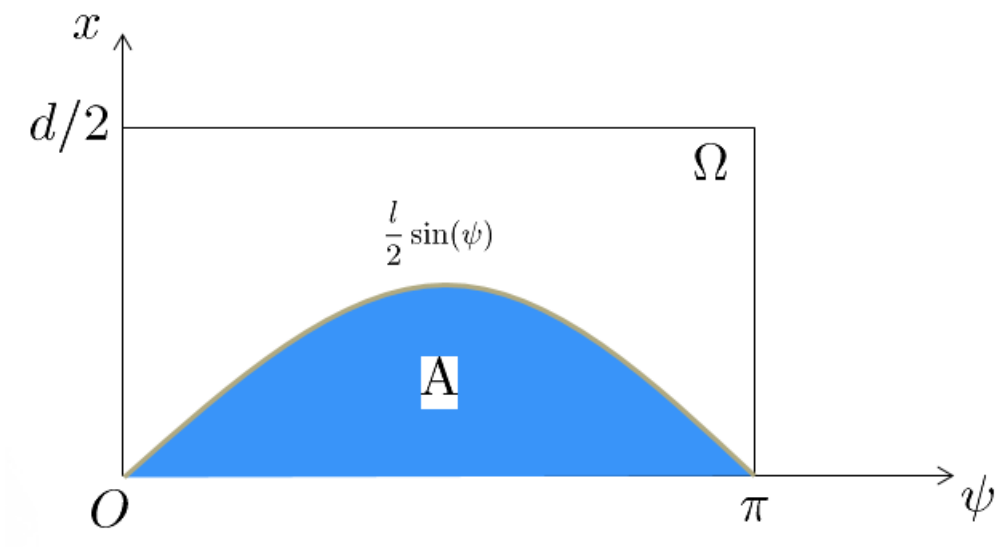
The needle hitting a line, which means

$$x \leq \frac{l}{2}\sin(\psi)$$

Working in the joint sample space. We are asked for the probability that the needle intersects one of the parallel lines. To answer this question, all we need do is identify the set of points in the joint sample space.

$$P(A) = \frac{S_A}{S_\Omega} = \frac{\int_0^\pi \frac{l}{2}\sin\psi\,d\psi}{\frac{d}{2}\pi} = \frac{2l}{d\pi}$$

$$\pi = \frac{2lN}{dn}$$

```
install.packages("animation")
library(animation)
buffon.needle(namx=500, interval=0)
```

```
buffon<-function(n, l=0.8, a=1){
   k<-0
   theta<-runif(n,0, pi); x<-runif(n,0, a/2)
    for (i in 1:n){
    if (x[i]<= l/2*sin(theta[i]))
       k<-k+1
  }
   2*l*n/(k*a)
}
buffon(10000)

## [1] 3.11
```

Exercise

Let X_n and Y_n be iid random variables each having a uniform distribution with joint probability density function are

$$f(x, y) = \begin{cases} 1, & 0 < x < 1, 0 < y < 1 \\ 0, & \text{others} \end{cases}$$

then $P(X^2 + Y^2 \leq 1) = \pi/4$, calculate π.

```
MC1 <- function(n){
   k <- 0; x <- runif(n); y <- runif(n)
   for (i in 1:n){
      if (x[i]^2+y[i]^2 < 1)
         k <- k+1
   }
   4*k/n
}
MC1(10000)

## [1] 3.14
```

4.2   Central limit theorem

Let $X_1, X_2, \ldots X_n$ denote the observations of a random sample from a distribution that has mean μ and positive variance σ^2. Then the random variable

$$Y_n = \frac{\sum_{i=1}^{n} X_i - n\mu}{\sqrt{n}\sigma} = \frac{\sqrt{n}(\bar{X}_n - \mu)}{\sigma}$$

converges in distribution to a random variable which has a normal distribution with mean zero and variance 1.

```
f=function(n) rchisq(n,5)
clt.ani(FUN = f)
```

4.3   Create random variables

The following step can create random variables obeying normal distribution

1. Suppose X be a iid random variable and sample from a uniform distribution. Creating $x_1, x_2, \ldots x_{12}$.

2. Calculating $y = x_1 + x_2 + \cdots + x_{12} - 6$. According to Central limit theorem, y is a random variable sample from a normal distribution with parameter $N(0, 1)$.

3. Calculating $z = \mu + \sigma y$. z is a normal distribution from $N(\mu, \sigma^2)$.

4. Repeat (1)-(4) k times.

```
MCN <- function(k){
        y <- numeric()
        for (i in 1:k){
                x <- runif(n=12,min=0,max=1)
                y[i]<-sum(x)-6
        }
        hist(y)
}
MCN(10000)
```

Histogram of y
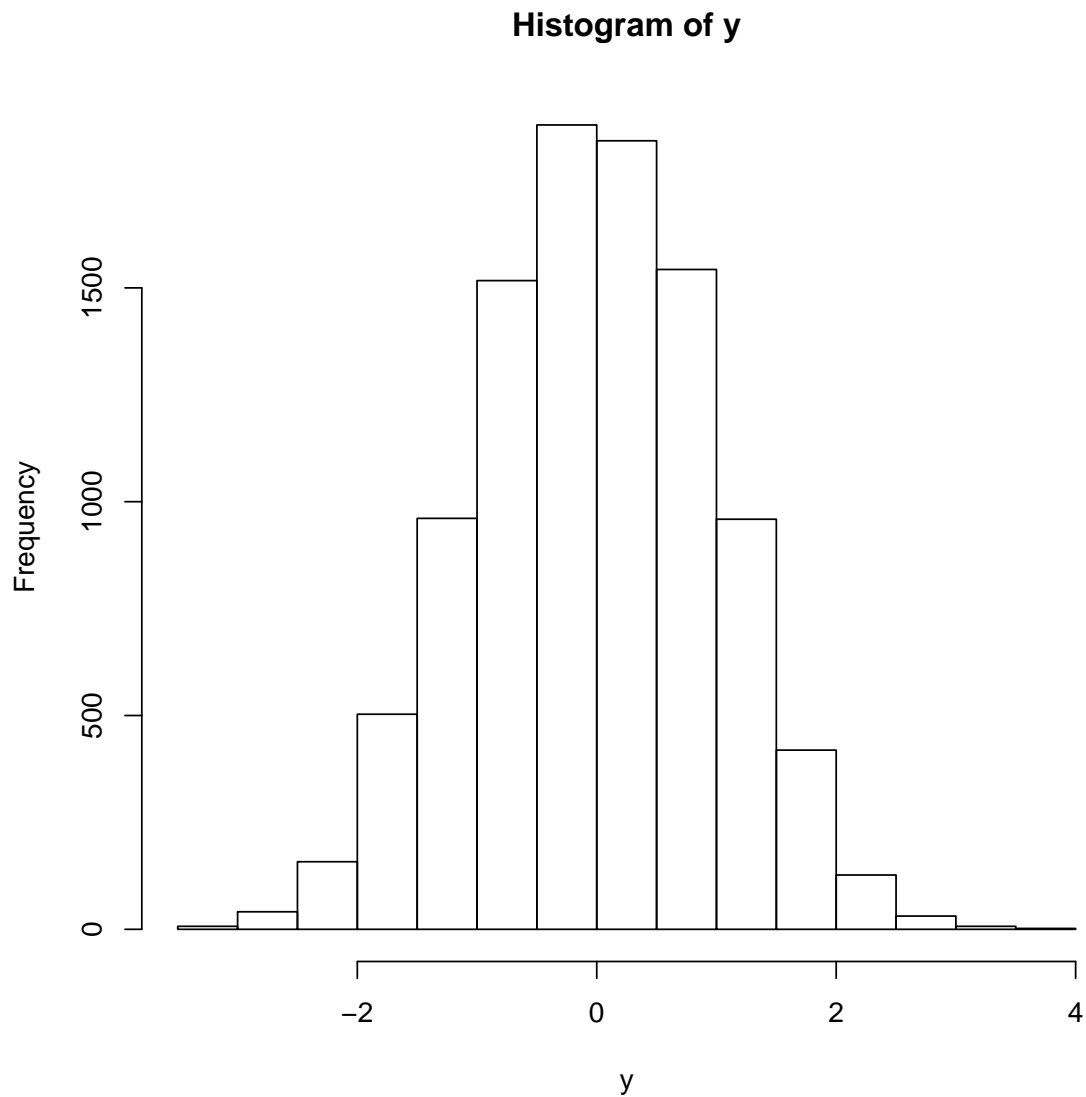
4.4   Monte Carlo Experiments

Since the least-squares estimators under the assumptions of CLRM have certain desirable statistical features summarized in the BLUE property. In this section, we prove this property more formally. But

in practice how does one know that the BLUE property holds? For example, how does one find out if the OLS estimators are unbiased? The answer is provided by the so-called Monte Carlo experiments, which are essentially computer simulation, or sampling, experiments.

Consider the following model:

$$Y_i = \beta_i + \beta_2 X_i + u_i \tag{1}$$

A Monte Carlo experiment proceeds as follows:

1. Suppose the true values of the parameters are as follows: $\beta_1 = 20$ and $\beta_2 = 0.6$.

2. You choose the sample size, say $n = 25$.

3. You fix the values of X for each observation. In all you will have 25 X values.

4. Suppose you go to a random number table, choose 25 values, and call them u_i (these days most statistical packages have built-in random number generators).

5. Since you know β_1, β_2, X_i, and u_i, using (1) you obtain 25 Y_i values.

6. Now using the 25 Y_i values thus generated, you regress these on the 25 X values chosen in step 3, obtaining $\hat{\beta}_1$ and $\hat{\beta}_2$, the least-squares estimators.

7. Suppose you repeat this experiment 99 times, each time using the same β_1, β_2, and X values. Of course, the u_i values will vary from experiment to experiment. Therefore, in all you have 100 experiments, thus generating 100 values each of β_1 and β_2. (In practice, many such experiments are conducted, sometimes 1000 to 2000.)

8. You take the averages of these 100 estimates and call them $\bar{\hat{\beta}}_1$ and $\bar{\hat{\beta}}_2$.

9. If these average values are about the same as the true values of β_1 and β_2 assumed in step 1, this Monte Carlo experiment "establishes" that the least-squares estimators are indeed unbiased. Recall that under CLRM $E(\hat{\beta}_1) = \beta_1$ and $E(\hat{\beta}_2) = \beta_2$.

These steps characterize the general nature of the Monte Carlo experiments. Such experiments are often used to study the statistical properties of various methods of estimating population parameters. They are particularly useful to study the behavior of estimators in small, or finite, samples.

```
###MCMC
beta0=1:1000
beta1=1:1000
b1=20;b2=0.6;n=25
x=10:34

for(i in 1:1000){
u=rnorm(25)
y=b1+b2*x+u
lm.m=lm(y~x)
beta0[i]=as.numeric(coef(lm.m)[1])
beta1[i]=as.numeric(coef(lm.m)[2])
}
#beta0
#beta1
mean(beta0)

## [1] 20
```

```
mean(beta1)

## [1] 0.6
```

Monte Carlo integration

Calculate

$$\theta = \int_0^1 e^{-x}dx$$

```
m <- 10000
#1
k<-0
for (i in 1:m){
        x<-runif(1)
        y<-runif(1)
        ifelse(y<=exp(-x),k<-k+1,k<-k)
}
J<-k/m;J

## [1] 0.631

#2
x <- runif(m)
theta.hat <- mean(exp(-x))
print(theta.hat)

## [1] 0.634

#TURE
print(1 - exp(-1))

## [1] 0.632
```

If we need to calculate $\int_a^b g(x)dx$, and $a < b$, then

$$y = (x - a)/(b - a)$$

and

$$\int_a^b g(x)dx = \int_0^1 g(y(b - a) + a)(b - a)dy$$

Further, if $c \le g(x) \le d$,

$$f(y) = \frac{1}{d - c}[g(a + (b - a)y) - c]$$

The above Equation is $0 \le f(y) \le 1$. Finally,

$$\int_a^b g(x)dx = S_0 \int_0^1 f(y)dy + c(b - a)$$

where $S_0 = (b - a)(d - c)$.

Exercise

Calculate

$$\int_0^1 e^{-x^2/2}/\sqrt{2\pi}dx$$

```
#1
m<-10000;k<-0
for (i in 1:m){
        x<-runif(1)
        y<-runif(1)
        ifelse(y<=exp(((-x^2)/2))/sqrt(2*pi),k<-k+1,k<-k)
}
J<-k/m;J

## [1] 0.343

#2
x <- runif(m)
theta.hat <- mean(exp(((-x^2)/2))/sqrt(2*pi))
print(theta.hat)

## [1] 0.341

#TURE
#0.341344
```

4.5   Bootstrap

```
library(bootstrap) #for the law data
head(law)

##    LSAT GPA
## 1   576 339
## 2   635 330
## 3   558 281
## 4   578 303
## 5   666 344
## 6   580 307

print(cor(law$LSAT, law$GPA))

## [1] 0.776

#set up the bootstrap
B <- 200 #number of replicates
n <- nrow(law) #sample size
R <- numeric(B) #storage for replicates
#bootstrap estimate of standard error of R
for (b in 1:B) {
        #randomly select the indices
```
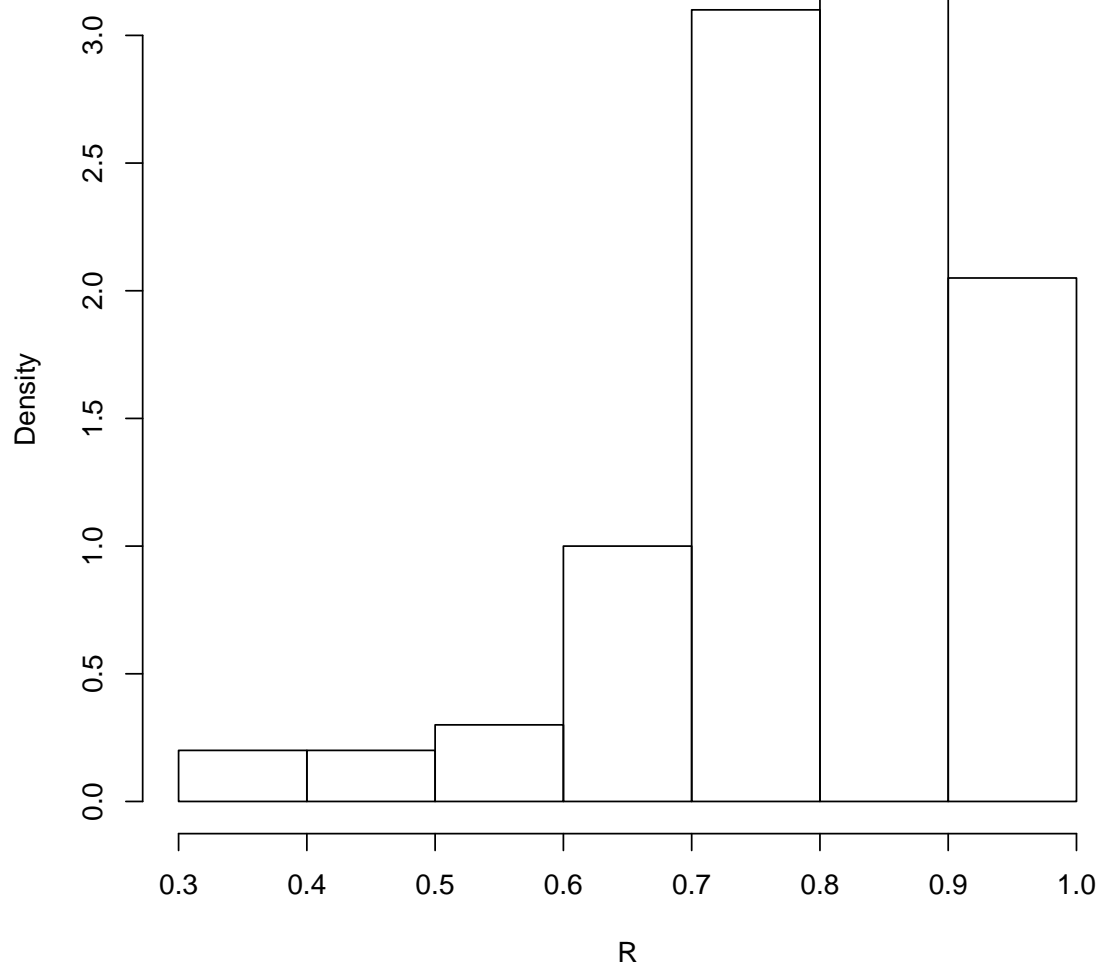
```
        i <- sample(1:n, size = n, replace = TRUE)
        LSAT <- law$LSAT[i] #i is a vector of indices
        GPA <- law$GPA[i]
        R[b] <- cor(LSAT, GPA)
}
#output
hist(R, prob = TRUE)
```

Histogram of R


```
mean(R)
```

```
## [1] 0.794
```