Ch1

Sequence

```
> 1:20-rep(seq(0,9, by=3), rep(5,4))
 [1]  1  2  3  4  5  3  4  5  6  7  5  6  7  8  9  7  8  9 10 11
```

t-statistics

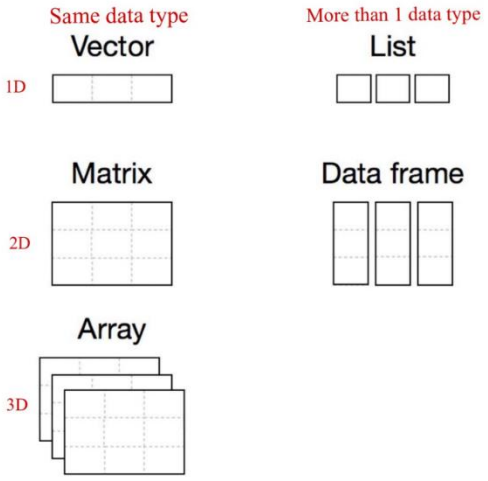
```
(t <- ((mean(x)-mean(y)))/(PooledSD*sqrt(1/n1+1/n2))) # t-statistic
qt(.975,n1+n2-2)  # critical value
(abs(t) > qt(.975,n1+n2-2)) # if TRUE, we should reject H_0
```

Object & Class & Data Type

object	possible modes	several modes possible in the same object ?
vector	numeric, character, complex, or logical	No
factor	numeric, or character	No
array	numeric, character, complex, or logical	No
matrix	numeric, character, complex, or logical	No
data.frame	numeric, character, complex, or logical	Yes
ts	numeric, character, complex, or logical	Yes
list	numeric, character, complex, logical, function, expression, or formula	Yes

`mode()` describe the **data type** used for storage, e.g., numeric, logical, character, etc.

`class()` describe the **object class** of the input variable, e.g., numeric, integer, list, matrix, factor, etc.



Ch2

Column Mean Using by()

```
by(data[,c(2,3)],data$Region,colSums)
```

Confidence Interval

```
n <- 1000
X <- rnorm(n)
Est <- mean(abs(X))    # estimate
SE <- sd(abs(X))/sqrt(n) # standard error
CI95 <- c(Est-qnorm(0.975)*SE, Est+qnorm(0.975)*SE)
# 95% confidence interval
c(Est, sqrt(2/pi), CI95)
```

Aggregate()

To split the variable `year86` by `Region`, we can use `aggregate(year86~Region,d,mean)`

```
aggregate(cbind(year86,year90)~Region+
dense,d,mean)
```

Ch3

Single Bar Chart

```
barplot(USPE[1:3,1],ylim=c(0,25),cex.names=0.8)
```

More about Bar Chart Arguments

```
a<-barplot(
  USPE[1:3,1:2], col=rainbow(3), ylim=c(0,50),
  beside=T, legend=T,
  args.legend=list(x="topright",bty="n",inset=c(-0.08, -0.02),cex=0.8),
  # Inset = Distance from Margin
  xlab="Year",
  ylab="Personal Expenditure",
  main="US Personal Expenditure in 1940 and 1945"
)
```

QQ-Plot for Uniform Distribution

```
n<-length(r)
r2<-sort(r)
i<-((1:n)-0.5)/n
q<-qunif(i)
plot(q,r2,main="Uniform QQ Plot")
abline(lsfit(q,r2), col="red")
```

type = "" Argument for Plot

- "p": is used for points plot
- "l": is used for lines plot
- "b": is used for both point plot and lines plot in a single place
- "c": is used to join empty point by the lines
- "o": is used for both lines and over-plotted point
- "h": is used for 'histogram plot'
- "s": is used for stair steps
- "n": is used for no plotting

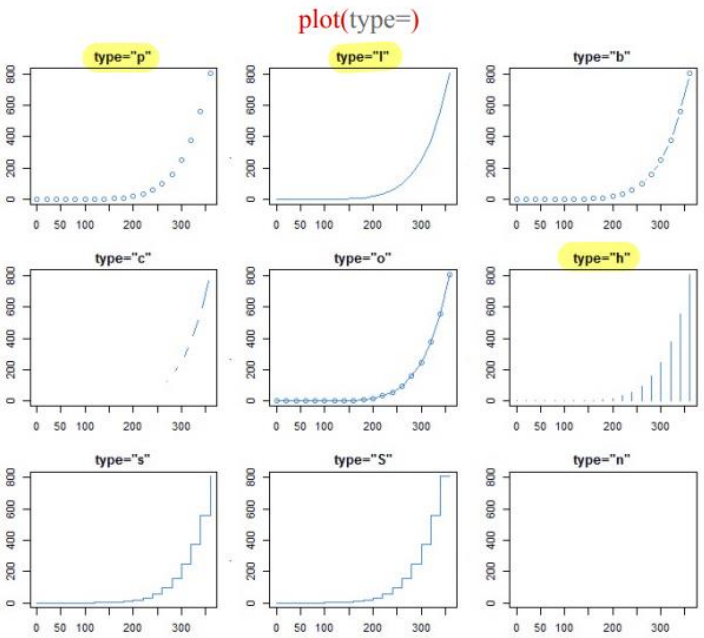
Conditional Selection on Dataframe

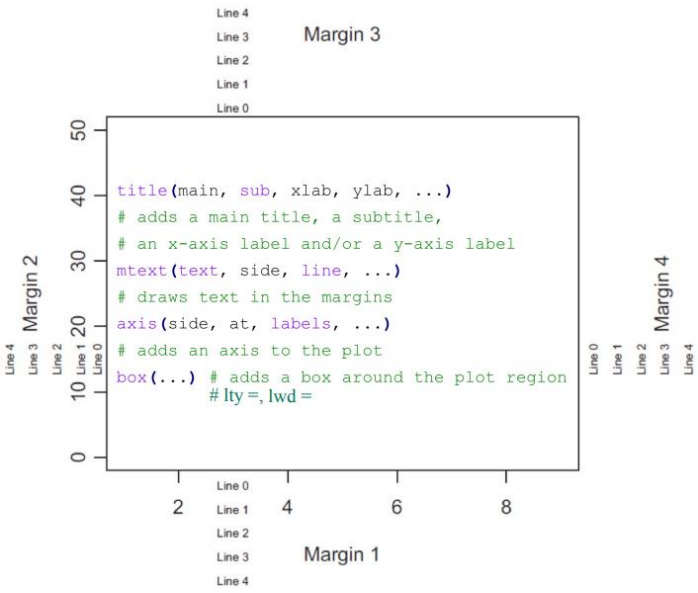
`d[d$year86<d$year90,]` would select the observation according to this logical vector.

Time Series

One commonly used transformation in financial time series is $u_t = \ln(S_t/S_{t-1})$, where S_t is the stock price or index at time t.

```
u1<-log(lag(t1)/t1)
u2<-log(lag(t2)/t2)
u3<-log(lag(t3)/t3)
```





Empty Plot

```
plot(0, 0, type="n", xlim=c(0,10), ylim=c(0,10),
     bty="n", xlab="", ylab="")
```

Ch4

Prime List

```
prime_list <- function(n) {
    if (n >= 2) {
        comp <- seq(2, n)
        primes <- c()
        for (i in seq(2, n)) {
            if (any(comp == i)) {
                primes <- c(primes, i)
                comp <- comp[(comp %% i) != 0]
            }
        }
        return(primes)
    } else {
        stop("Input value of n should be at least 2.")
    }
}
```

Fibonacci numbers

```
Fib1 <- 1
Fib2 <- 1
Fibonacci <- c(Fib1)
while (Fib2 < 300) {
    Fibonacci <- c(Fibonacci, Fib2)
    oldFib2 <- Fib2
    Fib2 <- Fib1 + Fib2
    Fib1 <- oldFib2
}
```

Compound Interest

```
r <- 0.11
period <- 1/12
debt_initial <- 1000
repayments <- 12
time <- 0
debt <- debt_initial
while (debt > 0) {
    time <- time + period
    debt <- debt*(1 + r*period) - repayments
}
cat('Loan will be repaid in', time, 'years\n')
```

Max Consecutive Appearance

```
max1<-function(v) {
    is_prev1<-FALSE       # initialize flag to False
    n1<-0; count<-0  # initialize counter
    for (i in v) {
        if ((i==1)&(is_prev1==TRUE)) {
            count<-count+1
            if (count>=n1) n1<-count
            next       # skip to next element in v
        }
        if ((i==1)&(is_prev1==FALSE)) {
            count<-1; is_prev1<-TRUE
            if (count>=n1) n1<-count
            next       # skip to next element in v
        }
        if ((i==0)&(is_prev1==TRUE)) {
            count<-0    # reset counter
            is_prev1<-FALSE
        }
    }
    return(n1)
}
```

Normal Table

```
y<-seq(0,3.4,0.1)
# define sequence of y from 0 to 3.4 with step 0.1
x<-seq(0,0.09,0.01)
# define sequence of x from 0 to 0.09 with step 0.01
z<-outer(y,x,"+")
# save the table to z, where z(i,j)=y(i)+x(j)
options(digits=4)      # specify output display to 4 decimal place
t<-pnorm(z)            # compute the left tail and save them to t
t<-rbind(x,t)          # add the first row to t
y<-c(0,y)              # add a zero to y
cbind(y,t)             # output the table
```

```
  y
x 0.0 0.0000 0.0100 0.0200 0.0300 0.0400 0.0500 0.0600 0.0700 0.0800 0.0900
  0.0 0.5000 0.5040 0.5080 0.5120 0.5160 0.5199 0.5239 0.5279 0.5319 0.5359
  0.1 0.5398 0.5438 0.5478 0.5517 0.5557 0.5596 0.5636 0.5675 0.5714 0.5753
  0.2 0.5793 0.5832 0.5871 0.5910 0.5948 0.5987 0.6026 0.6064 0.6103 0.6141
```

Ch5

Customise Operator

```
> "%+-%" <- function(x,s) { c(x-s,x+s) }
> 3 %+-% 5
[1] -2  8
```

Formatting Output

```
> sprintf("Pi is %f", pi)
# output real number with default option = 6 decimal places
[1] "Pi is 3.141593"
> sprintf("%.3f", pi)     # with 3 decimal places
[1] "3.142"
> sprintf("%5.1f", pi)    # fixed width=5 with 1 decimal places
[1] "  3.1"
> sprintf("%-10f", pi)    # left justified with fixed width=10
[1] "3.141593   "
> sprintf("%e", pi) # scientific notation
[1] "3.141593e+00"
```

Checking Symmetric Matrix

```
n1<-dim(x)[1]    # get the row dimension of x
n2<-dim(x)[2]    # get the column dimension of x
if (n1!=n2) stop("Input matrix is not a square matrix")
for (i in 1:n1) {      # check if x is symmetric
      for (j in 1:i) {
            if (x[i,j]!=x[j,i])
                  stop("Input matrix is not a symmetric matrix")
      }
}
```

Sierpinski triangle

```
set.seed(1234)     # set random seed
n<-5000            # number of points
for (i in 1:n) {
      col<-sample(c("b","g","r"),prob=c(1/3,1/3,1/3),size=1)
      # randomly pick a color
      if (col=="b") {          # color=blue
            x<-(x0+b1)/2       # mid-point between x0 and b
            y<-(y0+b2)/2
            points(x,y,pch=21,bg="blue")   # plot this point in blue
      }
      if (col=="g") {          # color=green
            x<-(x0+g1)/2       # compute mid-point bewtten x0 and g
            y<-(y0+g2)/2
            points(x,y,pch=21,bg="green")  # plot this point in green
      }
      if (col=="r") {          # color=red
            x<-(x0+r1)/2       # compute mid-point between x0 and r
            y<-(y0+r2)/2
            points(x,y,pch=21,bg="red")          # plot this point in red
      }
      x0<-x                    # update x0
      y0<-y                    # update y0
}
```

Recursive Function

```
fac<-function(n){
      # factorial function, assume n is an integer > 0
      if (n<=2) return(n)
      else return(n*fac(n-1))
      # fac calls itself; fac(n)=n*fac(n-1)
}
```

Slash Matrix

```
slash <- function(X) {
    m<-ncol(X)
    n<-nrow(X)
    (outer(1:n,1:m,"+")==min(m,n)+1)*X

    # for (i in 1:n) {
        # for (j in 1:m) {
            # if ((i+j)!=min(m,n)+1) X[i,j] <- 0
        # }
    # }
    # return(X)
}
```

Customise Sort()

```
sort <- function(x) {
    # x is initially the input vector and will be
    # modified to form the output
    for(last in length(x):2) {
        for(first in 1:(last - 1)) {
            if(x[first] > x[first + 1]) {
                # swap the pair
                save <- x[first]
                x[first] <- x[first + 1]
                x[first + 1] <- save
            }
        }
    }
    return (x)
}
```