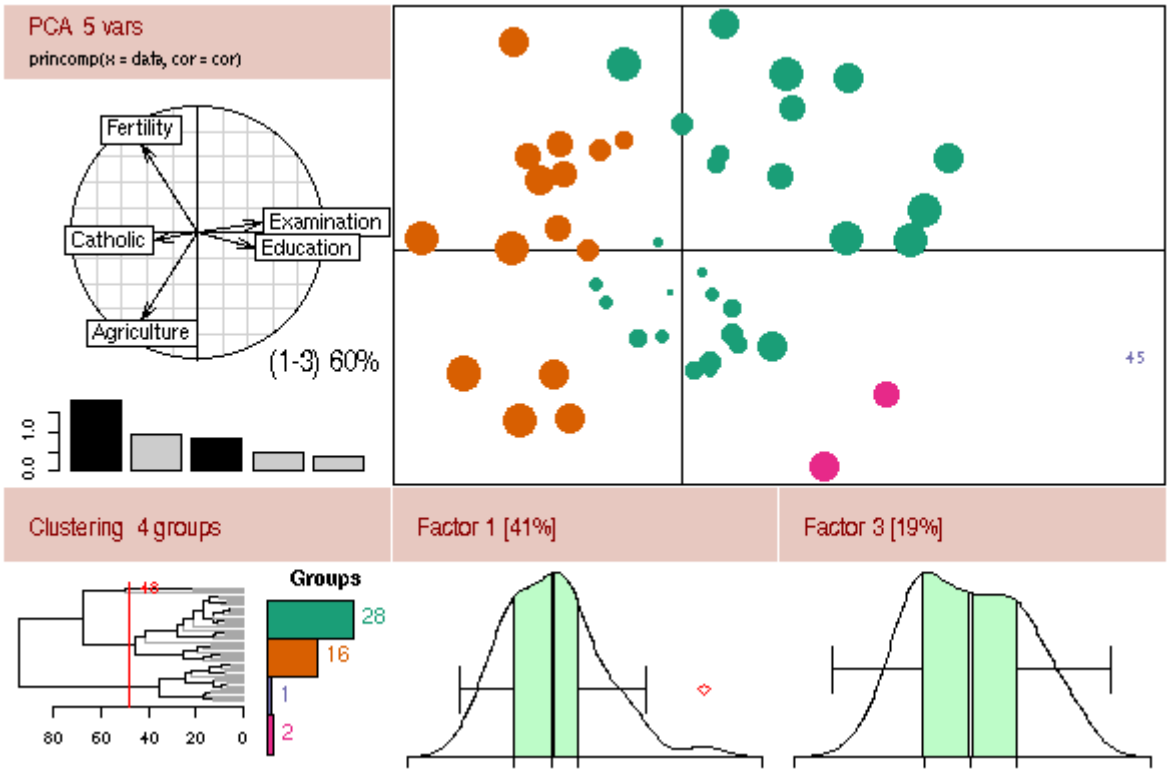
Software Livre R: aplicação estatística



Emanuel Fernando Maia de Souza

Luiz Alexandre Peternelli

Márcio Pupin de Mello

.

Índice

1. Algo sobre o R

1.1. Introdução

O uso de pacotes estatísticos para a análise de dados é de grande importância no que se refere à análise e a interpretação de resultados. Contudo observa-se que estes apresentam um custo de aquisição relativamente elevado, ou a criação de programas de alternativos. Dentre os softwares de domínio público, livres, que podem ser utilizados para análise de dados em geral, encontra-se o Ambiente R, ou simplesmente R, conforme usualmente chamado pelos seus usuários, apresenta código fonte aberto, podendo ser modificado ou implementado com novos procedimentos desenvolvidos por qualquer usuário a qualquer momento. Além do que, o R com um grande número de colaboradores das mais diversas áreas do conhecimento.

O R torna-se, portanto, uma importante ferramenta na análise e na manipulação de dados, com testes paramétricos e não paramétricos, modelagem linear e não linear, análise de séries temporais, análise de sobrevivência, simulação e estatística espacial, entre outros, além de apresentar facilidade na elaboração de diversos tipos de gráficos, no qual o usuário tem pleno controle sobre o gráfico criado.

O R pode ser obtido gratuitamente em http://cran.r-project.org, onde é apresentado em versões de acordo com o sistema operacional UNIX, Windows ou Macintosh. Além disso, encontra-se neste site mais informação sobre a sua utilização e uma central de correspondências onde profissionais de várias áreas do conhecimento podem contribuir na implementação de novos recursos assim como responder a dúvidas dos demais usuários.. Como o R é uma linguagem de programação orientada a objetos o usuário pode criar suas próprias funções, e sua própria rotina na análise de dados. Outro atributo do R é sua capacidade de interagir com outros programas estatísticos, bem como de banco de dados.

1.2. O programa

O R é uma linguagem orientada a objetos criada em 1996 por Ross Ihaka e Robert Gentleman que aliada a um ambiente integrado permite a manipulação de dados, realização de cálculos e geração de gráficos. Semelhante à linguagem S desenvolvida pela AT&T's Bell Laboratories e que já é utilizada para análise de dados (veja, por exemplo, Crawley, 2002), mas com a vantagem de ser de livre distribuição.

É importante salientar que o R não é um programa estatístico, mas que devido a suas rotinas permite a manipulação, avaliação e interpretação de procedimentos estatísticos aplicado a dados. O *R Core Team* ("defensores e detentores" do R o classificam como Ambiente R dado a suas características, entretanto, o abordaremos como um sistema integrado que permite a execução de tarefas em estatística).

Além dos procedimentos estatísticos o R permite operações matemáticas simples, e manipulação de vetores e matrizes. Assim como confecção de diversos tipos de gráficos.

1.3. Como instalar

Para a instalação do R basta conectar-se ao site http://cran.r-project.org, em CRAN "Comprehensive R Archive Network" escolher o local mais próximo de onde você encontra-se. No caso de Viçosa-MG: http://www.insecta.ufv.br/CRAN. Dar um clique duplo no link que corresponde ao sistema operacional do seu computador, e depois no link *base*, em seguida escolha o arquivo executável.

Agora basta seguir a rotina de instalação, e após instalado deve-se iniciar o R e clicar na barra de ferramentas em:

Packages, UPDATE PACKAGES FROM CRAN; para receber as versões atualizadas dos principais pacotes (é necessário que o computador esteja conectado na internet).

2. Iniciando o R

Com o R iniciado você verá o símbolo “>” em vermelho, que é o *prompt* do R (conhecido também como *R Console*) indicando que o sistema está pronto para receber seus comandos. Acima do *prompt* em cor azul encontram-se algumas informações sobre o sistema e alguns comandos básicos.

As funções disponíveis ficam guardadas em uma livraria localizada no diretório R_HOME/library (R_HOME é o diretório onde foi instalado o R). Esse diretório contém “pacotes de funções” (conhecidos como *packages*) que, por sua vez, estão estruturadas em diretórios. Estes são os principais pacotes do R. Todavia existem inúmeros outros pacotes que podem ser encontrados no CRAN. O pacote chamado BASE constitui o núcleo do R, contendo as funções básicas. Cada um desses pacotes instalados possui um diretório próprio, por exemplo, para o pacote BASE existe um caminho R_HOME/library/base/R/base com um arquivo em código ASCII que contém todas as funções desse pacote. Junto com o pacote *base* existem outros pacotes que já vêem carregados com o R. Essa estrutura permite que o seja necessário menos recursos computacionais para operar com o R.

Os pacotes podem ser carregados na memória a qualquer instante através da linha de comando do R digitando `library(nome_do_pacote)`. Para saber quais são os pacotes carregados basta digitar `search()`. Para se trabalhar com o R são necessários alguns conceitos que serão discutidos neste material e se constitui o foco deste curso.

2.1. Símbolos ou comandos importantes

Ação	Comando
Sair do programa	`q()`
Salva o trabalho realizado	`save.image()`
Lista todos os objetos da área de trabalho atual	`ls()`
Remove o objeto x	`rm(x)`
Remove os objetos x e y	`rm(x,y)`
Dado ausente (data missing)	`NA`
Mostra todos os pacotes instalados	`library()`
Carregar (p.ex.) o pacote nlme	`require(nlme)`

Para a execução de um linha de comando deve-se pressionar a tecla *“Enter”*

2.2. Obtendo ajuda

A ajuda em linha do R pode seu muito útil quando se deseja saber qual função utilizar ou como utilizar uma função determinada função. Na tabela abaixo são listados alguns comandos para realizar buscas no R:

Ação de Ajuda	Comando
Procurar por "multivariate" em todos os pacotes instalados	`help.search("multivariate")`
Obter ajuda sobre o comando X	`help(comandoX)`
Iniciar ajuda no browser padrão instalado	`help.start()`
Obter ajuda sobre (p.ex.) o pacote cluster	`help(package=cluster)`
Comando que procura objetos (p.ex.) pelo nome anova	`apropos("anova")`
Mostrar exemplos do "comandoX"	`example(comandoX)`
Listar as funções e operações contidas no pacote base do R	`ls("package:base")`

Uma outra opção para obter ajuda sobre funções, documentos, comandos do R é fazer uma busca rápida no site: http://finzi.psych.upenn.edu/ .

Para saber como uma função foi escrita e a qual classe ela pertence, basta digitar o nome da função completo e teclar *entrer*.

2.3. Algumas operações aritméticas

Você pode utilizar o R como uma calculadora, inclusive para cálculos com matrizes, como veremos em capítulos subseqüentes. No entanto, agora, nos atemos apenas a cálculos simples.

Exemplos:

Algumas operações podem ser realizadas apenas com os sinais de operação aritmética.

```
2+3          #somando estes números
 [1] 5
2+3*2        #observando prioridades: multiplicação primeiro, lembra?!
 [1] 8
2**4         #potências utilizado ** ou ^
 [1] 16
```

Outras funções são usadas como as encontradas em calculadoras científicas.

```
sqrt(9)           #raiz quadrada
 [1] 3
sin(3.14159)      #seno de Pi radianos é zero
```

```
 [1] 2.65359e-06
sin(pi)            #bem mais próximo.
 [1] 1.224606e-16
factorial(4)       #4!=4*3*2*1
 [1] 24
```

A tabela abaixo mostra algumas operações possíveis de ser realizadas no R:

Função	Significado
`log(x)`	Log de base e de x
`exp(x)`	Antilog de x (e^x)
`log(x,n)`	Log de base n de x
`Log10(x)`	Log de base 10 de x
`sqrt(x)`	Raiz quadrada de x
`choose(n,x)`	Combinação de n por x: n!/(x!(n-z)!)
`cos(x), sin(x), tan(x)`	Funções trigonométricas de x em radianos
`acos(x), asin(x), atan(x)`	Funções trig. Inversas de x em radianos

Existem outras inúmeras operações no R que não foram citadas aqui por não ser conveniente, porém estão disponíveis e podem ser encontradas em manuais introdutórios, consultar `help.search()`.

2.4. Manipulando objetos

2.4.1. Criando objetos

Um objeto pode ser criado com a operação de "atribuição", o qual se denota como uma flecha, com o sinal de menos e o símbolo ">" ou "<", dependendo da direção em que se atribui o objeto. Ou com um único sinal de igual. É importante dizer que o nome de um objeto deve começar com uma letra qualquer, maiúscula ou minúscula, que pode ser seguida de outra letra, número, ou caracteres especiais como o ponto.

Exemplo:

```
x<-10 #o objeto x receberá o valor 10
15->y #o objeto y receberá o valor 15
X<-6  #o objeto X receberá o valor 6
Y=15  # o objeto Y receberá o valor 15
```

Observe que existe diferença entre maiúscula e minúscula (mesmo para o Sistema Operacional Windows©)

```
x
 [1] 10
```

```
X
 [1] 6
```

OBS.: O símbolo "#" indica para o R um comentário.

Outro Exemplo:

O R pode ser usado para fazer cálculos. Você também pode armazenar o resultado de um cálculo em um objeto qualquer.

```
t<-sqrt(4)  #objeto x irá receber o valor da operação indicada
```

Para mostrar o conteúdo do objeto criado "t", digite apenas o nome do objeto na linha de comando do R, como abaixo:

```
t
 [1] 2
```

O número "1" entre colchetes significa que a visualização do objeto inicia-se pelo seu primeiro elemento. Esse comando é um comando implícito do comando `print()`, ou seja, escrevendo `print(t)` obteríamos o mesmo resultado que escrevendo apenas `t` (dentro de funções esse comando deve ser usado explicitamente).

2.4.2. Listando objetos

Agora que você já criou alguns objetos você certamente quer ter controle sobre eles. A função `ls()` mostra os objetos que você tem.

Exemplo:

```
a<-1; b<-2; c<-3  #observe o uso do ";" para separar os comandos
x<-"uso"; y<-"do comando"; z<-"list()"

ls()  #lista todos os objetos existentes na memória
 [1] "a" "b" "c" "x" "y" "z"
```

Note que o resultado é impresso como um vetor - porque de fato é um vetor! No caso, um vetor de caracteres com os nomes dos objetos existentes. Como em qualquer outra função do R você pode armazenar o resultado em um objeto, inclusive a `ls()`. Veja:

```
obj<-ls()   #armazena a lista de objetos
obj         #exibe a lista armazenada no objeto "obj"
 [1] "a" "b" "c" "x" "y" "z"
```

Provavelmente você não terá razão para fazer isto com freqüência, ou talvez nunca, mas isto ilustra como o R opera ...

2.4.3. Removendo objetos

Há uma função para remover objetos: `remove()` (ou simplesmente `rm()`). Para usar esta função basta fornecer o objeto a ser removido:

```
x<-1                #cria o objeto x
y<-2                #cria o objeto y
a<-10; b<-20        #cria os objetos a e b
rm(a)               #remove o objeto a
remove(b)           #remove o objeto b
rm(x,y)             #remove os objetos x e y
```

Para remover TODOS os objetos na sua área de trabalho digite:

```
remove(list=ls())       #remove TUDO!!!!!!
```

OBS.: ***Tome cuidado ao usar esta função, pois uma vez excluído, o objeto se torna irrecuperável. Além disso, o comando supracitado para remoção de todos os objetos não exibe mensagem de confirmação de exclusão!***

2.5. Atributos dos objetos

Já foi falado várias vezes que o R trabalha com objetos. Esses possuem nome, conteúdo e um atributo associado que especifica qual o tipo de dados representados pelo objeto. Em uma análise estatística, por exemplo, mesmo que dois objetos contenham o mesmo valor, os resultados se diferem quando esses possuem atributos diferentes. A maneira que as funções atuam nos objetos também depende de seu atributo.

Todo objeto possui atributos intrínsecos: *tipo* e *tamanho*. Com relação ao tipo ele pode ser: numérico, caractere, complexo e lógico. Existem outros tipos, como por exemplo, funções ou expressões, porém esses não representam dados.

As funções `mode()` e `length()` mostram o tipo e tamanho de um objeto, respectivamente.

Exemplo:

```
x<-c(1,3,5,7,11)
mode(x); length(x)      #mostra o tipo e tamanho do objeto x
```

```
 [1] "numeric"
 [1] 5
```

```
a<-"Angela"; b<-TRUE; c<-8i   #objetos com tipos diferentes
mode(a); mode(b); mode(c)     #exibe os atributos "tipo" dos objetos
```

```
 [1] "character"
 [1] "logical"
 [1] "complex"
```

Existe uma outra forma de se verificar atributos em um objeto, como por exemplo, usando a palavra "`is`", seguida de um ponto e nome do atributo a qual se deseja verificar.

```
is.numeric(x)     #verifica se o objeto x tem atributo de numérico
 [1] TRUE
```

Foi retornado "TRUE" (do inglês, VERDADEIRO). Caso o objeto x fosse do tipo caractere, por exemplo, o resultado seria FALSE.

A tabela abaixo sintetiza os objetos e seus possíveis atributos (tipos). Veja:

Objeto	Tipos	Suporta tipos diferentes
vetor	numérico, caractere, complexo ou lógico	Não
fator	numérico ou caractere	Não
matriz	numérico, caractere, complexo ou lógico	Não
array	numérico, caractere, complexo ou lógico	Sim
data.frame	numérico, caractere, complexo ou lógico	Sim
ts	numérico, caractere, complexo ou lógico	Sim
lista	numérico, caractere, complexo, lógico, função, expressão, etc	Sim

OBS.: ts é uma série temporal e não será abordada neste material

3. Alguns objetos especiais

Saber as diferenças entre os diversos tipos de objetos é importante para um uso mais adequado do R. Existem vários tipos de objetos que podem ser criados e manipulados.

3.1. Vetores

O R pode trabalhar com vetores - objetos que armazenam mais de um valor.

A função `c()` é usada para criar um vetor a partir de seus argumentos.

Exemplo:

```
x<-c(2,3,5,7,11)  #os 5 primeiros números primos
x                 #digitando o nome é exibido o conteúdo do objeto
```

```
 [1]  2  3  5  7  11
```

Os argumentos de `c()` podem ser escalares ou vetores.

```
y<-c(x,13,17,19)        #adicionando mais três números primos
y
```

```
 [1]  2  3  5  7 11 13 17 19
```

```
k<-c('a','b','c','d')       #ou caracteres alfanuméricos
k
```

```
 [1] "a" "b" "c" "d"
```

3.1.1. Seqüências

Há ainda outras formas de se gerar um vetor. Por exemplo, para gerar uma seqüência de números inteiros usam-se os "dois pontos". Veja:

```
a<-1:10     #cria uma seqüência de inteiros de 1 a 10
a           #exibe o conteúdo do objeto "a"
```

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

Se o vetor é muito longo e não "cabe" em uma linha o R vai usar as linhas seguintes para continuar imprimindo o vetor.

```
longo<-100:50     #seqüência decrescente de 100 a 50
longo             #exibe o conteúdo do objeto
```

```
  [1] 100  99  98  97  96  95  94  93  92  91  90  89  88  87  86  85
 [17]  84  83  82  81  80  79  78  77  76  75  74  73  72  71  70  69
 [33]  68  67  66  65  64  63  62  61  60  59  58  57  56  55  54  53
 [49]  52  51  50
```

Os números entre colchetes não fazem parte do objeto e indica a posição do vetor naquele ponto. Pode-se ver que [1] indica que o primeiro elemento do vetor está naquela

linha, [17] indica que a linha seguinte começa pelo décimo sétimo elemento do vetor e assim por diante.

3.1.2. Usando o comando seq()

Uma maneira mais geral de produzir seqüências de valores é usando a função `seq()` que tem como argumentos o início, fim e passos da seqüência.

```
seq(1,10,1)        #o mesmo que 1:10
 [1]  1  2  3  4  5  6  7  8  9 10
seq(1,10,2)        #de 2 em 2; observe que não terminará no valor 10
 [1] 1 3 5 7 9
seq(10,1,3)        #tentando ordem inversa...
 Erro em seq.default(10, 1, 3) : sinal errado no argumento 'by'

seq(10,1,-3)       #a forma correta é usando passo negativo...
 [1] 10  7  4  1
```

3.1.3. Usando rep()

Outra função útil para produzir vetores é a função `rep()` que retorna o primeiro argumento repetido o número de vezes indicado pelo segundo argumento:

```
rep(1,10)   #cria um repetição
 [1] 1 1 1 1 1 1 1 1 1 1
rep(c(1,2),10)
 [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
rep(c(0,1),c(10,5))  #para cada valor, um número de repetições
 [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
```

Pode-se ainda usar variáveis (objetos) como argumentos das funções:

```
X<-10
rep(c(1,2),X)      #cria um repetição do objeto "c(1,2)" X vezes
```

Se ambos os argumentos tem mais de um elemento, então cada elemento do primeiro argumento será associado ao elemento correspondente do segundo argumento.

Exemplos:

```
rep(4:1,1:4)       #examine cuidadosamente este exemplo
rep(1:5,each=2)    #o mesmo número de repetições para tordos elementos
```

3.2. Listas

Listas são objetos curiosos. Elas permitem combinar diferentes tipos de objetos em um mesmo objeto. Estas coisas podem ser vetores, matrizes, números e/ou caracteres e até mesmo outras listas.

Exemplo:

```
R<-list(versao=2.4, origem='Áustria', notas=c(9,10,8))
R
 $versao
 [1] 21

 $origem
 [1] "Áustria"

 $notas
 [1] 9 10 8
```

Listas são construídas com a função `list()`. Os componentes da lista são introduzidos usando a forma usual (nome = argumento) de atribuir argumentos em uma função. Quando você digita o nome de um objeto que é uma lista cada componente é mostrado com seu nome e valor.

Cada um desses componentes da lista pode ser acessado individualmente por seu nome antecedido pelo símbolo "$":

```
R$versao          #componente "versao" da lista "R"
 [1] 2.4
R$notas[2]  #segundo elemento de $notas
 [1] 10
```

Pode-se ainda acessar cada elemento pelo seu número de ordem na lista utilizando colchetes duplos:

```
pessoa[[1]]
 [1] 2.4
pessoa[[3]]
 [1] 98 95 96
```

3.2.1. Algumas funções que retornam listas

Muitas das funções do R retornam seu resultado na forma de listas. Um exemplo pode ser mostrado com o uso da função `t.test()`, que retorna um objeto que é uma lista.

Exemplo:

Tente isso: um teste t simples para dois conjuntos de números com médias diferentes:

```
tt<-t.test(rnorm(100),rnorm(100),var.equal=T)
```

Note que nada é exibido na tela porque o resultado foi armazenado no objeto "tt". Portanto basta digitar o nome do objeto para exibir seu conteúdo.

```
tt
```

```
        Two Sample t-test

data:  rnorm(100) and rnorm(100)
t = -1.3792, df = 198, p-value = 0.1694
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.49861179  0.08821032
sample estimates:
  mean of x   mean of y
-0.09627982  0.10892092
```

Note que esta saída não se parece muito com o formato de lista visto acima. Mas o resultado é uma lista. Você pode comprovar isso usando o comando abaixo:

```
is.list(tt)
```

```
[1] TRUE
```

E você pode listar os nomes dos componentes da lista com a função `names()`. Para este objeto com os resultados do teste T temos:

```
names(tt)
```

```
[1] "statistic"   "parameter"   "p.value"     "conf.int"    "estimate"
[6] "null.value"  "alternative" "method"      "data.name"
```

...e portanto podemos extrair elementos individualmente como, por exemplo:

```
tt$conf.int
```

```
[1] -0.2923768 -0.1162318
attr(,"conf.level")
[1] 0.95
```

Este componente é um vetor de tamanho 2 que mostra os limites do intervalo de confiança a 95% para diferença de médias, que neste caso não contém o zero. Você pode estar se perguntando: mas e este attr(,"conf.level"), o que é isto?

Isto é algo adicional chamado atributo do vetor. Atributos nos fornecem informações complementares sobre um objeto e muitas vezes são usados internamente pelo programa. Neste caso o atributo nos diz o nível de confiança com o qual o intervalo foi calculado.

3.3. Matrizes

Tudo feito até aqui foi baseado em vetores. Porém o R também é capaz de operar matrizes. Veja a seguir como manipular matrizes.

3.3.1. Criando matrizes

Há várias formas de criar uma matriz. A função `matrix()` recebe um vetor como argumento e o transforma em uma matriz de acordo com as dimensões especificadas.

Exemplo:

```
x<-1:12                  #cria uma seqüência de 1 a 12 no objeto x
xmat<-matrix(x,ncol=3)   #cria uma matriz de 3 colunas usando o objeto x
xmat                     #exibe a matriz criada
```

```
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Neste exemplo foi construída uma matriz de 3 colunas e 4 linhas usando os números de 1 a 12. Note que a matriz é preenchida ao longo das colunas. Para inverter este padrão deve-se adicionar o argumento `byrow=TRUE`, (que em inglês significa "por linhas") para dizer que a matriz deve ser preenchida por linhas:

```
matrix(x,ncol=3,byrow=TRUE)   #agora preenchendo a matriz pelas linhas
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

3.3.2. Obtendo informações sobre a matriz

Você pode verificar a dimensão de uma matriz com a função `dim()`:

```
x1<-matrix(1:12,ncol=4) #criando a matriz no objeto x1
dim(x1)                 #exibindo as dimensões da matriz x1
```

```
[1] 3 4
```

O valor retornado é um vetor com o número de linhas e colunas da matriz, nesta ordem. A função `summary()` opera em cada coluna da matriz como se fossem vetores:

```
summary(x1)
```

```
       X1              X2              X3              X4
 Min.   :1.0   Min.   :4.0   Min.   :7.0   Min.   :10.0
 1st Qu.:1.5   1st Qu.:4.5   1st Qu.:7.5   1st Qu.:10.5
 Median :2.0   Median :5.0   Median :8.0   Median :11.0
```

```
  Mean   :2.0   Mean   :5.0   Mean   :8.0   Mean   :11.0
  3rd Qu.:2.5   3rd Qu.:5.5   3rd Qu.:8.5   3rd Qu.:11.5
  Max.   :3.0   Max.   :6.0   Max.   :9.0   Max.   :12.0
```

Se você desejar um resumo de todos os elementos da matriz pode usar:

```
summary(as.numeric(x1))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.75    6.50    6.50    9.25   12.00
```

Alternativamente o comando `summary(as.vector(x1))` irá produzir o mesmo resultado.

3.3.3. Mais sobre como construir matrizes

Há outras funções que podem ser usadas para construir matrizes - `cbind` e `rbind` aumentam ou criam matrizes adicionando ("colando") colunas e linhas, respectivamente.

Exemplo:

```
x<-matrix(10:1,ncol=2)  #criando uma matriz qualquer
x
```

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

```
y<-cbind(x,1:5)         #observe que será adicionada uma 3ª coluna
y
```

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

```
y<-rbind(y,c(99,99,99))#adicionando uma nova linha...
y
```

```
      [,1] [,2] [,3]
 [1,]   10    5    1
 [2,]    9    4    2
 [3,]    8    3    3
 [4,]    7    2    4
 [5,]    6    1    5
 [6,]   99   99   99
```

Pode-se usar `cbind()` e `rbind()` também com a finalidade de "juntar" matrizes. Veja:

```
z<-cbind(y, rep(88,6), y)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
 [1,]   10    5    1   88   10    5    1
```

```
[2,]    9    4    2   88    9    4    2
[3,]    8    3    3   88    8    3    3
[4,]    7    2    4   88    7    2    4
[5,]    6    1    5   88    6    1    5
[6,]   99   99   99   88   99   99   99
```

3.3.4. Índices das matrizes

Da mesma forma que você pode extrair partes de vetores utilizando colchetes, podem ser extraídas partes de uma matriz. Porém isto é um pouquinho mais complicado, pois a matriz é um elemento que possui duas dimensões, enquanto vetores possuem apenas uma.

Para extrair um único elemento da matriz use colchetes com dois números separados por vírgula. O primeiro número indica o número da linha enquanto o segundo indica o número da coluna.

```
z[2,5]
```
```
[1] 9
```

Você pode extrair uma linha inteira ou uma coluna inteira usando apenas um número e a vírgula. Para extrair uma coluna coloque o número da coluna desejada depois da vírgula. Para extrair uma linha coloque o número da linha desejada depois da vírgula. Quando você extrai uma linha ou uma coluna o resultado é um vetor.

```
z[,4] #extraindo a quarta coluna
```
```
[1] 88 88 88 88 88 88
```
```
z[3,] #extraindo a terceira linha
```
```
[1]  8  3  3 88  8  3  3
```

Pode-se ainda extrair mais de uma linha ou coluna utilizando-se um vetor de índices. Neste caso o objeto resultante é uma matriz.

```
z[c(1,3,5),]      #extraindo três colunas
```
```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]   10    5    1   88   10    5    1
[2,]    8    3    3   88    8    3    3
[3,]    6    1    5   88    6    1    5
```
```
z[,5:7]                 #extraindo três colunas...
```
```
     [,1] [,2] [,3]
[1,]   10    5    1
[2,]    9    4    2
[3,]    8    3    3
[4,]    7    2    4
[5,]    6    1    5
[6,]   99   99   99
```

```
z[c(2,3),c(4,6)]       #tomando uma sub-matrix 2x2...
```

```
     [,1] [,2]
[1,]   88    4
[2,]   88    3
```

3.3.5. Mais sobre índices...

Uma coisa comum durante análises é querer selecionar todas as linhas de uma matriz que obedecem a certa condição definida pelas colunas. Geralmente dados são armazenados em matrizes onde cada linha corresponde à algum tipo de unidade enquanto que as colunas se referem a medidas tomadas destas unidades. Você pode querer selecionar as pessoas (linhas) que atendem a um certo critério (tamanho, peso etc).

Exemplo:

Vamos definir uma matriz onde as colunas armazenam: índices 1 a 5, idade e sexo (codificado com 0/1) de cinco pessoas.

```
pessoas<-cbind(c(1,2,3,4,5),c(43,55,52,23,46),c(1,1,0,1,1))
pessoas
```

Agora queremos extrair todas as pessoas que tem mais que 50 anos. Podemos fazer isto com um único comando como este:

```
idosos<-pessoas[pessoas[,2]>50,]
idosos
```

Note que este simples comando combina diversas operações de uma só vez. Podemos inspecionar parte a parte do comando, começando pela parte interna:

```
pessoas[,2]
```

```
[1] 43 55 52 23 46
```

Esta parte simplesmente selecionou a segunda coluna da matriz, que é um vetor.

```
pessoas[,2]>50
```

```
[1] FALSE  TRUE  TRUE FALSE FALSE
```

Agora o vetor selecionado foi comparado com o número 50 para verificar quais elementos do vetor eram maiores que este valor. O resultado foi um vetor lógico de TRUE e FALSE.

```
pessoas[pessoas[,2]>50, ]
```

```
     [,1] [,2] [,3]
[1,]    2   55    1
[2,]    3   52    0
```

Ao final foram selecionadas as linhas para as quais a condição (idade > 50) foi verdadeira.

3.4. Data.frame

Data.frames são muito parecidos com matrizes - eles têm linhas e colunas, e portanto duas dimensões. Entretanto, diferentemente de matrizes, cada coluna pode armazenar elementos de diferentes tipos. Por exemplo: a primeira coluna pode ser numérica enquanto a segunda pode ser constituída de caracteres.

Esses tipos de objetos são a melhor forma de se armazenar dados onde cada linha corresponde a uma unidade, indivíduo, ou pessoa, e cada coluna representa uma medida realizada em cada unidade, como dados provenientes de experimentos.

3.4.1. Lendo um data.frame de um arquivo texto

O R tem uma função que lê o conteúdo de um arquivo texto diretamente no formato de data.frame (Mais adiante esse assunto será mais bem detalhado). Um formato comum de dados é na forma de arquivo texto com uma linha para cada registro, com elementos separados por espaços ou vírgulas.

Considere o arquivo "musicas.txt":

CAD3004,Frank Black, Frank Black, 15, CD
Col4851,Weather Report, Sweetnighter, 6, CD
Rep2257,Neil Young, Decade I, 19, CD
Rep4335,Neil Young, Weld, 12, CD
Chp1432,Red Hot Chili Peppers, Mother's Milk, 13, Tape
EMI1233,Primus, The Brown Album, 12, Tape
Atl4500,Led Zeppelin, Led Zep 3, 11, CD

Usamos a função `read.table()` para ler estes dados e armazená-los em um objeto. Fornecemos o nome do arquivo (entre aspas) e o caractere de separação dos elementos (vírgula). O comando é:

OBS.: ***Neste comando é necessário informarmos o caminho completo do arquivo. Podemos usar a opção de menu suspenso "Arquivo – Mudar dir..." para alterarmos o diretório corrente para o onde se encontra o arquivo que desejamos importar. Neste caso, o comando pode citar apenas o nome do arquivo, sem a necessidade do caminho, como abaixo.***

```
musicas<-read.table("musicas.txt",sep=",",row.names=1,quote="")
musicas
```

```
                            V2               V3 V4    V5
CAD3004           Frank Black      Frank Black 15    CD
Col4851        Weather Report     Sweetnighter  6    CD
Rep2257            Neil Young        Decade I 19    CD
Rep4335            Neil Young             Weld 12    CD
Chp1432 Red Hot Chili Peppers    Mother's Milk 13  Tape
```

```
EMI1233                Primus  The Brown Album 12  Tape
Atl4500          Led Zeppelin        Led Zep 3 11    CD
```

Note que algumas colunas são numéricas enquanto outras são de caracteres (texto). Isto não é possível com matrizes, apenas com data.frame e listas.

OBS.: ***Se as colunas forem separadas por espaços ou tabulações no arquivo texto o argumento "sep" não é necessário.***

3.4.2. Índices como em matrizes e nomes como em listas...

O arquivo foi lido em algo que se parece um pouco com uma matriz. Podem-se usar índices para selecionar linhas e colunas da mesma forma que em matrizes:

```
musicas[,3]
```

```
 [1] 15  6 19 12 13 12 11
```

```
musicas[2,]
```

```
                    V2            V3 V4  V5
Col4851 Weather Report  Sweetnighter  6  CD
```

Os nomes das colunas podem ser definidos como em listas e as colunas são selecionadas usando o símbolo `$`:

```
musicas$V5
```

```
 [1]  CD    CD    CD    CD    Tape  Tape  CD
 Levels:  CD  Tape
```

Pode-se atribuir nomes às colunas usando a função `names()` associada a um vetor de nomes:

```
names(musicas)<-c('Artista','Nome','Nfaixas','Formato')
musicas$Nome
```

```
 [1]  Frank Black      Sweetnighter     Decade I         Weld
 [5]  Mother's Milk    The Brown Album  Led Zep 3
 7 Levels:  Decade I  Frank Black  Led Zep 3  Mother's Milk ...  Weld
```

Note que a primeira coluna, o numero de catálogo, não é parte do data.frame portanto não possui um valor em `names()`. Nomes de linhas podem ser definidos usando a função `row.names()`:

```
row.names(musicas)
```

```
 [1] "CAD3004" "Col4851" "Rep2257" "Rep4335" "Chp1432" "EMI1233"
 [7] "Atl4500"
```

3.4.3. Adicionando Colunas

Assim como em matrizes pode-se usar a função `cbind()` para se adicionar colunas a um data-frame. Se um nome for definido no argumento de `cbind()` este nome será associado à nova coluna. Veja:

```
musicas<-cbind(musicas,Faixa=c(7,6,9,10,9,8,8))
musicas
```

```
                       Artista             Nome Nfaixas Formato Faixa
CAD3004              Frank Black      Frank Black      15      CD     7
Col4851          Weather Report     Sweetnighter       6      CD     6
Rep2257              Neil Young          Decade I      19      CD     9
Rep4335              Neil Young             Weld      12      CD    10
Chp1432 Red Hot Chili Peppers    Mother's Milk      13    Tape     9
EMI1233                  Primus The Brown Album      12    Tape     8
Atl4500           Led Zeppelin        Led Zep 3      11      CD     8
```

3.5. Caracteres e Fatores

Toda coluna que a função `read.table()` encontrar que não for composta exclusivamente de números é definida como um fator. O usuário pode desejar que certas colunas sejam fatores enquanto outras não. No exemplo acima a coluna "Formato" é categórica e também "Artista", porém "Nome" provavelmente não é. As colunas podem ser convertidas de um formato para outro:

```
is.factor(musicas$Nome)
```

```
 [1] TRUE
```

```
musicas$Nome<-as.character(musicas$Nome)
musicas$Artista<-as.character(musicas$Artista)
```

3.6. Arrays

Arrays são objetos com propriedades semelhantes aos data.frames, porém são multidimensionais. São criados utilizando a função `array()`.

```
x<-1:18    #um vetor com de tamanho igual a 18.
A<-array(x,c(3,2,3))
```

```
, , 1

     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

, , 2
```

```
     [,1] [,2] [,3]
[1,]   10   13   16
[2,]   11   14   17
[3,]   12   15   18
```

```
A[,,1] # todos os valores da 1ª posição da terceira dimensão.
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

4. Entrando com dados

Diversos comandos podem ser usados, como os citados acima. Entre as funções que permitem importar dados podemos citar: `scan()`, `edit()`, `read.table()` e `data()`.

4.1. Uso da função scan()

Essa função permite o uso do *prompt* para entrada de dados em um vetor sem a separação por vírgulas, ou a edição de cada valor em uma linha distinta.

Exemplo:

Suponha que você deseja criar um vetor com uma coluna de valores que você já tenha digitado utilizando um editor de texto qualquer (um valor em cada linha). Então se pode usar os comandos "Copiar" e "Colar" oferecidos pelo sistema operacional.

Aí é só digitar `scan()` e depois "colar"...

```
teste<-scan()      #usado para entrada de dados
 1: 10
 2: 20
 3: 30
 4: 40
 5: 50
 6:
 Read 5 items
teste              #verifique que os valores agora estão armazenados
```

4.2. Uso da função edit()

É aberta uma planilha para digitação onde os dados podem ser armazenados como data.frame, alterações de funções etc.

Exemplo:

Utilizando o objeto chamado "*musicas*" suponha que você queira alterar o nome do artista o Formato da última linha de "CD" para "Vinil". Basta simplesmente usar a função `edit()`. Veja:

```
musicas2<-edit(musicas)
```

Para alterar a casa desejada basta clicar duas vezes e digitar o novo valor. Para sair basta clicar no "X".

4.3. Uso da função read.table()

Essa é uma ferramenta muito útil. O R pode ler arquivos de texto (ASCII) e também em outros formatos (Excel, SAS, SPSS, etc), e até mesmo acessar bancos de dados SQL. Porém as funções necessárias à realização de algumas dessas operações não se encontram na biblioteca BASE. Por esse motivo, nos restringiremos apenas ao primeiro caso: o de códigos ASCII (arquivos de texto).

A função é usada da seguinte maneira:

```
read.table("endereço completo do arquivo de dados",h=T)
```

OBS.: ***"h=T" é necessário se a primeira linha do arquivo de dados contém as informações sobre o nome das colunas (linha de cabeçalho). Caso contrário, escrever h=F; Outra observação importante é que as barras no endereço devem estar nesse sentido: " / ", mesmo quando o sistema operacional em questão é o Windows.***

4.3.1. Lendo dados de um arquivo ASCII

Exemplo:

Deseja-se ler os dados do arquivo "coord.dad" que contém as coordenadas espaciais de 10 pontos. As colunas estão separadas um espaço em branco. Veja o arquivo:

x y z
1 2 9
2 2 8
3 5 9
4 5 10
5 8 7
6 8 6
7 11 5
8 11 5
9 14 3
10 14 3

Observe que a primeira linha é de cabeçalho. Se este arquivo estiver salvo no disco flexível , basta digitar:

```
pontos<-read.table("A:/coord.dad",#caminho completo do arquivo
                   sep="",          #caractere de separação
                   h=T)             #primeira linha é cabeçalho
```

OBS.: ***Para ler arquivos de dados contidos no R utiliza-se da função*** **`data()`**.

5. Operações com vetores e matrizes

Vamos usar um grande exemplo para mostrar algumas operações possíveis com matrizes e vetores.

Exemplo:

Começaremos criando vetores que comporão as colunas de uma matriz. Veja:

```
coluna1<-c(2,1,0)
coluna2<-c(1,3,1)
coluna3<-c(1,1,2)
```

Produto de vetores:

```
coluna1%*%coluna1       #uma multiplicação da coluna 1 por ela mesma
```

```
      [,1]
[1,]    5
```

Observe que, se quiser a soma dos quadrados dos valores no vetor, então faremos coluna1*coluna1, daí a importância do uso do símbolo "%" antes e depois do asterisco. Veja:

```
coluna1*coluna1
```

```
[1] 4 1 0
```

Agora criando uma matriz de nome A com os vetores já criados. Observe que existem outras formas de criar matrizes.

```
A<-cbind(coluna1,coluna2,coluna3)
A
```

```
     coluna1 coluna2 coluna3
[1,]       2       1       1
[2,]       1       3       1
[3,]       0       1       2
```

Obtendo a transposta de A

```
t(A)
```

```
         [,1] [,2] [,3]
coluna1    2    1    0
coluna2    1    3    1
coluna3    1    1    2
```

Agora, apenas a título de exemplo, vamos fazer alguns cálculos.

```
t(A)%*%A    #A'A (A transposta "vezes" A)
```

```
         coluna1 coluna2 coluna3
coluna1        5       5       3
coluna2        5      11       6
coluna3        3       6       6
```

Finalmente, a inversa de uma matriz não singular quadrada:

```
solve(A)    #inversa da matriz A
```

```
                [,1]       [,2]       [,3]
  coluna1  0.5555556 -0.1111111 -0.2222222
  coluna2 -0.2222222  0.4444444 -0.1111111
  coluna3  0.1111111 -0.2222222  0.5555556
```

Caso ainda tenha dúvidas, você pode testar a veracidade da operação anterior usando um algebrismo simples. Multiplique a inversa pela matriz "normal" e o resultado deve ser uma matriz identidade com as mesmas dimensões da matriz a qual se deseja testar. Veja:

```
solve(A)%*%A      #verificação
```

```
          coluna1       coluna2 coluna3
  coluna1       1 -2.775558e-17       0
  coluna2       0  1.000000e+00       0
  coluna3       0  0.000000e+00       1
```

Para fazermos arredondamentos com 5 casas decimais, por exemplo, podemos escrever:

```
round(solve(A)%*%A)
```

```
          coluna1 coluna2 coluna3
  coluna1       1       0       0
  coluna2       0       1       0
  coluna3       0       0       1
```

Outro exemplo: inversa da matriz A'A:

```
solve(t(A)%*%A)
```

```
             coluna1    coluna2     coluna3
 coluna1  0.37037037 -0.1481481 -0.03703704
 coluna2 -0.14814815  0.2592593 -0.18518519
 coluna3 -0.03703704 -0.1851852  0.37037037
```

5.1. Algumas funções disponíveis

A tabela abaixo dá algumas operações lógicas e outras funções disponíveis no R.

Símbolo	Significado
`!=`	Diferente
`%%`	Modulo
`%/%`	Divisão inteira
`+, - , *, /`	Adição, subtração, multiplicação, divisão
`** ou ^`	Potência
`<, >`	Menor, maior que
`<=, >=`	Menor ou igual, maior ou igual que
`==`	Igual
`max(), min(), range()`	Maximo, mínimo e amplitude
`sum(x)`	Soma total de x

`mean(x), var(x)`	Média aritmética, variância amostral de x
`cor(x,y)`	Correlação entre os vetores x e y
`median(x)`	Mediana de x
`order(x)`	Vetor contendo as posições ordenadas crescentes de x
`sort(x)`	Versão ordenada de x
`rank(x)`	Retorna vetor com a classificação crescente de x
`ColSums(A)`	Retorna a soma das colunas da matriz A

5.1.1. Usando alguns tipos de combinação de operações:

Por exemplo, vamos criar um vetor com uma seqüência de 0 a 10:

```
x<-0:10
sum(x)       #soma desta seqüência
 [1] 55
sum(x<5)     #uma forma de descobrir quantos valores são menores que 5
 [1] 5
sum(x[x<5]) #soma dos valores menores que 5
 [1] 10
x<5   #obtendo a reposta lógica de x<5
 [1] T T T T T F F F F F
```

Você pode imaginar se os valores falsos fossem representados pelo zero (0) e os valores verdadeiros pelo valor um (1). Multiplicando valores lógicos por valores numéricos obtem-se valores numéricos.

```
1*(x<5)
 [1] 1 1 1 1 1 0 0 0 0 0
```

Agora imagine a multiplicação dos valores do vetor x pelos valores lógicos do valor x na condição discriminada:

```
x*(x<5)
 [1] 0 1 2 3 4 5 0 0 0 0 0
```

Quando aplicamos a função sum(x), temos a respostas da soma dos valores dos números 0+1+2+3+4+5 = 10.

```
sum(x*(x<5))
 [1] 10
```

Agora você percebe a diferença entre utilizar parênteses ou colchetes.

Para saber qual é a posição de um valor ou de determinados valores pode-se utilizar:

```
which(x<=5)
 [1] 1 2 3 4 5 6
```

Além destas opções pode-se contar com os operadores de conjuntos como: `intersect(x,y)`, `union(x,y)`, `setdiff(x,y)`, `setequal(x,y)`, `unique(x,y)`. Para saber mais consulte a ajuda do R.

6. Salvar ou ler arquivos *.R

Para salvar o arquivo de trabalho, deve-se clicar na barra de ferramentas em:

File -> Save Workspace... -> escolher o local para salvar e nomear o arquivo, ou, na linha de comando (*prompt*) do *R Console*, digitar o seguinte comando: `save.image("CAMINHO/nome_do_arquivo.RData")`, onde CAMINHO é o local (ou pasta) de destino, onde será salvo o arquivo.

Para ler um arquivo deve-se:

File -> Load Workspace... -> escolher o local e o arquivo desejado. Ou utilizar o comando `load("CAMINHO/nome_do_arquivo")` no *prompt* de comandos do R.

A utilização desta operação, quando se trabalha com arquivos no R, é a possibilidade de ter armazenado todos os objetos criados em análises facilitando assim o trabalho de revisão.

7. Criando funções

7.1. Entendendo o conceito de função

Alguns usuários consideram como uma das maiores vantagens do R é a facilidade para criação de novas funções, mas deve-se destacar que já existem diversas funções já disponíveis. Agora vamos criar algumas funções simples.

Para criar uma função é necessário realizar a assignação da seguinte forma:

```
nome_da_funcao<-function(argumento1, argumento2, ..., argumento n)
```

Sendo o uso da função dado por:

```
nome_da_funcao(argumento1, argumento2, ..., argumento n)
```

Exemplo:

Vamos começar com um exemplo muito simples de como criar e usar uma função no R. Uma função pode ser criada para executar diversas operações de modo automático e é programada (criada) com procedimentos algorítmicos (passos). Porém uma ela pode ser usada também para executar uma tarefa simples, como mostraremos. O problema é que não é usual programar uma função que execute uma linha de comando apenas.

Vamos criar uma função que calcule a média de um conjunto de dados. Veja:

```
media<-function(dados)  #onde dados será o parâmetro de entrada
   {                    #o "{" indica o início (begin)
   print(sum(dados)/length(dados)) #será exibido na tela a média de
"dados"
   }                    #já o "}" representa o fim (end)
```

Neste momento a função está pronta e carregada na memória. Agora vamos criar um conjunto de dados e calcular a média deste usando a função que acabou de ser criada.

```
x<-1:10                 #criando um conjunto de dados qualquer
media(x)                #o parâmetro não precisa chamar "dados"
 [1] 5.5

mean(x)                       #apenas pra conferir!
 [1] 5.5
```

Usualmente usamos um editor de textos, pela comodidade e recursos que este apresenta na edição de funções e rotinas para a análise de dados no R. Usufruindo do já conhecido "Copiar e Colar" de modo que torna-se mais simples consertar possíveis falhas de programação (quando essas acontecem devido a um erro devido a montagem do algoritmo).

Se você desejar apenas ver o conteúdo de uma função já criada basta escrever seu nome no *prompt* de comando:

```
media
```

```
function(dados)#onde dados será o parâmetro de entrada
   {#o "{" indica o início (begin)
   print(sum(dados)/length(dados))#será exibido na tela a média de
"dados"
   }#já o "}" representa o fim (end)
```

7.2. Execuções condicionais

A linguagem R dispõe de ordem condicionais da seguinte forma:

if (condição) "expres_1" else expres_2

Onde o resultado da condição deve ser uma valor lógico (T-TRUE ou F-FALSE), e se este for verdadeiro (T) será executada a expres_1, caso contrário será executada a expres_2. O uso do condicional "else" não é obrigatório.

Operadores lógicos & (AND ou E) e | (OR ou OU) podem ser usados como condições de uma expressão *if*.

```
x<-6 ; y<-4   # criando alguns dados
if(x == 6 & y>3) print("Verdadeiro") else print("Falso")
```

```
[1] "Verdadeiro"
```

Existe uma versão vetorizada da construção condicional *if /then*, que é a função *ifelse*, cuja sintaxe no R é: ifelse(condição, expres_1 , expres_2), e cujo resultado é um vetor. A expres_1 será aplicado caso a condição for verdadeira e a expres_2 caso contrário.

```
x<-1:6   # criando o vetor a ser testado
ifelse(x>=5,x+1,x-2)       # condição a ser testada e comando a ser
executado.
```

```
[1] -1  0  1  2  6  7
```

7.3. Funções envolvendo ciclos

Ciclos são processos iterativos no qual sua função irá executar uma seqüência de comandos até uma condição previamente estabelecida. É importante nestes casos, que as iterações tenham uma condição finita.

Pode-se utilizar comandos como `while(condição)` ou `for(condição)`. Para o caso especifico do comando "for" faremos uso de um contador como será mostrado no exemplo abaixo. O termo (i in 1:10) que quer dizer que o contador 'i' irá de 1 a 10 a cada unidade (estes operadores trabalham com uma unidade dos números naturais)

Exemplo:

Esta função é capaz de calcular diversas medias e variâncias:

```
med_var<-function(...)
   {
   data<-list(...)
   n<-length(data)
   means<-numeric(n)
   vars<-numeric(n)
   for (i in 1:n)
      {
      means[i]<-mean(data[[i]])
      vars[i]<-var(data[[i]])
      }
   print(means)
   print(vars)
   }
```

Invocando a função:

```
x<-rnorm(100)     #gera números pseudo-aleatórios da distrib. normal
y<-rnorm(200)
z<-rnorm(300)
med_var(x,y,z)
```

```
 [1] -0.14678761 -0.10985526 -0.09748565
 [1] 0.9921148 0.8865427 0.8585223
```

Outro Exemplo:

Criando uma função para resolver o seguinte problema de probabilidade, através de simulação.

Considere quatro minúsculos insetos semelhantes colocados em uma caixa. Considere que estes insetos só se diferenciam quanto ao sexo. Dois são machos e dois são fêmeas. Dois insetos são selecionados ao acaso. Qual a probabilidade de que ambos os insetos sejam do mesmo sexo, quando ambos são tomados simultaneamente? Resposta: 1/3

```
simula<-function(n)
   {
   #Quatro insetos na caixa
   #entrar com os nº de iterações
   n<-n                              #aqui é o número dos insetos
   caixa<-c("m","m","f","f")
   res<-integer(n)                   #cria um vetor de tamanho n
   for (h in 1:n)
      {
      cx<-sample(caixa)              #tomadas ao acaso sem reposição
      if (cx[1]==cx[2]) {res[h]<-1} else {res[h]<-0}
      }
   prob<-mean(res)
   return(prob)
   }
```

Agora vamos usar a função criada para fazer 10 iterações

```
simula(10)          #fazendo 10 iterações
 [1] 0.1            #lembre-se que este é um pequeno número de amostras
```

E observe que quanto maior o número de iterações, valor da probabilidade se aproxima do esperado: 1/3

```
simula(1000)        #veja o valor da probabilidade para 1000 simulações
 [1] 0.3325
```

7.3.1. Função para plotar símbolos

```
simbolos<-function()
   {
   k<- 0
   plot(c(0,1),c(0,1),lab=c(0,0,0),xlab="",ylab="",type="n")
   for(i in 1:9)
      {
      for(j in 1:9)
         {
            points(j/10,i/10,pch=k,col=i)
k<- k+1
         }
      }
   }
simbolos()
```

OBS.: ***Essa função apresenta um procedimento não implementado no R, e assim ela retorna um aviso "Warning mnssage". Todavia ela executa a tarefa requerida.***

8. Estatística descritiva

A Estatística Descritiva tem se difundido bastante com o advento dos computadores. Com crescente aumento da capacidade de realização de grandes volumes de cálculos em pequenos intervalos de tempo, tornou-se, quase que obrigatório, uma análise descritiva dos dados, seja para efeito de apresentação de informações ou até mesmo para uso preliminar dessas informações para fins de análises (estatísticas) futuras.

Neste tópico, buscamos mostrar algo relacionado à medidas de posição e dispersão, assim como dar uma pequena noção da facilidade de se obter esses resultados usando o R.

A parte de apresentação gráfica, como por exemplo, a criação de histogramas e ogivas, será vista posteriormente no capítulo *"Criando gráficos como o R"*

8.1. Algumas notações

8.1.1. Somatório

Na verdade o somatório nada mais é do que uma notação simplificada de várias somas. Mesmo com sua simplicidade, ele também tem seu espaço no R. Veja o exemplo abaixo:

```
x<-c(1,2,3,4)      #criando um vetor qualquer
sum(x)             #obtendo o somatório do vetor criado
```

```
[1] 10
```

Resolvendo com o R...

Considere as variáveis X e Y que representam, respectivamente, as notas de duas disciplinas, para um grupo de 6 alunos.

X = {90, 95, 97, 98, 100, 60}
Y = {60, 70, 80, 60, 90, 75}
Encontre:

a) $\sum_{i=1}^{6} X_i$

```
X<-c(90,95,97,98,100,60)      #criando o vetor X
sum(X)                        #calculando o somatório
```

```
[1] 540
```

b) $\sum_{\substack{i=2 \\ i\neq 5}}^{6} Y_i^2$

```
Y<-c(60,70,80,60,90,75) #criando o vetor Y
sum(Y^2)-Y[1]^2-Y[5]^2  #somatório subtraindo os termos da exceção
```

```
[1] 20525
```

8.1.2. Produtório

Pode-se fazer uma analogia com o somatório, ressaltando que este referencia-se à multiplicação.

```
x<-c(1,2,3,4)      #criando um vetor qualquer
prod(x)            #obtendo o produtório do vetor criado
```

```
[1] 24
```

Resolvendo com o R...

Dado:
X = {32 , 12 , 45 , 9 , 78 , 16 , 54 , 14}
Encontre:

$$\prod_{i=1}^{7} X_i$$

```
X<-c(32,12,45,9,78,16,54,14)  #criando o vetor X
prod(X)/X[8]                  #produtório, retirando a exceção
```

```
[1] 10480803840
```

8.2. Medidas de posição amostral

Dentre as várias medidas de dispersão, nos preocuparemos apenas com as medidas de tendência central.

8.2.1. Média

A média é a medida de posição mais conhecida e pode ser obtida facilmente no R através do comando `mean()`. Veja:

```
x<-c(1,2,3,4,5)    #criando um vetor
mean(x)            #obtendo a média
```

```
[1] 3
```

8.2.2. Mediana

A mediana é uma medida de posição (tendência central) indicada quando o conjunto de dados possui valores extremos.

É válido lembrar que a mediana é obtida do conjunto de dados quando este se encontra ordenado, não importando se crescente ou decrescentemente. Porém o R já leva

em conta a ordenação, sem a necessidade de o usuário ordenar os dados antes de executar o comando que dá a mediana. Veja:

```
x<-c(1,2,18,7,6)  #vetor qualquer não ordenado
median(x)         #obtendo mediana
```

```
[1] 6
```

8.2.3. Moda

A moda é o valor mais freqüente do conjunto de dados. Um conjunto de dados pode ser *unimodal*, quando este possui apenas um valor modal, *bimodal*, quando possui dois valores de moda e *multimidal*, para conjunto de dados com mais de dois valores modais.

Aqui vai uma função desenvolvida para calcular o valor modal (ou os valores, quando o conjunto de dados tiver mais que um valor para a moda). Ela foi desenvolvida apenas para conjuntos de dados agrupados em vetores, matrizes ou fatores. Veja:

```
moda<-function(d)
   {
   if ((is.vector(d) || is.matrix(d) || is.factor(d)==TRUE) &&
      (is.list(d)==FALSE))
      {
      dd<-table(d)
      valores<-which(dd==max(dd))
      vmodal<-0
      for(i in 1:(length(valores)))
         if (i==1) vmodal<-as.numeric(names(valores[i]))
            else
            vmodal<-c(vmodal,as.numeric(names(valores[i])))
      if (length(vmodal)==length(dd))
         print("conjunto sem valor modal")
            else return(vmodal)
      }
      else print("o parâmetro deve ser um vetor ou uma matriz")
   }

x<-c(1,2,3,4,5,5,5,5,5,6,6,7,7,8)
moda(x)
```

```
[1] 5
```

Resolvendo com o R...

Dado o conjunto de dados abaixo, encontre a média, a mediana e a moda dos dados:

20 7 5 9 6 21 24 10 12 22 21 16 13 6 6 2 19 3 10 7 2 18 4 6 18 12 4 13 9 3

```
x<-scan() #use copiar e colar, para criar o conjunto de dados

mean(x)     #obtendo o valor da média
```

```
[1] 10.93333
```

```
median(x)   #observe que não é necessário ordenar
```

```
 [1] 9.5
moda(x)      #após introduzir a função de cálculo de moda no R
 [1] 6
```

8.3. Medidas de dispersão amostral

As medidas de dispersão também constituem elementos fundamentais na caracterização de um conjunto de dados. Algumas delas podem ser citadas abaixo.

8.3.1. Variância

Com apenas um comando podemos obter a variância usando o R. Veja o exemplo abaixo:

```
x<-c(1,2,3,4,5)   #criando um vetor
var(x)            #obtendo a variância
 [1] 2.5
```

8.3.2. Desvio padrão

Pode ser obtido por:

```
x<-c(1,2,3,4,5)   #um vetor qualquer
sd(x)             #obtendo o desvio padrão
 [1] 1.581139
```

Ou por:

```
sqrt(var(x))      #definições...
  [1] 1.581139
```

8.3.3. Amplitude total

A amplitude total pode ser obtida de uma forma indireta, subtraindo-se o máximo valor do conjunto de dados pelo mínimo deste. Veja o exemplo:

```
x<-c(2,4,5,6,10)  #um conjunto de dados qualquer
range(x)          #mostra o min(x) e o max(x)
 [1]  2 10
max(x)-min(x)     #amplitude total obtida de forma indireta
 [1] 8
```

Resolvendo com o R...

Um psicólogo deseja obter informações sobre o grau de dispersão de dados referentes à idade dos freqüentadores de um grupo de Alcoólicos Anônimos. Ele coletou os seguintes dados:

33 17 39 78 29 32 54 22 38 18

Ele quer saber a variância, o desvio padrão, amplitude total, erro padrão da média, e coeficiente de variação de seu conjunto de dados.

```
x<-c(33,17,39,78,29,32,54,22,38,18) #conjunto de dados

var(x)                  #variância
 [1] 339.5556
sd(x)                   #desvio padrão
 [1] 18.42703
max(x)-min(x)           #amplitude total
 [1] 61
sd(x)/sqrt(length(x))   #erro padrão da média
 [1] 5.82714
sd(x)/mean(x)*100       #coeficiente de variação em %
 [1] 51.1862
```

8.4. Covariância e Correlação

A covariância e a correlação entre dois conjuntos de dados quaisquer podem ser obtidos pelos comandos `cov(x,y)` e `cor(x,y)`, respectivamente. Veja o exemplo:

```
x<-c(1,2,3,4,5)   #criando um vetor qualquer
y<-c(6,7,8,9,10)  #criando outro vetor
cov(x,y)          #obtendo a covariância entre x e y
 [1] 2.5
cor(x,y)          #obtendo a correlação
 [1] 1
```

9. Sobre probabilidade

9.1. Algumas Distribuições

Diversas situações reais muitas vezes se aproximam de certas distribuições estocásticas definidas por algumas hipóteses. Daí a importância de se conhecer e manipular algumas destas distribuições tão presentes em nosso cotidiano.

Veja a lista abaixo com algumas funções para gerar valores amostrais de distribuições e seus respectivos parâmetros. Nos detalhes de cada função, os valores indicados (como por exemplo: `mean=0`, na distribuição normal) são os possíveis por definição (*default)*, mas podem ser alterados pelo usuário ao seu bel prazer, já os que não estão indicados, significa que o parâmetro deve ser especificado pelo usuário.

Distribuição/função	Função
beta	`rbeta(n,shape1,shape2)`
binomial	`rbinom(n,size,prob)`
binomial negativa	`rnbinom(n,size,prob)`
Cauchy	`rcauchy(n,location=0,scale=1)`
estatística de Wilcoxon's	`rwilcox(nn,m,n,n),rsignrank(nn,n)`
exponencial	`rexp(n,rate=1)`
Fisher-Snedecor (F)	`rf(n,df1,df2)`
gamma	`rgamma(n,shape,scale=1)`
Gauss (normal)	`rnorm(n,mean=0,sd=1)`
geométrica	`rgeom(n,prob)`
hypergeométrica	`rhyper(nn,m,n,k)`
logística	`rlogis(n,location=0,scale=1)`
log-normal	`rlnorm(n,meanlog=0,sdlog=1)`
Poisson	`rpois(n,lambda)`
qui-quadrado (χ^2)	`rchisq(n,df)`
'Student' (t)	`rt(n,df)`
uniforme	`runif(n,min=0,max=1)`
Weibull	`rweibull(n,shape,scale=1)`

Adicionalmente, outras letras (p, d, q, além do r) adicionadas previamente ao código das distribuições podem ser usadas, com diferentes propósitos. Resumidamente temos:

r: Gerador de números aleatórios. Requer argumentos especificando o tamanho da amostra, além dos parâmetros requeridos pela distribuição de interesse;

p: Função de probabilidade. Requer um vetor de percentis, além dos parâmetros requeridos pela distribuição de interesse;

d: Função densidade. Requer um vetor de percentis, além dos parâmetros requeridos pela distribuição de interesse;

q: Função de percentis. Requer um vetor de probabilidades ($0 < p < 1$), além dos parâmetros requeridos pela distribuição de interesse.

Exemplos:

A probabilidade de ocorrência de um valor menor que 20 em uma distribuição normal de média 50 e desvio padrão igual a 15 pode ser obtida com o código abaixo:

```
pnorm(20,   #o valor referência para o cálculo da probabilidade
      50,   #o segundo parâmetro se refere a media
      15)   #e por último o valor do desvio padrão
```

```
 [1] 0.02275013
```

Experimente agora tentar encontrar o valor da probabilidade de ocorrência de valores menores ou iguais ao valor da média. Qual seria o resultado esperado?

```
pnorm(50,50,15)
```

```
 [1] 0.5
```

Verifique também o que acontece quando se altera o valor do desvio padrão o caso acima.

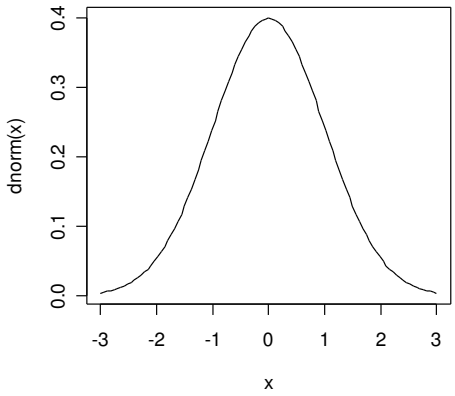
Agora, se você deseja computar o percentil 0.96 de uma distribuição de Qui-quadrado com 5 graus de liberdade use:

```
qchisq(0.96,5)
```

```
 [1] 11.64433
```

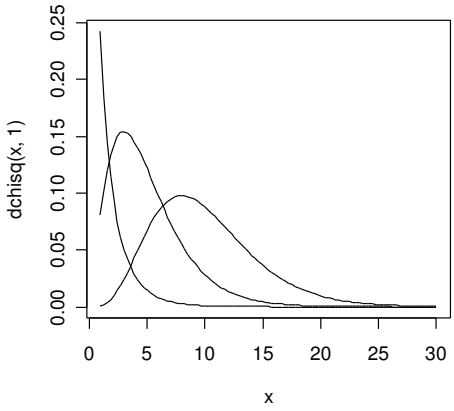
A letra d antes dos códigos poderá ser usada, de maneira muito interessante como será visto adiante, para fazer o gráfico da distribuição de uma variável aleatória contínua (função densidade de probabilidade, ou como é mais conhecida: f.d.p.). Por exemplo, para desenhar a curva de uma distribuição normal padrão no intervalo [-3,3] faça o seguinte:

```
curve(dnorm(x),-3,3)    #desenha uma curva de distrib normal em [-3,3]
```



Usando essa função poderíamos comparar diferentes formas da distribuição de certas variáveis aleatórias quando os valores de seus parâmetros são alterados. O exemplo abaixo mostra a distribuição de qui-quadrado quando seus valores de graus de liberdade alternam entre 1, 5 e 10. Confira:

```
curve(dchisq(x,1),1,30)              #qui-quadrado: 1 grau de liberdade
curve(dchisq(x,5),1,30,add=T)        #agora com 5 graus de liberdade
curve(dchisq(x,10),1,30,add=T)       #e por último 10 graus de liberdade
```



Se a variável for discreta, devemos substituir a função "`curve`" por "`points`" (o comando `plot()` também funciona). Veja em `?points` como usar esse comando. Nesse caso é necessário usar o argumento `type="h"` para desenhar linhas verticais sobre os valores de x. Veja o exemplo no tópico "Binomial" abaixo.

9.1.1. Binomial

A distribuição Binomial advém da distribuição de Bernoulli quando repetimos um esnsaio (algumas vezes referido como "provas") de Bernoulli "n" vezes. Onde p é a probabilidade de sucesso. Veja:

Resolvendo com o R...

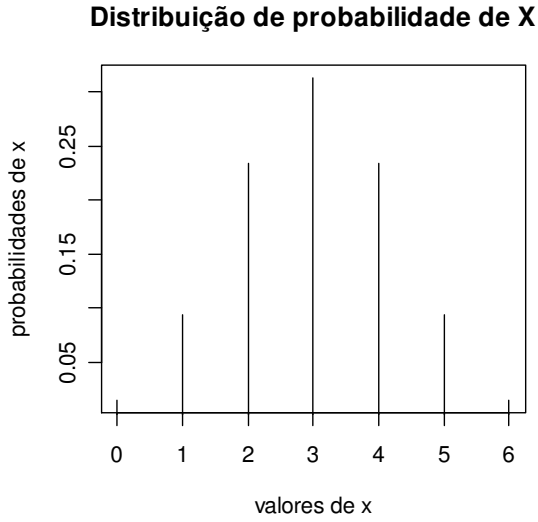
Considere que a probabilidade de certa peça artesanal ser produzida com perfeição pelo artesão igual a 0,5. Considere que o artesão produza 6 peças por vez. Pede-se:

a) Obter a distribuição de probabilidades do número peças perfeitas produzidas pelo artesão (em uma vez – 6 peças);

```
bino<-dbinom(0:6,6,.5)  #obtendo a tabela
bino
```

```
 [1] 0.015625 0.093750 0.234375 0.312500 0.234375 0.093750 0.015625
```

```
plot(0:6,                       #intervalo desejado
     bino,                      #vetor com os valores de probabilidade
     type="h",                  #adiciona um traça do eixo ao ponto
     xlab='valores de x',       #texto do eixo x
     ylab='probabilidades de x',#texto do eixo y
     main='Distribuição de probabilidade de X')#título
```



9.1.2. Poisson

A distribuição de Poisson é utilizada quando não é prático ou mesmo possível determinarmos o número de fracassos ou o número total de provas de um experimento. É muito útil para descrever as probabilidades do número de ocorrências num campo ou intervalo contínuo (em geral de tempo ou espaço). Veja o exemplo abaixo:

Resolvendo com o R...

Num trabalho de campo realizado por um topógrafo há, em média, 4 erros grosseiros por Km^2 levantado. Pergunta-se:

a) Qual a probabilidade de que um Km^2 contenha pelo menos 1 erro grosseiro?

```
dpois(0,4)
```

```
 [1] 0.01831564
```

b) Estime o número provável de Km^2 que não contêm erros numa área de 100 Km^2.

```
dpois(0,4)*100
```

```
 [1] 1.831564
```

9.1.3. Normal

Sem dúvida a mais popular das distribuições de probabilidade tem algumas particularidades que a tornam especial. A distribuição normal permite a realização de vários procedimentos estatísticos que não são possíveis em outras distribuições como o teste t de Student entre outros. Veja alguns exemplos envolvendo essa distribuição:

Resolvendo com o R...

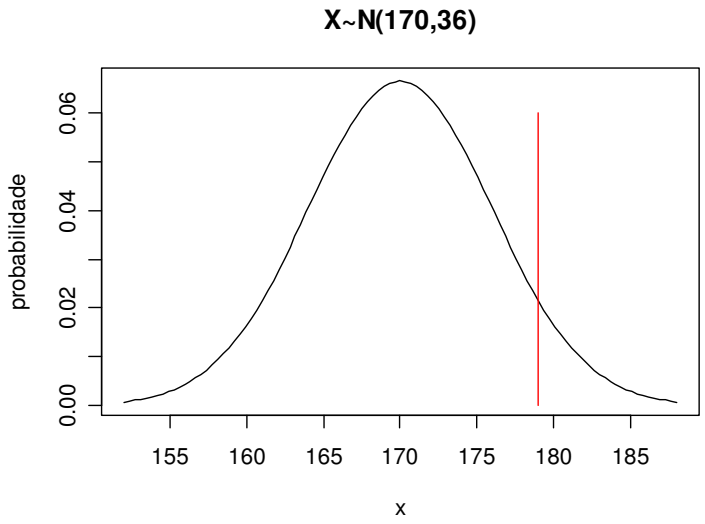
Suponha que um pesquisador coletou dados de estatura de jovens em idade de alistamento militar. Sabendo-se que a estatura de um acerta população segue a distribuição normal o pesquisador pode escrever que X~N(170;36), onde X é a variável aleatória altura com unidades em centímetros. Pede-se:

a) Qual a probabilidade de encontrarmos um jovem com mais de 1,79 metros de altura?

```
1-pnorm(179,170,6)
```

```
 [1] 0.0668072
```

```
curve(dnorm(x,170,6),          #distr normal: media=170 e desv.padrão=6
      152,188,                 #limites inferior e superior do gráfico
      main="X~N(170,36)",      #título do gráfico
      ylab="probabilidade")    #texto do eixo y
lines(c(182,182),              #início e fim da linha em rel ao eixo x
      c(0,0.06),               #início e fim da linha em rel ao eixo y
      col=2)                   #cor da linha: vermelha
```



O valor de probabilidade encontrada corresponde exatamente a área do gráfico abaixo da curva normal e à direita da linha vermelha.

b) Encontre o valor da estatura para qual a probabilidade de encontrarmos valores menores que o deste seja de 80%.

```
qnorm(0.8,170,6)
 [1] 175.0497
```

9.2. Geração de números aleatórios

O R pode gerar números aleatórios de várias formas. Pode-se gerar um número qualquer, dentro de um intervalo pré-estabelecido ou em uma distribuição de interesse.

Veremos abaixo as duas formas:

9.2.1. Gerar números em intervalos pré-estabelecidos

Deve-se primeiro estabelecer o intervalo, ou seja, quais valores que o(s) número(s) gerado(s) pode(m) assumir. Depois se devem determinar quantos números serão gerados, com ou sem reposição. Veja o exemplo a seguir:

Exemplo:

Para simular o lançamento de um dado honesto 100 vezes usando o R, podemos usar o comando `sample()`, onde o primeiro parâmetro do parêntese informa quais valores podem ser assumidos (no nosso caso, os 6 valores contidos nas faces do dado), em seguida informamos quantas vezes queremos "jogar o dado", e devemos dizer também ao R que os números podem se repetir, ou seja, "com reposição `(re=TRUE)`".

```
x<-c(1,2,3,4,5,6) #determinado quais valores podem ser assumidos
sample(x,         #aqui mandamos sortear dentre os valores de x
      100,        #aqui é o tamanho da amostra
      re=TRUE)    #re abrevia "replace" do inglês, indicando reposição
  [1] 4 2 4 2 6 1 3 5 5 5 2 4 6 3 6 6 6 3 4 6 4 6 4 3 6 5 4
 [28] 6 3 4 5 3 2 3 4 4 5 3 2 4 3 2 3 1 4 4 1 6 1 6 1 2 4 5
 [55] 6 4 5 4 5 3 5 6 6 3 6 4 3 1 6 6 1 1 3 5 5 5 6 6 5 3 6
 [82] 4 4 5 1 2 5 2 5 5 3 3 1 5 2 5 4 1 1 3
```

9.2.2. Gerar números de uma distribuição de interesse

No R existem várias funções (distribuições) prontas para gerar números aleatórios. Basta usar a seguinte codificação: letra "r", seguido do código da distribuição de interesse e seus parâmetros. Veja o exemplo abaixo:

```
runif(1)    #gera um número aleatório de uma distribuição uniforme.
Nessa distribuição o único parâmetro exigido é o tamanho da amostra
```

```
 [1] 0.8318596
```

```
#outro exemplo, distribuição normal:
# tamanho da amostra é 10; média 30; desvio padrão 5
rnorm(10,30,5)
```

```
 [1] 22.66061 32.23040 20.74191 24.96324 36.26748 35.04722
 [7] 33.33221 27.13449 28.04094 25.90274
```

10. Criando gráficos com o R

O R é uma poderosa ferramenta no que diz respeito à confecção de gráficos e afins. Na estatística, em especial, ele possibilita a criação de histogramas, ogivas, curvas de distribuições e regressões e muito mais.

Em análises estatísticas, podemos usufruir das ferramentas gráficas para fazermos um estudo inicial dos nossos dados. Muitas vezes torna muito mais simples o entendimento de um problema ou situação se conseguimos visualizarmos as variáveis envolvidas graficamente.

Você pode ver alguns exemplos de gráficos que podem ser criados no R com o comando abaixo:

```
demo(graphics)
```

O R possui diferentes funções geradoras de gráficos, e essas são classificados como:

Funções gráficas de alto nível: criam novos gráficos na janela, definindo eixos, título, etc. Exemplos: `plot, hist, image, contour, persp` etc.

Funções gráficas de baixo nível: permitem adicionar novas informações em gráficos já criados, como novos dados, linhas etc. Exemplos: `points, lines, abline, polygon, legend` etc.

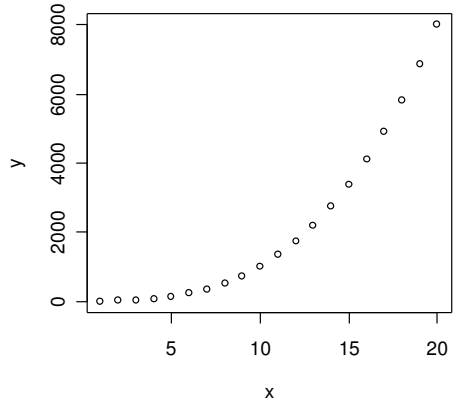
Funções gráficas iterativas: permitem retirar ou adicionar informações aos gráficos já existentes, usando por exemplo o cursor do mouse. Exemplos: `locator, identify`.

10.1. Uso da função plot()

10.1.1. Um gráfico simples

A função `plot()` inicia um novo gráfico. Em sua forma mais simples a função recebe valores de coordenadas x e y:

```
x<-1:20
y<-x**3
plot(x,y)   #plota as variáveis x e y.
```

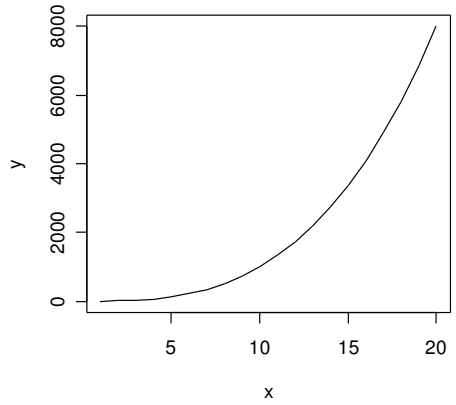


Este último comando faz com que o R abra uma nova janela. Novos gráficos irão sobrescrever o gráfico na mesma janela.

Todas as alterações geradas dentro de uma função gráfica são consideradas alterações temporárias, veja algumas opções abaixo.

Gráficos com linhas ligando os pontos podem ser obtidos utilizando o argumento opcional `type="l"` (letra L minúsculo) na função `plot()`:

```
plot(x,y,type="l")
```



Há várias outras opções para os gráficos. Examine estes exemplos:

```
plot(x,y,type="b")
plot(x,y,type="o")
plot(x,y,type="s")
plot(x,y,type="c")
plot(x,y,type="h")
```

Se quiser obter mais informações sobre o comando `plot()` digite `?plot` no *prompt* do *R Console*.

10.1.2. Adicionando mais dados a um gráfico

Podem-se adicionar pontos ou linhas a um gráfico já existente usando as funções *points* e *lines*:

```
plot(x,y)
points(rev(x),y)
lines(x,8000-y)
```

10.1.3. Mudando o padrão dos pontos

Pode-se usar diferentes padrões para os pontos usando o argumento pch=. Diferentes tipos de símbolos são associados a diferentes números. Pode-se ainda usar caracteres como o símbolo desejado.

```
plot(x,y)
points(rev(x),y,pch=3)        #adiciona cruzes
points(x,8000-y,pch="%")      #usando o símbolo porcento
```

Os primeiros símbolos numéricos para gráficos são os seguintes:

Números 7 a 14 são composições de símbolos obtidos por sobreposição dos símbolos básicos. Os números 15 to 18 são versões sólidas dos símbolos 0 a 4. Examine os exemplos (ou ver função que gerava um gráfico com símbolos):

```
plot(x,y)
plot(x,y,pch="@")
plot(x,y,pch=1:3)
plot(1:20,1:20,pch=1:20)      #útil para exibir os vários símbolos
```

10.1.4. Mudando as linhas

A largura das linhas pode ser mudada com o argumento `lwd=`, enquanto os estilos das linhas podem ser modificados com o argumento `lty=`:

```
plot(x,y)
lines(x,y,lwd=2)        #linha grossa
lines(rev(x),y,lty=2)   #linha interrompida
```

10.1.5. Definindo o intervalo dos eixos

Se você quiser preencher um mesmo gráfico com linhas e pontos que possuem diferentes amplitudes deve usar o argumento `type="n"`. Com este argumento um "gráfico em branco" é criado, são ajustadas apenas as margens e eixos do gráfico e o restante é deixado em branco. A seguir adiciona-se linhas e pontos desejados. Você deve fornecer coordenadas x e y que cubram a amplitude de valores de todos os elementos que você deseja adicionar ao gráfico.

Exemplo:

```
plot(c(0,20),c(-8000,8000),type='n')
lines(x,y)
lines(x,-y)
```

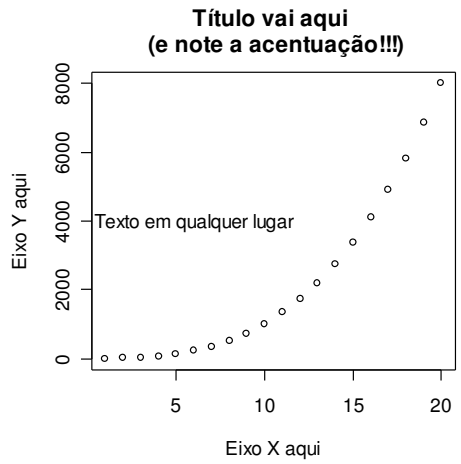
Outro exemplo:

```
plot(c(0,20),c(0,30),type='n')
segments(5,3,15,20)
lines(c(12,15,7),c(3,10,8),col="red")
abline(30,-2,lty=2,col="blue")
```

10.1.6. Adicionando texto

Você pode dar nome aos eixos com os argumentos `xlab=` e `ylab=`. O título pode ser fornecido usando a função `title()`, e qualquer texto pode ser adicionado em qualquer lugar do gráfico utilizando a função `text()`:

```
plot(x,y,xlab="Eixo X aqui",ylab="Eixo Y aqui")
title("Título vai aqui \n (e note a acentuação!!!)")
text(6,4000,"Texto em qualquer lugar")
```



10.1.7. Identificadores no gráfico

Por vezes, é interessante identificarmos um ponto ou um conjunto de pontos especificamente em um gráfico gerado. Essa identificação pode ser facilmente obtida e de maneira bem interativa quando usamos o comando `identify()`.

Exemplo:

Suponha que temos um conjunto de cidades e suas respectivas coordenadas planas (x e y). Assim, cada cidade pode ser identificada individualmente por suas coordenadas. Veja:

```
x<-c(2,3,4,5,6,7,8,9)                          #coordenadas x
y<-c(15,46,56,15,81,11,61,55)                  #coordenadas y
nomes<-paste("cidade",LETTERS[1:8],sep="")     #nomes das cidades
cidades<-data.frame(x,y,row.names=nomes)       #juntando tudo...
cidades                                        #exibindo...
```

```
        x  y
```

```
cidadeA 2 15
 ... ... ...
cidadeG 8 61
cidadeH 9 55
```

Assim, a cidadeA tem coordenadas (2,15), a cidadeB (3,46) e assim sucessivamente.

Podemos plotar as coordenadas das cidades de modo a poder visualizar sua distribuição espacial.

```
plot(cidades)
```

Acontece que não conseguimos distinguir facilmente na figura qual é a cidadeC, por exemplo. Esse problema pode ser resolvido quando temos indexados os nomes das cidades às coordenadas plotadas no gráfico. Então usamos o comando `identify()`.

Após entrarmos com o comando, quando passamos o mouse por sobre o gráfico ele vira uma cruz, e ao clicar próximo ao ponto que se deseja identificar, sua descrição e exibida instantaneamente.

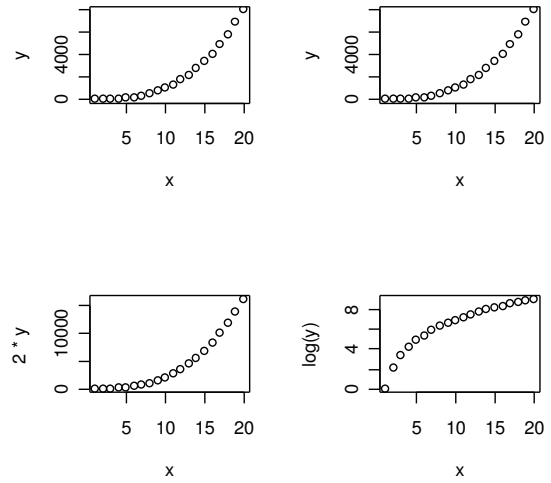
```
identify(x,y,           #coordenadas gráficas dos pontos
      nomes,            #identificação descrita dos pontos
      n=3)              #número de pontos a serem identificados
```

Como podemos perceber o programa ainda exibe, no *prompt*, os valores dos índices das referências.

10.1.8. Gráficos múltiplos

Você pode dar instruções para o programa mostrar diversos gráficos pequenos em uma mesma janela ao invés de um apenas. Para isto use a função `par()`:

```
par(mfrow=c(2,2)) #arranjamento "2 por 2"
plot(x,y)
plot(x,y)
plot(x,2*y)
plot(x,log(y))
# etc…
```



Neste caso você pode acomodar até 2x3=6 gráficos na mesma janela. A tela vai "limpar" quando você tentar fazer o sétimo gráfico.

A função `par()` pode fazer diversas outras coisas relacionadas aos gráficos. Veremos outras funcionalidades mais tarde, e você pode também consultar a documentação com `help(par)` ou `?par`.

OBS.: Para retornar ao padrão com apenas um gráfico por janela digite `par(mfrow=c(1,1))` ou feche a janela gráfica antes de outro comando.

Uma outra opção para gerar gráficos múltiplos é através da função *layout*. Veja as instruções utilizando a documentação através do comando `help(layout)` ou `?layout`.

10.1.9. Parâmetros Gráficos

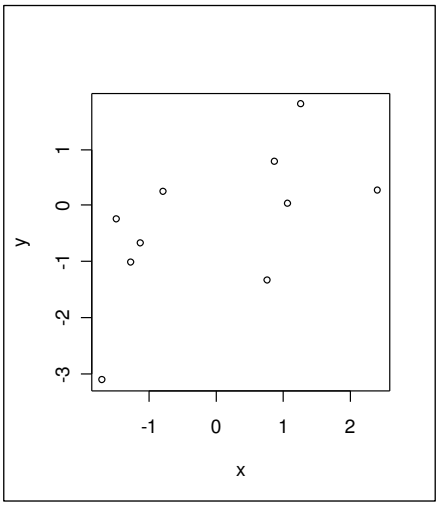
Alguns parâmetros podem ser usados no intuito de personalizar um gráfico no R. A lista completa desses parâmetros pode ser obtida com o comando `?par`. Veja abaixo um exemplo de aplicação do uso de parâmetros para alterar a forma de apresentação de um gráfico.

Exemplo:

Vamos criar dois conjuntos de 10 números cada um, gerados aleatoriamente com distribuição pseudo-normal de média zero e desvio padrão igual a um.

```
x<-rnorm(10)
y<-rnorm(10)
```

Agora compare os dois gráficos abaixo. Ambos referem-se aos mesmos objetos. Porém o segundo contém uma série de recursos adicionais. Compare também os códigos usados para gerar cada um deles.



1º gráfico:

```
plot(x,y)
```

2º gráfico:

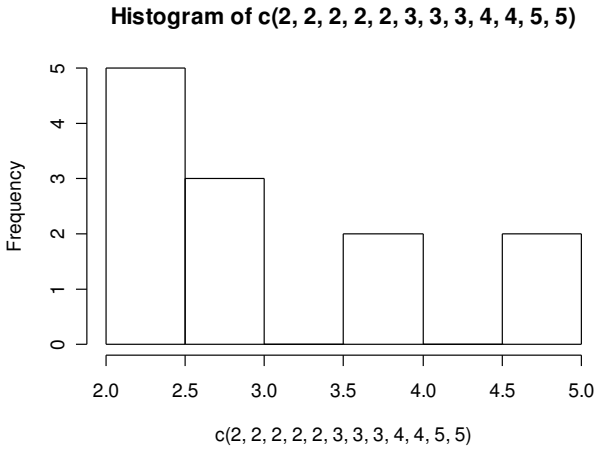
```
plot(x, y,                                    #plota x e y
      xlab="Dez números quaisquer",           #nomeia o eixo x
      ylab="Outros dez números",              #nomeia o eixo y
      main="Como personalizar um gráfico",#referente ao título
      xlim=c(-2,3),                           #limites do eixo x
      ylim=c(-3,2),                           #limites do eixo y
      col="red",                              #define a cor dos pontos
      pch=22,                                 #o formato dos pontos
      bg="yellow",                            #cor de preenchimento
      tcl=0.4,                                #tamanho dos traços dos eixos
      las=1,                                  #orientação do texto em y
      cex=1.5,                                #tamanho do objeto do ponto
      bty="l")                                #altera as bordas
```

10.2. Histogramas

10.2.1. Um exemplo bem simples

A função `hist()` produz um histograma dos dados informados em seu argumento enquanto a função `barplot()` produz um gráfico de barras. Veja:

```
hist(c(2,2,2,2,2,3,3,3,4,4,5,5))
```



Para criar exemplos melhores vamos utilizar a função `runif()` que gera números aleatórios com distribuição uniforme entre 0 e 1, como já mencionado anteriormente neste material. Veja:

```
x<-runif(100)      #100 números aleatórios da distribuição supracitada
hist(x)            #cria um histograma dos dados armazenados em x
```

Isto mostra uma distribuição razoavelmente uniforme dos 100 números. Mas se ficarmos adicionando mais números aos anteriores iremos obter uma distribuição aproximadamente normal. Para ilustrar isto vamos primeiro dividir a área gráfica para acomodar 4 gráficos com o comando:

```
par(mfrow=c(2,2))
```

Então criar vários histogramas com "x" sendo adicionado de outros valores amostrais gerados pela função `runif()`:

```
hist(x)
x<-x+runif(10)
hist(x)
x<-x+runif(100)
hist(x)
x<-x+runif(1000)
hist(x)
```

E gradativamente a distribuição resultante irá se aproximar da distribuição Normal.

10.2.2. Alterando alguns parâmetros

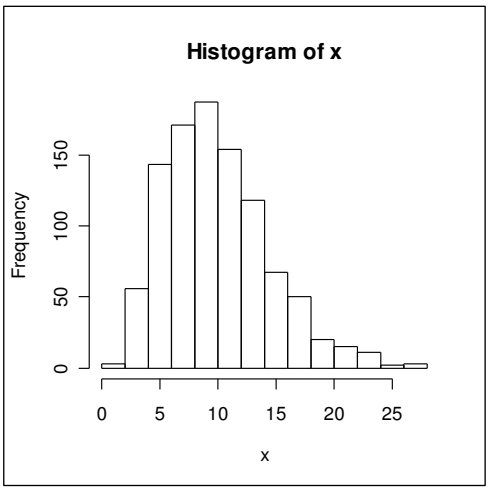
Os histogramas criados no R seguem um certo padrão (conhecido como parâmetros *default*) que podem ser alterados de acordo com a preferência do usuário. Você pode obter informações detalhadas desses parâmetros se usar os recursos de ajuda do R. Contudo, vamos mostrar algo, a esse respeito, com o exemplo que se segue.

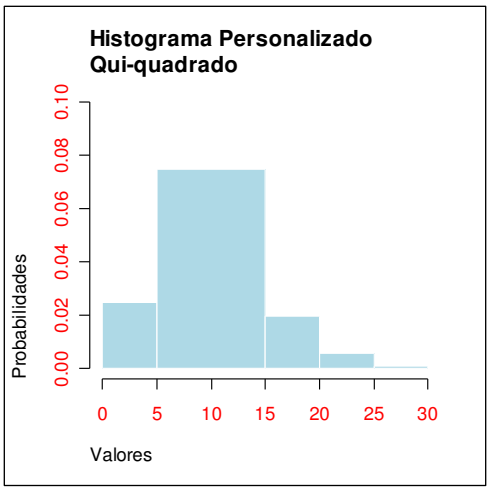
Exemplo:

Vamos criar um conjunto de dados que siga a distribuição de Qui-quadrado com 1000 elementos e com 10 graus de liberdade.

```
x<-rchisq(1000,10)
```

Agora, de maneira análoga ao exemplo visto anteriormente em "Parâmetros gráficos", vamos comparar dois histogramas gerados por códigos diferentes. O primeiro, é criado da maneira mais simples possível e o segundo com o uso de vários parâmetros alterados pelo gosto e/ou necessidade do usuário.





1º gráfico

```
hist(x)
```

2º gráfico

```
hist(x,                              #histograma de x
     main="Histograma Personalizado\nQui-quadrado",#título
     xlab="Valores",                 #texto do eixo das abscissas
     ylab="Probabilidades",          #texto do eixo das ordenadas
     br=c(c(0,5),c(5,15),5*3:6),     #intervalos das classes
     xlim=c(0,30),                   #limites do eixo de x
     ylim=c(0,0.1),                  #limites do eixo y
     col="lightblue",                #cor das colunas
     border="white",                 #cor das bordas das colunas
     prob=T,                         #para mostrar as probabilidades
     right=T,                        #intervalos fechados à direita
     adj=0,                          #alinhamento dos textos
     col.axis="red")                 #cor do texto nos eixos
```

Resolvendo com o R...

Suponha um conjunto de dados coletados por um professor que se refere ao tempo gasto (em minutos) pelos alunos para a resolução de um problema de álgebra. Veja:

25 27 18 16 21 22 21 20 18 23 27 21 19 20 21 16

Construa um histograma do conjunto de dados usando 6 classes com intervalos fechados à esquerda.

```
dados<-c(25,27,18,16,21,22,21,20,18,23,27,21,19,20,21,16)
hist(dados,                    #este é o conjunto de dados
     nc=6,                     #número de classes igual a 6
     right=F,                  #para o intervalo fechado à esquerda
     main="Histograma",        #define o título do histograma
     xlab="tempo (em minutos)",#texto do eixo x
     ylab="frequencia",        #texto do eixo y
     col=8)                    #usa a cor cinza nas barras
```

10.2.3. Ogiva

A ogiva nada mais é do que o histograma da freqüência acumulada ao invés da simples. Veja a comparação abaixo:

```
par(mfrow=c(1,2))                          #dois gráficos em uma janela
fi<-c(rep(2,3),rep(4,6),5,rep(6,2))        #freqüência simples
fa<-c(rep(2,3),rep(4,9),rep(5,10),rep(6,12))#freqüência acumulada

hist(fi,nc=4,ylim=c(0,12),main="Histograma")#"freq simples"
hist(fa,nc=4,ylim=c(0,12),main="Ogiva")   #histograma da "acumulada"
```

10.3. Gráficos de barras

Verifique estes comandos:

```
barplot(table(c(2,2,2,2,2,3,3,3,4,4,5,5)))
barplot(table(c(2,2,2,2,2,3,3,3,4,4,5,5)),hor=T)
```

Para saber mais sobre gráficos do R, veja nos manuais.

OBS.: ***Digite*** **`demo(graphics)`** ***no prompt para visualizar alguns exemplos de gráficos que podem ser gerados no R.***

11. Testes Estatísticos

O R inclui em sua gama de utilidades, uma poderosa ferramenta da estatística contemporânea: os testes estatísticos. Dentre esses, podemos destacar os testes de média, amplamente usados em várias áreas do conhecimento.

11.1. Teste t (de Student)

O teste t é bastante usado em várias situações do cotidiano quando se deseja fazer comparações entre uma ou mais médias, sejam elas dependentes ou não.

Abaixo estão exemplos de vários modos de realizarmos o teste t. Para todos eles utilizaremos os dois vetores abaixo:

```
x<-c(30.5,35.3,33.2,40.8,42.3,41.5,36.3,43.2,34.6,38.5)
y<-c(28.2,35.1,33.2,35.6,40.2,37.4,34.2,42.1,30.5,38.4)
```

11.1.1. Para uma média

Vamos testar se x tem média igual ou maior que 35. Então:

$H_0 : \mu_x = 35$

$H_a : \mu_x > 35$

Agora, para realizarmos o teste basta entrar com o comando:

```
t.test(x,                      #amostra a ser testada
       mu=35,                  #hipótese de nulidade
       alternative="greater")  #teste unilateral pela direita
```

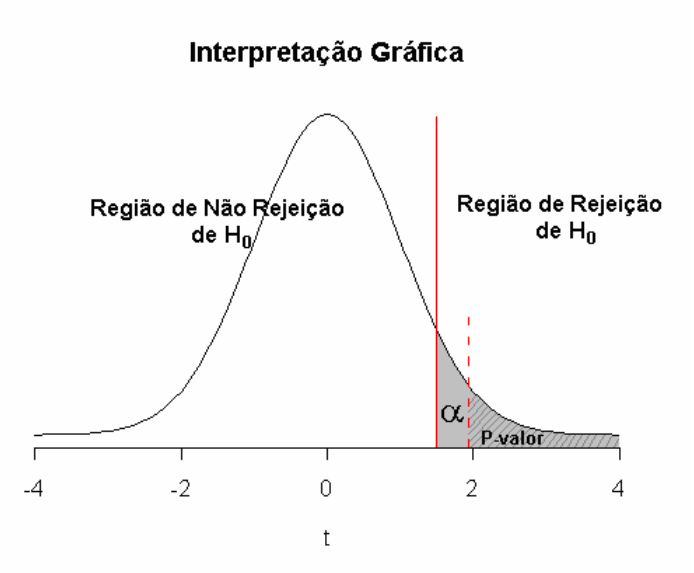
```
        One Sample t-test

data:  x
t = 1.9323, df = 9, p-value = 0.04268
alternative hypothesis: true mean is greater than 35
95 percent confidence interval:
 35.13453      Inf
sample estimates:
mean of x
    37.62
```

Agora basta fazer a interpretação correta da saída do R.

Para saber qual hipótese foi aceita, basta verificar o valor do p-value e estipular um nível de significância. Se neste exemplo o nível de significância (α) fosse de 5% a hipótese alternativa seria aceita uma vez que o p-value foi menor ou igual a 0,05. Caso o p-value tivesse sido maior que 5% então aceitaríamos a hipótese de nulidade.

Como a hipótese alternativa foi a aceita isso implica que a amostra "x" possui média estatisticamente diferente do valor 35 a um nível de significância de 5%.

Veja o Gráfico que interpreta este resultado:



Veja que a linha tracejada vermelha define o valor da estatística do teste (t_{calc}) que neste exemplo é 1,9323. Este valor define o início da região hachurada, que é o p-value.

Observe que o p-value é menor que o nível de significância α (que é toda a região cinza, incluindo a área do p-value). Portanto o t_{calc} está na Região de Rejeição da hipótese de nulidade.

11.1.2. Para duas médias independentes

Para a realização do teste t pressupõe-se que as amostras possuem variâncias iguais além de seguirem distribuição normal. Vamos a um exemplo:

Exemplo:

Suponha que queremos testar se "x" e "y" possuem média estatisticamente iguais a um nível de significância de 1%. Suponha também que essas amostras sejam independentes. Logo:

$$H_0 : \mu_x = \mu_y$$

$$H_a : \mu_x \neq \mu_y$$

E agora o comando...

```
t.test(x,y,                    #amostras a serem testadas
      conf.level = 0.99)       #nível de confiança
```

```
         Welch Two Sample t-test

data:  x and y
t = 1.1148, df = 17.999, p-value = 0.2796
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 -3.369829  7.629829
sample estimates:
```

```
mean of x mean of y
     37.62     35.49
```

O raciocínio para interpretação do resultado pode se fazer de maneira análoga ao exemplo anterior (quando estávamos testando apenas uma média).

11.1.3. Para duas médias dependentes

Neste caso vamos usar o mesmo nível de significância do exemplo das amostras independentes.

As hipóteses se mantêm:

$H_0 : \mu_x = \mu_y$

$H_a : \mu_x \neq \mu_y$

Agora basta adicionar o argumento `paired=T`, informando que as amostras são dependentes.

```
t.test(x,y,              #amostras a serem testadas
      conf.level=0.99,  #nível de confiança
      paired=T)         #indica dependência entre as amostras
```

```
        Paired t-test

data:  x and y
t = 3.6493, df = 9, p-value = 0.005323
alternative hypothesis: true difference in means is not equal to 0
99 percent confidence interval:
 0.2331487 4.0268513
sample estimates:
mean of the differences
                   2.13
```

Mais uma vez o resultado do teste pode ser obtido pela interpretação do p-value, ou usando o intervalo de confiança: se a média da diferença entre as médias estiver contida no intervalo de confiança, implica que essa diferença não é significativa.

11.2. Teste F

Exemplo:

Verificar se duas máquinas produzem peças com a mesma homogeneidade a resistência à tensão. Para isso foram sorteadas amostras que consistiam de 6 peças de cada maquina e obtivemos as seguintes resistências.

Máquina A	145	127	136	142	141	137
Máquina B	143	128	132	138	142	132

O que se pode concluir fazendo um teste de hipótese adequado para um nível de significância de 5%?

Segundo o teste F, podemos montar as seguintes hipóteses:

$$H_0 : \sigma_A^2 = \sigma_B^2$$

$$H_a : \sigma_A^2 \neq \sigma_B^2$$

Primeiro vamos fazer o teste passo a passo, usando o R como uma calculadora:

Para realizar as análises no R vamos entrar com os dados nos objetos que vamos chamar de "ma" e "mb" e calcular o tamanho das amostras que vão ser armazenadas nos objetos "na" e "nb". Observe que ambos os conjuntos de dados possuem 1 grau de liberdade.

```
ma<-c(145,127,136,142,141,137)
na<-length(ma)
mb<-c(143,128,132,138,142,132)
nb<-length(mb)
```

Já que iremos usar o teste F, temos:

$$Fcal = \frac{maior\ s^2}{menor\ s^2}$$

```
vma<-var(ma)
vmb<-var(mb)
fcal<-vma/vmb
fcal        #observando o valor da estatística do teste
```

```
 [1] 1.082056
```

```
pval<-pf(fcal,na-1,nb-1,lower=F)*2  #calculando o valor o p-value.
pval
```

```
 [1] 0.9331458
```

Observe que a multiplicação por 2 no final da expressão que dá o valor do p-value ocorreu porque se trata de um teste bilateral.

Existe uma maneira muito mais fácil de fazer um teste F no R. Veja:

Pode-se escrever uma função, ou utilizar a função existente no R, que para o caso é `var.test`. Vejamos seus argumentos:

```
args(var.test)
```

```
 function(x, ....)
 NULL
```

Note que a saída não é muito informativa. Isto indica que `var.test` é um método com mais de uma função associada. Vamos usar então:

```
?var.test
```

Logo vemos que basta informar os vetores à função:

```
var.test(ma,mb)
```

```
        F test to compare two variances

 data:  ma and mb
```

```
F = 1.0821, num df = 5, denom df = 5, p-value = 0.9331
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.1514131 7.7327847
sample estimates:
ratio of variances
          1.082056
```

Note que esta análise foi baseada na hipótese alternativa de variâncias diferentes (teste bi-lateral). A interpretação do resultado pode ser feito, da mesma maneira que no teste t, pelo valor do p-value. Como este é maior que 0,05 ou até mesmo que 0,90 aceitamos a hipótese de nulidade com um nível de significância de 5% e, se fosse viável, até mesmo com esse nível igual a 90%.

11.3. Outros testes

11.3.1. Qui-quadrado

Suponha que você deseje avaliar se uma amostrem (n= 100) dos números de 0 a 10 é realmente aleatória.

Para gerar a amostra basta:

```
amos<-round(runif(100)*10)
```

Para conferir a freqüência é só usar a função *table*:

```
freq<-table(amos)
freq
```

```
 amos
  0  1  2  3  4  5  6  7  8  9 10
  5 11 11  9 13 15  9  6  9  7  5
```

Uma forma prática de se capturar estas freqüências é usar o comando *as.numeric()*.

```
chisq.test(as.numeric(freq))
```

```
   Chi-squared test for given probabilities
 data:  as.numeric(freq)
 X-squared = 11.54, df = 10, p-value = 0.317
```

11.3.2. Kolmogorov-Smirnov

Exemplo:

Pega-se 50 pessoas ao acaso e mensura-se suas respectivas massas em quilogramas (Kg). Agora queremos saber se esse conjunto de dados segue a distribuição de qui-quadrado com um nível de significância de 5%.

Vamos assumir que já temos os valores coletados no vetor "pesos". Lembre das diferentes formas de entrar com dados no R.

```
 46.88 47.17 64.46 67.84 85.76 65.41 60.10 75.84 61.21
 61.65 63.87 53.95 63.66 69.06 76.41 75.56 69.04 35.18
 66.42 58.78 73.02 51.69 90.88 53.01 64.31 61.91 79.42
 57.78 62.73 60.63 63.29 46.53 84.64 61.76 85.08 59.66
 54.89 94.18 59.89 68.56 75.66 72.06 62.00 43.43 73.38
 73.31 66.37 73.72 66.15 67.79
```

Agora basta entrar com o comando corretamente:

```
ks.test(pesos,          #amostra a ser testada
      "pchisq",         #"p" seguido do nome da distribuição
      49)               #são os graus de liberdade da amostra

        One-sample Kolmogorov-Smirnov test

data:  pesos
D = 0.6402, p-value < 2.2e-16
alternative hypothesis: two.sided
```

Como o p-value é menor ou igual a 0,05 (5%) podemos assumir que os dados não tem distribuição de qui-quadrado a 5% de probabilidade.

11.3.3. Teste para a normalidade - shapiro.test()

Por vezes temos necessidade de identificar com certa confiança se uma amostra ou conjunto de dados segue a distribuição normal. Isso é possível, no R, com o uso do comando `shapiro.test()`. Veja:

Agora aplicando o teste ao exemplo anterior.

```
shapiro.test(pesos)

        Shapiro-Wilk normality test

data:  pesos
W = 0.9835, p-value = 0.7078
```

Observe que não há necessidade de informar a distribuição, uma vez que este teste é usado apenas para a distribuição normal (conhecida por muitos como distribuição de Gauss).

Apenas para melhor assimilar o teste, veja:

```
qqnorm(pesos)     #obtendo o normal probability plot só para comparação
qqline(pesos)     #colocando uma linha auxiliar
```

O comando `qqnorm()` nos fornece diretamente um gráfico da distribuição de percentagens acumuladas chamado de gráfico de probabilidade normal. Se os pontos deste gráfico seguem um

padrão aproximado de uma reta, este fato evidencia que a variável aleatória em questão tem a distribuição aproximadamente normal.

OBS.: ***Um teste de comparações múltiplas será abordado mais adiante.Outros testes podem ser encontrados na documentação do R utilizando o comando help.search(nome_do_teste).***

12. Análise de Variância (ANOVA)

Estudos estatísticos contemporâneos contemplam a análise de variância tendo em vista que este procedimento permite identificar e quantificar as variações ocorridas em um experimento, discriminando a parte da variação associada ao modelo pelo qual o experimento foi procedido da variação que se dá devido ao acaso.

No R encontram-se diversos procedimentos para se executar a ANOVA, entretanto o usuário deve estar atento ao escolher e realizar a análise, pois alguns erros são freqüentes, como, por exemplo, não especificar algum fator, esquecer sinal no modelo, dentre outros.

A tabela abaixo mostra alguns modelos e suas usuais formulações:

Modelo	Fórmula
DIC	y~t onde t é uma variável categórica
DBC	`y~t+b`
DQL	`y~t+l+c`
Fatorial/ DIC	y~N*P igual a N + P + N:P
Fatorial/ DBC	y~b+N*P igual a b+N+P+N:P
Regressão linear simples	y~x onde x é uma variável exploratória
Regressão quadrática	y~x+x2 onde x2 é um objeto x2<-x^2

`lm()` para regressão linear (*linear models*);

`aov()` para ANOVA, com erros NID;

`glm()` para ANOVA, com estrutura de erros especificada. (*generalised linear models*);

`nlme()` para modelos mistos (*nonlinear mixed-effects models*);

`nls()` para modelos não lineares (*nonlinear least squares*).

12.1. DIC

O DIC (Delineamento Inteiramente Casualizado) trata de experimentos onde os dados não são pré-separados ou classificados em categorias.

Exemplo:

Entrada dos dados da resposta do experimento:

```
res<-scan()
```

```
25 31 22 33 26 25 26 29 20 28 28 31 23 27 25 34 21 24 29 28
```

Criando os nomes dos tratamentos na ordem correspondente:

```
trat<-factor(rep(paste("tr",1:4,sep=""),5))
```

Fazendo a ANOVA:

```
resultado<-aov(res~trat)
```

Para exibir o quadro da ANOVA:

```
anova(resultado)
```

```
 Analysis of Variance Table

 Response: res
           Df  Sum Sq Mean Sq F value   Pr(>F)
 trat       3 163.750  54.583  7.7976 0.001976 **
 Residuals 16 112.000   7.000
 ---
 Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

Pode-se obter os gráficos da ANOVA facilmente com:

```
plot(resultado)
```

OBS.: ***Em todos os tipos de análises de variância, para todas as variáveis qualitativas devem ser criados fatores e não vetores, ou seja, o objeto que contém os nomes (ou números) dos tratamentos, dos blocos, etc., devem ser fatores e não vetores.***

12.2. DBC

O DBC (Delineamento em Blocos Casualizados) abrange os três princípios básicos da experimentação: repetição, casualização, e o controle local.

Veja o exemplo de uma análise de variância no R de um experimento segundo o DBC.

Exemplo:

Suponha que uma Nutricionista elaborou 4 dietas e quer aplicá-las em 20 pessoas a fim de testar suas eficiências quanto à perda de peso. Porém ela notou que entre essas 20 pessoas existem 5 grupos de faixas iniciais de peso. Então, para aumentar a eficácia do teste ela separou os 20 indivíduos em 5 grupos de faixas de peso. Então ela tem:

Dietas: dieta 1 – dieta 2 – dieta – 3 – dieta 4;

Grupos: peso A – peso – B – peso C – peso D – peso E.

A tabela abaixo resume o valor da perda de peso, arredondados em quilogramas, de cada indivíduo. Veja:

	dieta 1	dieta 2	dieta 3	dieta 4
peso A	2	5	2	5
peso B	3	7	4	3
peso C	2	6	5	4
peso D	4	5	1	3
peso E	2	5	4	4

A Nutricionista deseja determinar se existe diferença significativa entre as dietas a um nível de significância de 5%.

Esse é um problema típico de ANOVA usando o DBC, onde os blocos são os grupos de pesos. Veja como podemos proceder no R.

Criando o vetor de dados, o de tratamentos e o de blocos, respectivamente:

```
dad<-c(2,5,2,5,3,7,4,3,2,6,5,4,4,5,1,3,2,5,4,4)
bloc<-gl(5,4,label=c(paste("peso",LETTERS[1:5])))
trat<-rep(paste("dieta",1:4),5)
```

Agora vamos criar um data.frame contendo todos os dados:

```
tabela<-data.frame(blocos=bloc,tratamentos=factor(trat),dados=dad)
tabela
```

Agora basta entrarmos com o comando de forma correta.

O comando que gera a análise de variância é o "`aov()`", e o comando que exibe o quadro da ANOVA é o "`anova()`". Então podemos gerar o quadro da análise de uma só vez, associando os dois comandos. Veja:

```
resultado<- aov(                    #procede os cálculos da ANOVA
      dados~tratamentos+blocos,     #modelo estatístico utilizado
      tabela)                       #objeto com os elementos do modelo

resultado   #chamando o objeto que contém a ANOVA
 Call:
    aov(formula = dados ~ tratamentos + blocos, data = tabela)

 Terms:
                 tratamentos blocos Residuals
 Sum of Squares         25.2    3.2      16.8
 Deg. of Freedom           3      4        12

 Residual standard error: 1.183216
 Estimated effects may be unbalanced
```

Porém esse não é o quadro de ANOVA com que estamos acostumados a lidar. É aí que entra o comando "`anova()`"

```
anova(resultado)  #gera a tabela de análise de variância
```

```
Analysis of Variance Table

Response: dados
            Df Sum Sq Mean Sq F value  Pr(>F)
tratamentos  3   25.2     8.4  6.0000 0.00973 **
blocos       4    3.2     0.8  0.5714 0.68854
Residuals   12   16.8     1.4
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

Agora basta interpretar os resultados: Observe que o efeito dos tratamentos (dietas) se deu de maneira significativa a 1% de significância, porém a 5% ele foi "não significativo", o que implica que a resposta encontrada pela Nutricionista foi: "Não, as dietas não apresentam diferenças significativas a 5% de probabilidade".A resposta do exemplo já foi dada, porém vamos prosseguir usando este mesmo exemplo pra mostrar como obter mais informações a respeito da ANOVA. Veja:

Caso quiséssemos obter os resíduos, poderíamos fazê-lo através do comando "resid()", que exibe os resíduos correspondentes a cada uma das 20 observações:

```
residuos<-resid(resultado)    #gerando um quadro de resíduos
residuos
     1      2      3      4      5      6      7      8      9     10
 -0.30  -0.30  -0.90   1.50  -0.05   0.95   0.35  -1.25  -1.05  -0.05
    11     12     13     14     15     16     17     18     19     20
  1.35  -0.25   1.95  -0.05  -1.65  -0.25  -0.55  -0.55   0.85   0.25
```

Podemos observar que a soma dos resíduos tende a zero:

```
sum(residuos)
 [1] 4.510281e-16
```

E que a soma dos quadrados desses resíduos corresponde a SQResíduo da ANOVA:

```
sum(residuos^2)
 [1] 16.8
```

Os totais de tratamentos podem ser obtidos por:

```
tapply(dad,trat,sum)
 dieta 1 dieta 2 dieta 3 dieta 4
      13      28      16      19
```

As médias de tratamentos podem ser obtidas por:

```
tapply(dad,trat,mean)
 dieta 1 dieta 2 dieta 3 dieta 4
     2.6     5.6     3.2     3.8
```

Os totais de blocos podem ser obtidos por

```
tapply(dad,bloc,sum)
```

```
 peso A peso B peso C peso D peso E
     14     17     17     13     15
```

As médias de blocos podem ser obtidas por:

```
tapply(dad,bloc,mean)
```

```
 peso A peso B peso C peso D peso E
   3.50   4.25   4.25   3.25   3.75
```

Esses dados podem ser úteis em análises futuras.

OBS.: ***Para alterar certos valores no conjunto de dados originais, podemos fazer*** `tabela<-edit(tabela)`***. Será aberta uma planilha no R. Após a alteração dos valores de interesse, basta fechar a janela da planilha que as alterações serão automaticamente salvas no objeto "tabela"***

12.3. Fatorial

Os experimentos fatoriais são aqueles em que estudamos dois ou mais fatores simultaneamente, onde cada um desses fatores pode possuir dois ou mais níveis. A vantagem desse tipo de experimento é que além de termos o "controle" dos fatores individualmente, consideramos também a interação entre eles, ou seja, sabemos se esses fatores atuam de formas independentes ou se existem interações entre eles. Os experimentos fatoriais podem ser conduzidos segundo o DIC, DBC e outros modelos.

12.3.1. Experimentos com 2 fatores segundo o DIC

Exemplo:

Um Engenheiro Agrimensor resolve estudar o efeito da distância e do ângulo de visada ao alvo no erro linear cometido na centragem do ponto a ser visado. Ele quer saber também se esses dois fatores atuam relacionadamente ou independentemente. Então ele resolve fazer um experimento fatorial segundo um DIC com 2 repetições, obtendo o seguinte quadro:

	Dist 1		Dist 2		Dist 3		Dist 4	
Ang 1	0,7	0,5	1,0	1,3	1,0	0,9	0,9	0,9
Ang 2	1,5	1,6	2,0	1,2	1,2	1,3	1,6	1,2
Ang 3	0,8	1,2	1,9	0,6	1,6	1,1	1,3	1,0

A ANOVA para este experimento pode ser montada com:

```
e<-
c(0.7,0.5,1.0,1.3,1.0,0.9,0.9,0.9,1.5,1.6,2.0,1.2,1.2,1.3,1.6,1.2,0.8,1.
2,1.9,0.6,1.6,1.1,1.3,1.0)
a<-gl(3,8,label=c(paste("Ang",1:3)))
d<-rep(gl(4,2,label=c(paste("Dist",1:4))),3)
dados<-data.frame(angulos=a,distancias=d,erro=e)
```

```
anova(aov(erro~distancias+angulos+distancias*angulos,dados))
 Analysis of Variance Table

 Response: erro
                    Df  Sum Sq Mean Sq F value  Pr(>F)
 distancias          3 0.24792 0.08264  0.6296 0.60974
 angulos             2 1.21083 0.60542  4.6127 0.03266 *
 distancias:angulos  6 0.34583 0.05764  0.4392 0.83909
 Residuals          12 1.57500 0.13125
 ---
 Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

O quadro da ANOVA mostra que distância e ângulo atuam independentemente. Podemos verificar também que houve diferença significativa apenas nos ângulos se adotarmos um nível de significância de 1%. O restante da variação encontrada nos valores do erro se deu "apenas" devido ao acaso.

12.3.2. Fatorial usando o DBC

Experimentos fatoriais também podem ser conduzidos segundo um Delineamento em Blocos Casualizados. Veja o exemplo:

Exemplo:

Um Engenheiro Agrimensor quer testar diferentes modelos de alvos, a fim de avaliar se o desenho afeta os resultados. Ele deseja controlar o efeito da distância e do ângulo de visada, assim como no exemplo anterior. Então ele seleciona 3 modelos de alvos diferentes (A, B e C), e coleta os seguintes valores para o erro linear (em milímetros):

ALVO A	Dist 1	Dist 2	Dist 3
Ang 1	0,2	0,7	0,8
Ang 2	0,4	0,8	0,9
Ang 3	0,5	1,2	1,2

ALVO B	Dist 1	Dist 2	Dist 3
Ang 1	0,6	0,8	1,1
Ang 2	0,9	1,3	1,5
Ang 3	1,2	1,4	1,8

ALVO C	Dist 1	Dist 2	Dist 3
Ang 1	0,7	1,1	1,5
Ang 2	0,9	1,5	1,7
Ang 3	1,2	1,7	1,5

Fazendo a ANOVA no R:

```
e<-c(0.2,0.7,0.8,0.4,0.8,0.9,0.5,1.2,1.2,
      0.6,0.8,1.1,0.9,1.3,1.5,1.2,1.4,1.8,
      0.7,1.1,1.5,0.9,1.5,1.7,1.2,1.7,1.5)
b<-gl(3,9,label=c(paste("ALVO",LETTERS[1:3])))
a<-rep(gl(3,3,label=c(paste("Ang",1:3))),3)
d<-rep(paste("Dist",1:3),9)
dados<-data.frame(bloco=b,angulo=a,distancia=d,erro=e)
anova(aov(erro~distancia+angulo+bloco+distancia*angulo,dados))
```

```
Analysis of Variance Table

Response: erro
                 Df  Sum Sq Mean Sq F value    Pr(>F)
distancia         2 1.72667 0.86333 46.0444 2.305e-07 ***
angulo            2 0.98667 0.49333 26.3111 8.735e-06 ***
bloco             2 1.58000 0.79000 42.1333 4.204e-07 ***
distancia:angulo  4 0.03333 0.00833  0.4444    0.7748
Residuals        16 0.30000 0.01875
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

De acordo com a tabela da ANOVA podemos perceber que a distância e o ângulo de visada atuam independentemente, uma vez que a interação entre eles foi "não significativa". Podemos concluir também que os níveis dos fatores distância e ângulo de visada influenciam no erro. Mas o mais importante é que de acordo com a ANOVA pode-se afirmar que existe diferença na eficiência dos diferentes modelos de alvos, no que diz respeito ao erro linear, que era o principal objetivo do Engenheiro Agrimensor.

12.4. Experimentos em Parcelas Subdivididas

O termo "parcelas subdivididas", assim como o "fatorial", refere-se a maneira como os tratamentos são organizados. Neste tipo de experimento estuda-se dois tipos de fatores de maneira simultânea (fatores primários e secundários: parcelas e subparcelas respectivamente).

O R possibilita também a análise de experimentos conduzidos segundo experimentos dessa natureza. Veja o exemplo que segue:

12.4.1. Um exemplo segundo o DBC

Exemplo:

Suponha um experimento em parcelas subdivididas segundo o DBC como mostra a tabela abaixo:

A1	A2

Bloco	B1		B2		B3		B1		B2		B3	
1	12	12	15	14	15	16	21	19	22	20	16	19
2	15	16	16	17	12	12	18	19	19	21	21	20
3	17	16	13	15	12	11	17	19	20	18	19	21
4	14	13	16	15	14	17	16	17	17	20	18	18

Veja como podemos resolver usando o R:

```
A<-gl(2,24,label=paste("A",1:2,sep=""))   #cria o fator das parcelas
B<-rep(gl(3,8,label=paste("B",1:3,sep="")),2)   #fator das subparcelas
bl<-rep(gl(4,2,label=paste("bl",1:4,sep="")),6)
dados<-c(12,12,15,16,17,16,14,13,
      15,14,16,17,13,15,16,15,
      15,16,12,12,12,11,14,17,
      21,19,18,19,17,19,16,17,
      22,20,19,21,20,18,17,20,
      16,19,21,20,19,21,18,18)      #vetor das observações
tabela<-data.frame(A=A,B=B,bloco=bl,dados=dados)
saida<-aov(dados~bloco+A+B+A*B+Error(bloco/A),tabela)
summary(saida)
```

```
Error: bloco
      Df Sum Sq Mean Sq
bloco  3 5.5000  1.8333

Error: bloco:A
          Df Sum Sq Mean Sq F value  Pr(>F)
A          1 252.08  252.08  59.314 0.00455 **
Residuals  3  12.75    4.25
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Error: Within
          Df  Sum Sq Mean Sq F value Pr(>F)
B          2  12.042   6.021  2.0116 0.1485
A:B        2   4.542   2.271  0.7587 0.4756
Residuals 36 107.750   2.993
```

Uma informação importante: se o delineamento fosse o DIC, então o resíduo(a) seria adicionado do componente representado pelo bloco. Uma forma possível de obter as médias para todos os níveis dos fatores e suas combinações seria:

```
model.tables(saida,type="means")
```

Uma vez obtidas as médias, basta multiplicar cada valor pelo número de elementos que o originaram.

12.5. Teste de Comparações Múltiplas

12.5.1. Teste Tukey

Há vários testes de comparações múltiplas disponíveis na literatura, muitos deles também disponíveis no R, e os que não estão são um convite aos novos usuários a estarem implementando com os recursos do R.

Vejamos duas formas de se usar o teste de Tukey, a primeira usando a função `TukeyHSD()` e a segunda fazendo os cálculos necessários com o R. Para ambos os casos vamos usar os seguintes dados:

```
dados<-c(25,31,22,33,26,25,26,29,20,28,28,31,23,27,25,34,21,24,29,28)
trat<-factor(rep(paste("tr",1:4,sep=""),5))
tabela<-data.frame(trat=trat,dados=dados)
ANOVA<-aov(dados~trat,tabela)

result1<-TukeyHSD(ANOVA,"trat")
result1
```

```
   Tukey multiple comparisons of means
     95% family-wise confidence level

       Tukey multiple comparisons of means
         95% family-wise confidence level

     Fit: aov(formula = dados ~ trat, data = tabela)

     $trat
             diff        lwr       upr     p adj
     tr2-tr1    4 -0.7874018  8.787402 0.1192178
     tr3-tr1    3 -1.7874018  7.787402 0.3123298
     tr4-tr1    8  3.2125982 12.787402 0.0010547
     tr3-tr2   -1 -5.7874018  3.787402 0.9313122
     tr4-tr2    4 -0.7874018  8.787402 0.1192178
     tr4-tr3    5  0.2125982  9.787402 0.0391175
```

Você também pode visualizar os resultados graficamente através de:

```
plot(result1)
```

Desta maneira, são oferecidos muitos valores que não são de interesse e as vezes até dificulta a interpretação dos resultados.

Uma outra possibilidade é utilizar a distribuição de Tukey (opções q e p implementadas) para encontrar os valores de tabelados ou os valores de probabilidade (consultar `help(qtukey)`).

13. Regressão

O principal objetivo da análise de regressão é verificar se existe uma relação, e quantificar essa relação, entre duas variáveis quantitativas, tentando formular uma relação direta entre uma ou mais variáveis independentes e seu(s) efeito(s) na variável dependente. O melhor método para a escolha do modelo matemático/estatístico que representará essa relação pode ser obtido com a visualização do diagrama de dispersão. Os modelos podem ser de formas variadas: linear, quadrático, exponencial, logarítmico, etc. Veja a seguir como gerar alguns desses modelos usando o R.

13.1. Polinomial Simples

13.1.1. Linear

Quando os dados se agrupam seguindo a forma de uma reta, provavelmente existe uma relação de linearidade entre as variáveis envolvidas. Veja um exemplo típico:

Exemplo:

Um engenheiro civil coleta dados em um laboratório estudando a dilatação de um pilar de concreto segundo a temperatura ambiente no local onde está o pilar. Veja os dados (fictícios):

T (ºC)	18	16	25	22	20	21	23	19	17
Dilatação Linear (mm)	5	3	10	8	6	7	9	6	5

Posso realizar um estudo de regressão nestes dados?. Qual modelo usar? Como montar a equação que relaciona a temperatura com a dilatação neste estudo? A temperatura realmente exerce influência na dilatação do pilar? Eu posso quantificar essa relação?

Essas são as perguntas que podemos nos fazer ao depararmos com os dados que foram apresentados acima. Suas respostas podem ser encontradas fazendo-se uma análise de regressão. Veja:

Primeiro vamos entrar com os dados da tabela no R, criando 2 objetos: um que conterá os valores de temperatura e outro para a dilatação.

```
temp<-c(18,16,25,22,20,21,23,19,17)
dilat<-c(5,3,10,8,6,7,9,6,5)
```

O estudo de regressão pode ser feito, inicialmente com a definição do modelo. Para isso vamos visualizar os pontos plotados em um diagrama de dispersão:

```
plot(temp,dilat)  #a variável independente deve vir primeiro
```

O diagrama sugere uma tendência linear dos dados. Então vamos montar um modelo de regressão linear simples (simples, pois existe apenas uma variável independente – "temp" relacionada a variação da variável dependente – "dilat").

Montando o modelo:

```
reglin<-lm(dilat~temp)
reglin

 Call:
 lm(formula = dilat ~ temp)

 Coefficients:
 (Intercept)          temp
     -8.1710        0.7323
```

Da saída acima podemos tirar duas informações: o valor do intercepto (valor onde a reta de regressão intercepta o eixo da dilatação) e o valor que representa um coeficiente de relação entre a dilatação e a temperatura, ou seja, quanto a dilatação irá variar para cada variação unitária da temperatura. Esses valores são comumente chamados de "β_0" e "β_1" respectivamente.

Logo podemos concluir que o modelo matemático/estatístico desta regressão é:

$$\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 .X$$

$$\hat{Dil} = -8{,}1710 + 0{,}7323.T$$

onde a temperatura é dada em ºC e a dilatação em mm.

Podemos obter os valores estimados (preditos) pelos valores tabelados de "temp" da função com o comando:

```
predict(reglin)
          1          2          3          4          5
  5.009677   3.545161  10.135484   7.938710   6.474194
          6          7          8          9
  7.206452   8.670968   5.741935   4.277419
```

O primeiro valor acima (5,009677) representa o valor obtido para a dilatação quando a temperatura é 18ºC (primeiro valor do objeto "temp"), e assim sucessivamente até o último valor de "temp", gerando nove valores.

Agora vamos plotar novamente os dados e acrescentar a função encontrada no diagrama:

```
plot(temp,dilat)         #plotar o diagrama de dispersão
abline(reglin)           #desenha a reta de regressão ajustada
pred<-predict(reglin)    #usando os valores estimados
for(i in 1:(length(temp)))
      {
      lines(c(temp[i],temp[i]),c(dilat[i],pred[i]))
      #pequenos segmentos entre os valores observados e os calculados
      }
```

Podemos também realizar uma análise de variância na regressão com:

```
anova(reglin)
```

```
Analysis of Variance Table

Response: dilat
          Df Sum Sq Mean Sq F value    Pr(>F)
temp       1 36.938  36.938  201.40 2.048e-06 ***
Residuals  7  1.284   0.183
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1
```

Com ela podemos verificar que o coeficiente β_1 exerce influência significativa na regressão uma vez que o p-value encontrado foi da ordem de 10^{-6}.

E também obter muitas informações com:

```
summary(reglin)
```

```
Call:
lm(formula = dilat ~ temp)

Residuals:
      Min        1Q    Median        3Q       Max
-0.545161 -0.206452 -0.009677  0.258065  0.722581

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -8.1710     1.0475  -7.801 0.000107 ***
temp          0.7323     0.0516  14.191 2.05e-06 ***
---
Signif. codes:  0 `***' 0.001 `**' 0.01 `*' 0.05 `.' 0.1 ` ' 1

Residual standard error: 0.4283 on 7 degrees of freedom
Multiple R-Squared: 0.9664,     Adjusted R-squared: 0.9616
F-statistic: 201.4 on 1 and 7 DF,  p-value: 2.048e-06
```

Com o quadro acima podemos fazer toda uma análise que não detalharemos neste material.

13.1.2. De grau maior que 1

Qualquer modelo de regressão polinomial pode ser obtido com um comando simples: o `lm()` que vem do inglês *"linear models"*.

Veja o exemplo abaixo:

```
fert<-c(10,20,30,40,50,60,70,80,90,100)
prod<-c(42,61,81,94,98,96,83,79,59,43)
plot(fert,prod)
```

Observe a necessidade do argumento `"I()"` para interações como `x^2`.

```
reg<-lm(prod~fert+I(fert^2))  #modelo de regressão quadrática
reg
```

```
Call:
```

```
lm(formula = prod ~ fert + I(fert^2))

Coefficients:
(Intercept)          fert     I(fert^2)
   15.51667       2.95720      -0.02716
```

Para “desenhar” a curva ajustada...

```
curve(15.51667+2.95720*x-0.02716*x*x,0,100,add=T,col=2)
```

Várias outras análises podem ser feitas como anteriormente na regressão linear. Veja uma delas:

```
anova(reg)
Analysis of Variance Table

Response: prod
           Df Sum Sq Mean Sq  F value    Pr(>F)
fert        1    7.6     7.6   0.5878    0.4683
I(fert^2)   1 3894.6  3894.6 302.2072 5.126e-07 ***
Residuals   7   90.2    12.9
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Os outros modelos de regressão polinomial podem ser obtidos de maneira análoga. Por exemplo, na regressão cúbica poderíamos escrever:

```
lm(y~x+I(x^2)+I(x^3))
```

E na de quarto grau:

```
lm(y~x+I(x^2)+I(x^3)+I(x^4))
```

E assim sucessivamente.

13.2. Polinomiais Múltiplos

Os modelos múltiplos são aqueles em que duas ou mais variáveis independentes influenciam na variação da variável dependente. Eles podem ser de grau 1, 2, ou maior. O exemplo a seguir aborda uma regressão polinomial múltipla de 3º grau. Veja:

Exemplo:

Vamos supor que queremos ajustar uma superfície de tendência - uma equação de regressão polinomial de grau 3 que descreva o comportamento das coordenadas de pontos que representam o relevo de um local. As coordenadas são dadas nos eixos cartesianos (x, y, z) onde z é a cota do ponto (dados abaixo).

Estamos supondo que z é função de x e y.

Um modelo polinomial de 3º grau tem a forma:

$$\hat{z} = \beta_0 + \beta_1 x + \beta_2 y + \beta_3 x^2 + \beta_4 xy + \beta_5 y^2 + \beta_6 x^3 + \beta_7 x^2 y + \beta_8 xy^2 + \beta_9 y^3$$

```
x<-1:15                                              #valores de x
y<-c(2,5,6,7,9,10,12,11,11,13,14,12,12,15,15)        #valores de y
z<-c(1,2,2,14,17,15,12,12,11,8,8,3,3,6,16)           #valores de z
coord<-data.frame(x,y,z)                             #juntando x,y,z

#agora vamos montar o modelo
modelo<-z~x+y+I(x^2)+I(x*y)+I(y^2)+I(x^3)+I(x^2*y)+I(x*y^2)+I(y^3)

#fazendo a regressão
re<-lm(modelo,coord)    #primeiro o modelo, depois os dados
re                      #veja o resultado
```

```
 Call:
 lm(formula = modelo, data = coord)

 Coefficients:
 (Intercept)            x            y       I(x^2)     I(x * y)
      8.2112      16.0255     -14.3588      -2.0589      -1.0705
      I(y^2)       I(x^3)   I(x^2 * y)   I(x * y^2)       I(y^3)
      2.5672       0.2580      -0.4776       0.4961      -0.2444
```

Um dos grandes problemas é entrar com todas as interações quando o polinômio a ser ajustado possui graus elevados. Só para se ter uma idéia, no ajuste de um polinômio de grau 7, por exemplo, temos 36 interações possíveis, x, y, x^2, xy, y^2, x^3, ... , y^7. Cada um desses termos deve ser explicitado na função `lm()`, de modo que teríamos que digitá-los um a um. Este problema pode ser resolvido com a função desenvolvida abaixo. O único parâmetro exigido por ela é o grau do polinômio que se deseja criar. Veja:

```
fpol<-function(d)
   {
   nt<-(((d+1)*(d+2))/2)-1
   termos<-rep("vazio",nt)
   cont<-1
   for (j in 1:d)
      {
      t<-j
      for (i in j:0)
         {
         termos[cont]<-paste("I(x^",i,"*y^",(t-i),")",sep="")
         cont<-cont+1
         }
      }
   f<-as.formula(paste("z~",paste(termos,collapse="+")))
   return(f)
   }
```

No caso do exemplo acima bastaríamos digitar:

```
modelo.da.funcao<-fpol(3)
modelo.da.funcao
```

```
 z ~ I(x^1 * y^0) + I(x^0 * y^1) + I(x^2 * y^0) + I(x^1 * y^1) +
     I(x^0 * y^2) + I(x^3 * y^0) + I(x^2 * y^1) + I(x^1 * y^2) +
     I(x^0 * y^3)
```

E assim ela funciona para qualquer grau polinomial. Em casos de graus muito elevados seu computador pode não ter memória suficiente para gerar as interações. Veja:

```
fpol(100)          #polinômio de grau 100 - 5151 interações!!!
 Error: protect(): protection stack overflow
```

13.2.1. Superfície de Resposta

Continuando o exemplo acima, vamos ajustar uma superfície de resposta com a regressão ajustada.

```
x<-1:15                                              #valores de x
y<-c(2,5,6,7,9,10,12,11,11,13,14,12,12,15,15)        #valores de y
z<-c(1,2,2,14,17,15,12,12,11,8,8,3,3,6,16)           #valores de z
coord<-data.frame(x,y,z)                             #juntando x,y,z
#agora vamos montar o modelo
modelo<-z~x+y+I(x^2)+I(x*y)+I(y^2)+I(x^3)+I(x^2*y)+I(x*y^2)+I(y^3)

#fazendo a regressão
re<-lm(modelo,coord)    #primeiro o modelo, depois os dados
re                      #veja o resultado
```

```
 Call:
 lm(formula = modelo, data = coord)

 Coefficients:
 (Intercept)            x            y       I(x^2)     I(x * y)
      8.2112      16.0255     -14.3588      -2.0589      -1.0705
      I(y^2)       I(x^3)   I(x^2 * y)   I(x * y^2)       I(y^3)
      2.5672       0.2580      -0.4776       0.4961      -0.2444
```

Os coeficientes da função ajustada podem ser obtidos um a um com:

```
re[[1]][[1]]      #para o primeiro coeficiente
re[[1]][[2]]      #para o segundo
re[[1]][[3]]      #para o terceiro e assim sucessivamente
```

Pode-se então construir uma função que represente de fato a função ajustada:

```
fun<-function(x,y)
      {re[[1]][[1]]+          #intercepto (beta 0)
      re[[1]][[2]]*x+         #beta 1 * x
      re[[1]][[3]]*y+         #beta 2 * y
      re[[1]][[4]]*x^2+       #beta 3 * x^2
      re[[1]][[5]]*x*y+       #beta 4 * x * y
      re[[1]][[6]]*y^2+       #beta 5 * y^2
      re[[1]][[7]]*x^3+       #beta 6 * x^3
      re[[1]][[8]]*x^2*y+     #beta 7 * x^2 * y
      re[[1]][[9]]*x*y^2+     #beta 8 * x * y^2
      re[[1]][[10]]*y^3}      #beta 9 * y^3
```

Agora vamos gerar uma superfície usando a função ajustada do exemplo acima:

```
valx<-seq(min(x),max(x),0.5)  #valores de x a entrar na função
valy<-seq(min(y),max(y),0.5)  #valores de y a entrar na função
superf<-outer(valx,valy,fun)  #matriz de valores interpolados por "fun"
```

O objeto "superf" contém os valores de coordenadas da superfície ajustada que pode ser exibida com o comando `persp()`. Veja:

```
persp(superf,           #objeto que contém os valores interpolados
      theta=310,        #angulo horizontal de exibição da superfície
      phi=30,           #angulo vertical
      expand=0.5,       #para expandir o gráfico
      col=2,            #usa a cor vermelha
      shade=0.5,        #sombreamento
      ticktype="simple")#apenas o sentido dos eixos (sem escala)
```

OBS.: ***Outros resultados gráficos podem ser obtidos com as alterações dos parâmetros da função `persp()`. Você pode ver alguns gráficos modelos digitando `demo(persp)`.***

13.3. Modelos não lineares

Para o ajuste de regressões não-lineares com o R aconselhamos o uso da função `nls()` do pacote "nlme", por sua simplicidade e versatilidade. É necessário que se carregue o pacote antes invocar a função. Pode-se utilizar os comandos `require(nlme)` ou `library(nlme)`.

Em seguida basta usar a seguinte estrutura de comando:

```
nls(modelo,dados,valores iniciais estimados dos parâmetros)
```

Você pode obter informações detalhadas deste comando digitando `?nls`.

Outros métodos podem ser usados para ajustar o modelo de regressão desejado. Você pode conhecer alguns deles digitando `help.search("regression")` no *R Console*.

Exemplo

Num projeto de construção de uma barragem é de grande interesse equacionar a relação entre a cota do nível d'água e o volume armazenado quando esta cota é atingida. Essa relação é obtida a partir de um diagrama cota-volume estimado através do levantamento topográfico da região onde será construída a barragem e suas respectivas curvas de nível.

Suponha os dados a seguir, com a cota dada em metros e o volume em quilômetros cúbicos:

```
cota<-c(1,2,3,4,5,6,7,8,9,10)
volume<-c(7,10,14,20,31,40,58,84,113,165)
dados<-data.frame(cota,volume)
```

Veja como os dados se dispersam:

```
plot(dados)
library(nlme)

funcao<-volume~a*exp(b*cota)
exponencial<-nls(funcao,      #modelo que se deseja ajustar($)
                 dados,       #data.frame com o conjunto de dados
                 start=c(a=1,b=1))#valores iniciais dos parâmetros($$)
```

```
#($)usando o método dos mínimos quadrados.
#($$)são valores iniciais estimados para os parâmetros
(coeficientes).Quando a equação converge esses valores podem ser
quaisquer diferentes de zero.

summary(exponencial)

Formula: volume ~ a * exp(b * cota)

Parameters:
  Estimate Std. Error t value Pr(>|t|)
a 5.116389   0.227553   22.48 1.62e-08 ***
b 0.346720   0.004879   71.06 1.71e-12 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.559 on 8 degrees of freedom

Correlation of Parameter Estimates:
        a
b -0.9885

#desenhando a curva ajustada
curve(5.1163887*exp(0.34672*x),     #equação ajustada
      1,                            #limite inferior eixo das abscissas
      10,                           #limite superior
      add=T,                        #acrescentar no gráfico anterior
      col=2)                        #cor da curva (2 = vermelha)
```

Pode-se obter a analise detalhada da regressão com:

14. Nonlinear Mixed-Effects Models

Para a análise de modelos que contenham efeitos aleatórios o R apresenta uma biblioteca muito versátil e extremamente poderosa, chamada *nlme – linear e nonlinear mixed effect modesl* (já usada anteriormente para o ajuste de regressões não lineares) que permite a avaliação de modelos mistos lineares e não lineares.

Para acessá-la basta entrar com o seguinte comando:

```
library(nlme)
```

Exemplo:

Vamos utilizar um exemplo contido no próprio R.

```
data(Orthodont)
Orthodont          #para visualizar os dados
```

Uma forma de escrever o modelo seria:

```
fm1<-lme(distance~age,Orthodont) #quando o efeito aleatório é "age"
fm1
```

```
 Linear mixed-effects model fit by REML
   Data: Orthodont
   Log-restricted-likelihood: -221.3183
   Fixed: distance ~ age
 (Intercept)         age
  16.7611111   0.6601852

 Random effects:
  Formula: ~age | Subject
  Structure: General positive-definite
             StdDev    Corr
 (Intercept) 2.3269555 (Intr)
 age         0.2264214 -0.609
 Residual    1.3100414

 Number of Observations: 108
 Number of Groups: 27
```

Se você quiser ver só os efeitos fixos:

```
anova(fm1)
```

Para os componetes de variância:

```
VarCorr(fm1)
```

Para obter os efeitos de aleatórios ou fixos:

```
ranef(fm1)  #para efeitos aleatórios
fixef(fm1)  #para efeitos fixos
```

Para ter informações adicionais como o AIC (Akaike's An Information Criterion):

```
AIC(fm1)
```

Para dados dos quais não se especificou a fórmula, pode-se usar:

```
expm<-lme(res~tratamentos,data=dados,random=~1|blocos)
```

Este seria um modelo em blocos casualizados com efeitos de tratamentos fixos e blocos aleatórios. E sempre que necessário utilize dos comandos de ajuda do R.

Referências e Bibliografias consultadas

CRAWLEY, M.J. **Statistical Computing to Data Analysis using S-plus**. New York: Wiley, 761p. 2002.

FREUND, J. E. **Estatística Aplicada**: **Economia, Administração e Contabilidade**. Tradução de Alfredo Alves de Farias. 9. ed. Porto Alegre: Editora Bookman,. 404 p. 2000.

R DEVELOPMENT CORE TEAM. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org. 2005.

VENABLES, W. N., SMITH, D. M. and the R DEVELOPMENT CORE TEAM. **An Introduction to R. Notes on R: Programming Environment for Data Analysis and Graphics.** Version 2.2.0. Áustria: 2005.