Uma Breve Introdução ao R

Eduardo Lemos, Ricardo Masuda, Samuel Vianna, Vitor Landi

Contents

1 Princípios Básicos

1.1 Primeiros Passos

R é uma linguagem orientada à objetos que são armazenados na memória ativa do computador. Uma variável é um objeto que irá representar um valor ou expressão atribuído a ela. Só é possível armazenar um dado ou expressão pra cada variável, quando for atribuído mais de uma informação, o dado que estava antes armazenado será subscrito.

1.1.1 Comandos Básicos

Primeiramente, para a melhor utilização do R, é necessário saber alguns comandos básicos. São eles:

- `control + L`: Limpar o console
- `control + R` ou `control + enter`: Compilar o código escrito
- `rm(list = ls())`: limpar memória
- `#`: fazer comentários no código

1.1.2 Atribuição de Valores

Pode-se atribuir um valor à um objeto dentro do ambiente do R de duas formas diferentes: `<-` e `=`.

Exemplos:

```
# atribuindo o valor 10 para a variavel x
x <- 10
x
```

```
## [1] 10
```

```
# atribuindo o valor 5 para a variavel y
y = 5
y
```

```
## [1] 5
```

Observação: Vale ressaltar que o sinal de igual é usado para a atribuição de valores, e não denotar igualdade, para isso é usado dois sinais (`==`).

1.1.3 Tipos de Variáveis

Toda variável declarada possui uma classe específica, de acordo com o seu conteúdo.

Para verificar a classe de uma determinada variável, utiliza-se a função `class`.

Exemplos:

```
# numérica
x <- 1.5
class(x)
```

```
## [1] "numeric"
```

```
# caractere: palavras, textos, etc
y <- "estatística"
class(y)
```

```
## [1] "character"
```

```
# lógico: TRUE, FALSE
z <- 4 < 5
class(z)
```

```
## [1] "logical"
```

1.1.4 Utilizando Ajuda (help)

Para buscar ajuda no R, pode-se usar a função `help()` ou o operador `?`.

Exemplos:

```
# Buscando ajuda sobre a função log

help(log)

?help
```

1.2 Operações Básicas

No ambiente R, existem uma série de operações básicas que são muito usuais e de grande importância. Tais como:

1.2.1 Operações simples

- `^`: Potencialização
- `/`: Divisão
- `*`: Multiplicação
- `+`: Adição
- `-`: Subtração

1.2.2 Operações lógicas

- `<`: Menor
- `<=`: Menor ou igual
- `>`: Maior
- `>=`: Maior ou igual
- `==`: Igual
- `!=`: Diferente
- `&`: AND
- `!`: NOT
- `|`: OR
- `FALSE ou 0`: Valor booleano falso (0)
- `TRUE ou 1`: Valor booleano verdadeiro (1)

1.2.3 Operações matemáticas

- `abs(x)`: Valor absoluto de x
- `log(x,b)`: Logaritmo de x com base b
- `log(x)`: Logaritimo natural de x
- `log10(x)`: Logaritmo de x na base 10
- `exp(x)`: Exponencial elevado a x
- `sin(x)`: Seno de x
- `cos(x)`: Cosseno de x
- `tan(x)`: Tangente de x
- `round(x, digits = n)`: Arredonda x com n decimais
- `ceiling(x)`: Arredonda x para o maior valor
- `floor(x)`: Arredonda x para o menor valor
- `sqrt(x)`: Raiz quadrada de x

1.3 Estruturas Básicas

1.3.1 Vetor

Um vetor é um conjunto de valores atribuidos à uma variável. Para criar um vetor, utiliza-se o comando `c()`.

Exemplos de vetores:

```
vetor1 <- c(1, 1, 2, 3, 5, 8)
idades <- c(17, 20, 22, 18, 30)
alunos <- c("Ricardo", "Samuel", "Vitor", "Ellen", "Mariana")
vetor2 <- c(0, vetor1, 0)
```

Existem funções que permitem criar e manipular vetores com características com maior facilidade, a seguir, estão algumas delas:

Sequências

Para criar um vetor baseado em uma sequência, pode-se usar a função `seq()`, que cria um vetor do valor A até o valor Z.

Exemplos:

```
# Criar um vetor de 1 a 10
vetor1 <- seq(from = 1, to = 10)
vetor1
```

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

```
#outra forma de criar o vetor de 1 a 10
vetor1.1 <- 1:10
```

Perceba que, por padrão, o intervalo entre os números gerados é 1. Porém, também pode-se alterar a distância entre os elementos (ou a "distância de passos"), com o argumento `by = N`, e a quantidade de elementos criados, com o argumento `length.out = N`.

Exemplos:

```
# Criar vetor de 1 a 10, com tamanho do passo = 2
vetor2 <- seq(from = 1, to = 10, by = 2)
vetor2
```

```
## [1] 1 3 5 7 9
```

```
# Criar vetor de 1 a 10, com 4 elementos
vetor3 <- seq(from = 1, to = 10, length.out = 4)
vetor3
```

```
## [1]  1  4  7 10
```

Operações em vetores

É possível aplicar uma série de operações em vetores, a seguir, algumas das operações mais utilizadas:

- `length(x)`: número de elementos do vetor x
- `sum(x)`: soma dos elementos do vetor x
- `prod(x)`: produto dos elementos do vetor x
- `max(x)`: seleciona o maior elemento do vetor x
- `min(x)`: seleciona o menor elemento do vetor x
- `range(x)`: retorna o menor e o maior elemento do vetor x

Criando vetores com a função `paste`

É possível também manipular vetores "colando" partes com a função `paste`.

Pode-se usá-lo para adicionar tanto um prefixo quanto um sufixo, usando as seguintes sintaxes:

- Prefixo: `paste("prefixo", vetor, sep = "separador")`
- Sufixo: `paste(vetor, "sufixo", sep = "separador")`

Exemplos:

```
x <- 1:10
# adicionando o prefixo "número", separando com "_"
paste("número", x, sep = "_")
```

```
##  [1] "número_1"  "número_2"  "número_3"  "número_4"  "número_5"  "número_6"
##  [7] "número_7"  "número_8"  "número_9"  "número_10"
```

```
# adicionando sufixo e atribuindo o resultado à variavel "y"
y <- c(paste(11:20, "número", sep = "%"))
y
```

```
##  [1] "11%número" "12%número" "13%número" "14%número" "15%número" "16%número"
##  [7] "17%número" "18%número" "19%número" "20%número"
```

Caso deseja-se adicionar um elemento "grudado" ao valor, pode-se tanto usar o argumento `sep=""` dentro da função `paste`, como a função `paste0`.

Exemplo:

```
# usando sep = ""
z <- c(paste("numero", 21:30, sep = ""))
z
```

```
##  [1] "numero21" "numero22" "numero23" "numero24" "numero25" "numero26"
##  [7] "numero27" "numero28" "numero29" "numero30"
```

```
#usando paste0
w <- c(paste0("numero", 21:30))
w
```

```
##  [1] "numero21" "numero22" "numero23" "numero24" "numero25" "numero26"
##  [7] "numero27" "numero28" "numero29" "numero30"
```

Repetições

É possível repetir um elemento ou um vetor com a função `rep()`. A seguir, alguns dos argumentos mais utilizados dentro da função:

- `times`: define o número de vezes que o número ou vetor inteiro será repetido
- `each`: define o número de vezes que cada elemento em um vetor será repetido
- `length.out`: define o tamanho do vetor de saída

Exemplos:

```
# repetindo um número 10 vezes
r1 <- rep(5, times = 10) # ou somente rep(5,10)
r1
```

```
##  [1] 5 5 5 5 5 5 5 5 5 5
```

```
x <- c("a", "b", "c")

# repetindo o vetor inteiro 5 vezes
rep(x, times = 5)
```

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

```r
# repetindo cada elemento do vetor 5 vezes
rep(x, each = 5)
```

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

```r
# criando um vetor de tamnaho 7
rep(x, length.out = 7)
```

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

Selecionando um elemento no vetor

Caso deseja-se saber qual o elemento se encontra em uma determinada posição de um vetor, denotada por i, pode-se localizá-lo utilizando a sintaxe `vetor[i]`

Vale ressaltar que a contagem é iniciada a partir do valor 1, diferente de certas linguagens de programação em que a contagem começa na posição 0.

Exemplo:

```r
# localizando o décimo terceiro número par entre 10 e 50
valores <- seq(10, 50, by = 2)
valores[13]
```

```
## [1] 34
```

1.3.2 Matriz

Uma matriz é uma generalização de um vetor, tendo duas dimensões (linhas e colunas). Podemos pensar em um vetor como uma matriz com uma de suas dimensões igual a 1. A sintaxe é dada abaixo, em que "L"é o número de linhas, "C"é o número de colunas e se "Q"= 1 ativa disposição por linhas, se "Q"= 0 mantém disposição por colunas (ou T ou F).

```
x <- matrix(data = dados, nrow = L, ncol = C, byrow = Q)
```

Exemplos:

```r
# Criando uma matriz de 2 linhas, 5 colunas e disposição por linhas

ml <- matrix(data = c(1:10), nrow = 2, ncol = 5, byrow = 1)
ml
```

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

```r
# Criando uma matriz de 2 linhas, 5 colunas e dispoção por colunas:
mc <- matrix(data = c(1:10), nrow = 2, ncol = 5, byrow = 0)
mc
```

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

Selecionando elemento da matriz

Para selecionar um elemento de uma matriz utilizamos a indexação por colchetes na variável que representa a matriz com os índices separados por vírgula.

Exemplos:

```r
# Selecionando a linha 2 e coluna 4 da matriz ml
ml[2,4]
```

```
## [1] 9
```

```
# Selecionando a linha 2 da matriz ml
ml[2,]
```

```
## [1]  6  7  8  9 10
```

```
# Selecionando as colunas 2,3 e 4 da matriz ml
ml[,2:4]
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    4
## [2,]    7    8    9
```

```
# Outra forma de ler a matriz ml
ml[,]
```

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

Operações de matrizes

`-A*B`: Produto elemento a elemento de A e B `-A% * &B`: Produto matricial de A por B `-apern(A)`: Matriz transposta de A `-t(A)`: Matriz transposta de A `-solve(A)`: Matriz inversa de A `-solve(A,B)`: Resolve o sistema linear Ax = B `-det(A)`: Retorna o determinante de A `-diag(v)`: Retorna uma matriz diagonal onde o vetor v é a diagonal `-diag(A)`: Retorna um vetor que é a diagonal de A `-diag(n)`: Sendo n um inteiro, retorna uma matriz identidade de ordem n `-eigen(A)`: Retorna os autovalores e autovetores de A

1.3.3 Array

Um array é uma generalização de uma matriz, em que os dados podem ser distribuidos em n dimensões de tamanhos ti, i E {1, 2, . . . , n}. A sintaxe utilizada é dada abaixo, em que "dim"é um vetor de dimensão do array.

```
x <- array(data = dados,dim = c())
```

Exemplos:

```
# Criando um array com dimensão de linhas e 5 colunas
a <- array(data = c(1:10), dim = c(2,5))
a
```

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

```
# Criando um array de 3 dimensões
b <- array(1:18, dim = c(2,3,3))
b
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
```

```
## [2,]    8   10   12
##
## , , 3
##
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
```

Selecionar um elemento do array O acesso dos elementos de um array é análogo ao de matriz e vetor, diferenciando no fato de que são informados n campos, considerando que são n dimensões.

Exemplo:

```
# Acessando um elemento do array b do exemplo anterior
b[1,2,3]
```

```
## [1] 15
```

1.3.4 Lista

Listas são estruturas genéricas e flexíveis que permitem armazenar diversos formatos em um único objeto.

`list(elemento1, elemento2, elementon)`

Exemplos:

```
# Criando vetores *s, **b*, e formando uma lista com esses vetores
s <- c("aa", "bb", "cc", "dd", "ee")
b <- c(TRUE, FALSE, TRUE, FALSE, FALSE)
x <- list(s, b, 3)
x
```

```
## [[1]]
## [1] "aa" "bb" "cc" "dd" "ee"
##
## [[2]]
## [1]  TRUE FALSE  TRUE FALSE FALSE
##
## [[3]]
## [1] 3
```

Operações com membros da lista

Para operações com membros da lista utiliza-se a sintaxe `lista[]`. A seguir, alguns exemplos utilizando a lista criada no exemplo anterior:

```
# Imprimindo o segundo membro da lista x:
x[2]
```

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

```
# Imprimindo o segundo e o terceiro membro da lista x

x[c(2,3)]
```

```
## [[1]]
## [1]  TRUE FALSE  TRUE FALSE FALSE
##
## [[2]]
## [1] 3
```

Operações com elementos dos membros da lista

Também pode-se realizar operações com elementos dentro de um membro da lista. Para isso, utiliza-se a seguinte sintaxe:

`lista[[membro]][elemento]`

Exemplos:

```
# Imprimindo o terceiro elemento do segundo membro da lista x

x[[2]][3]
```

```
## [1] TRUE
```

1.3.5 Data Frame

O data-frame é uma estrutura semelhante à uma matriz porém com cada coluna sendo tratada separadamente. Desta forma podemos ter colunas de valores numéricos e colunas de caracteres no mesmo objeto.

Dentro da mesma coluna todos elementos tem que ser do mesmo tipo. Cada vetor (coluna) tem que ter o mesmo número de observações.

A sintaxe é dada da seguinte forma:

`data.frame(elemento1 = x1,..., elementoN = xn)`

Exemplo:

```
n <- c(2, 3, 5)
s <- c(" aa ", " bb ", " cc ")
b <- c(TRUE, FALSE , TRUE)
t <- c(paste0("H" ,1:3) )

# Criando um data-frame df com elementos(vetores) n, s, b e t
df <- data.frame(n, s, b, t)
df
```

```
##   n    s     b  t
## 1 2  aa   TRUE H1
## 2 3  bb  FALSE H2
## 3 5  cc   TRUE H3
```

Pode-se alterar tanto o nome das linhas como o nome das colunas de um data frame, utilizando as funções `row.names()` e `col.names()`, respectivamente.

Exemplos:

```
# Dando nome as linhas do data-frame df
row.names(df) <- c("linha1","linha2","linha3")
df
```

```
##        n    s     b  t
## linha1 2  aa   TRUE H1
## linha2 3  bb  FALSE H2
## linha3 5  cc   TRUE H3
```

É possível selecionar um determinado elemento dentro de um data frame, selecionando uma linha e coluna especifica de duas formas diferentes:

```
# Selecionando observação da primeira linha e segunda coluna do data-frame
df[1,2]
```

```
## [1] " aa "
```

```
# Outra forma de selecionar a observação da primeira linha e segunda coluna do data-frame
df["linha1", "s"]
```

```
## [1] " aa "
```

De forma análoga, pode-se selecionar uma coluna inteira de um data frame. pode-se imprimir o output de duas formas: como um vetor e como coluna de um data frame.

Exemplos de output como vetor:

```
# Duas formas de imprimir um vetor com os elementos da terceira coluna
df[[3]]
```

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

```
df[["b"]]
```

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

```
# Outras formas de imprimir um vetor com elementos da terceira coluna
df$b
```

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

```
df[,"b"]
```

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

```
df[,3]
```

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

Exemplos de output como coluna:

```
# Imprimir apenas a terceira coluna
df[3]
```

```
##            b
## linha1  TRUE
## linha2 FALSE
## linha3  TRUE
```

```
df["b"]
```

```
##            b
## linha1  TRUE
## linha2 FALSE
## linha3  TRUE
```

```
# Imprimir apenas a segunda e a terceira coluna
df[c("b", "s")]
```

```
##            b    s
## linha1  TRUE  aa
## linha2 FALSE  bb
## linha3  TRUE  cc
```

Também é possível selecionar uma linha especifica de um data frame. Neste caso, o R retornará na mesma estrutura do data-frame apenas a linha específica.

Sintaxe:

```
dataframe[linha,]
```

Exemplos:

```
# Imprimir apenas a segunda linha
df[2,]
```

```
##        n    s     b  t
## linha2 3  bb  FALSE H2
```

```
df['linha2',]
```

```
##        n    s     b  t
## linha2 3  bb  FALSE H2
```

```
# Imprimir apenas a segunda e a terceira linha
df[c(2,3),]
```

```
##        n    s     b  t
## linha2 3  bb  FALSE H2
## linha3 5  cc   TRUE H3
```

```
df[c("linha2", "linha3"),]
```

```
##        n    s     b  t
## linha2 3  bb  FALSE H2
## linha3 5  cc   TRUE H3
```

1.4 Tabelas

Uma das formas de se visualizar facilmente os dados é por meio de tabelas, permitindo um olhar mais amplo e claro de um conjunto de informações.

1.4.1 Tabelas Simples

Para criar uma tabela simples de uma variavel, utiliza-se a função `table(variavel)`.

Exemplo:

```
# criando uma tabela simples para a variavel "sexo"
sexo <- c("F", "F", "F", "M", "M")
table(sexo)
```

```
## sexo
## F M
## 3 2
```

```
# criando uma tablea simples para a variavel "turma"
turma <- c(rep("A", 2), rep("B", 3))
table(turma)
```

```
## turma
## A B
## 2 3
```

1.4.2 Tabelas de Contingência

Uma tabela de contingência é usada quando se deseja cruzar informações sobre duas variáveis.

A sintaxe utilizada é semelhante à anterior: `table(variavel1, variavel2)`

Exemplo:

```r
# criando uma tabela de contingência para as variáveis "sexo" e "turma"
table(sexo, turma)
```

```
##     turma
## sexo A B
##    F 2 1
##    M 0 2
```

Observação: Para criar uma tabela de contingência com duas variáveis é necessário que ambas tenham o mesmo número de elementos.

1.4.3 Tabelas de Proporção

Quando é de interesse obter a frequência relativa das variáveis de uma tabela de contingência, utiliza-se a tabela de proporção, usando a seguinte sintaxe:

```
prop.table(X = tabela, margin = ...)
```

Observação: Para utilizar a função acima, é necessário que a tabela de contingência da qual se deseja obter as frequências já tenha sido criada anteriormente e atribuida à um objeto.

A opção `margin =` indica qual a marginal será utilizada na tabela, seguindo a seguinte ordem:

- `NULL`: proporção total
- `1`: proporção por linha
- `2`: proporção por coluna

Exemplo:

```r
#atribuindo a tabela ao elemento "tabela"
tabela <- table(sexo, turma)

# calculando a proporção total para a tabela anterior
# prop.table(tabela) ou
prop.table(tabela, margin = NULL)
```

```
##     turma
## sexo   A   B
##    F 0.4 0.2
##    M 0.0 0.4
```

```r
# calculando a proporção por linha
prop.table(tabela, margin = 1)
```

```
##     turma
## sexo         A         B
##    F 0.6666667 0.3333333
##    M 0.0000000 1.0000000
```

1.5 Funções

Funções são sequências de código definidas pelo usuário para executar uma sequência específica de comandos. É possível escrever funções no R através da seguinte sintaxe:

```r
funcao <- function(argumento1, argumento2, ...)
{
  sequencia de código utilizando os argumentos
}
```

Exemplos:

```r
# Criando uma função para elevar um argumento x ao quadrado
fx <- function(x){x^2}
fx(2)
```

```
## [1] 4
```

```r
y <- 1:10 # Criando um vetor com a sequencia de 1 a 10
fx(y)
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```r
x <- matrix(1:15, nrow=5, ncol=3) # Criando uma matriz 3x5
fx(x)
```

```
##      [,1] [,2] [,3]
## [1,]    1   36  121
## [2,]    4   49  144
## [3,]    9   64  169
## [4,]   16   81  196
## [5,]   25  100  225
```

1.6 Funções Apply

Funções da família apply são utilizadas para se aplicar outras funções em diferentes tipos de estruturas de dados, a aplicação das funções nessas estruturas é feita de forma iterativa, sem a necessidade de usar loops (como `while` ou `for`). Diferentes funções apply são usadas para diferentes estruturas de dados.

1.6.1 Apply

A função `apply` é utilizada em matrizes, –data frames– ou arrays. Ela retorna um vetor ou array dos valores obtidos aplicando a função argumento nos dados, sendo utilizada da seguinte forma:

```r
apply(X, Margem, Funcao)
```

Onde `X` representa os dados (array, matriz ou data.frame), `Margem` representa a margem que será utilizada na iteração (sendo `1` para linha, `2` para coluna), e `Funcao` representa a função a ser aplicada.

Exemplos:

```r
matriz <- matrix(1:16, 4, 4) # Criando uma matriz 4x4
matriz # Visualizando a matriz
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

Abaixo aplicamos a função `sum` (terceiro argumento) ao objeto `matriz` (primeiro argumento), na primeira marginal (segundo argumento), desta forma, a função `apply` retorna um vetor com as somas de cada linha da matriz.

```r
apply(matriz, 1, sum)
```

```
## [1] 28 32 36 40
```

Abaixo, usamos o segundo argumento como `2`, assim, a função retorna um vetor com as somas das colunas da matriz.

```r
apply(matriz, 2, sum)
```

```
## [1] 10 26 42 58
```

Também é possível utilizar funções criadas por usuários ou funções de outros pacotes.

Exemplo:

```r
apply(matriz, 1, function(x){x^2+0.5})
```

```
##       [,1]  [,2]  [,3]  [,4]
## [1,]   1.5   4.5   9.5  16.5
## [2,]  25.5  36.5  49.5  64.5
## [3,]  81.5 100.5 121.5 144.5
## [4,] 169.5 196.5 225.5 256.5
```

1.6.2 Tapply

A função `tapply` funciona da mesma forma que a função `apply`, mas podendo usar uma variável como indice de marginal. A função é utilizada da seguinte forma:

```r
tapply(X, Indice, Funcao)
```

Exemplo:

```r
dados <- data.frame(sexo = rep(c("M", "F"),
                               c(9, 11)),
                    idade = c(79, 2, 95, 22, 25, 73, 82, 23, 6, 19,
                              43, 39, 9, 88, 89, 41, 4, 13, 92, 33))

# Calculando a méria da idade de acordo com sexo
tapply(dados$idade, dados$sexo, mean)
```

```
##        F        M
## 42.72727 45.22222
```

1.6.3 Sapply

A função `sapply` é utilizada para aplicar funções em cada elemento de um objeto de tipo lista, utilizando a seguinte sintaxe:

```r
sapply(X, Funcao)
```

Onde `X` representa os dados (de tipo lista) e `Funcao` representa a função a ser aplicada em cada elemento dessa lista.

Exemplo:

Abaixo a função sapply retorna um vetor de comprimento `3`, onde cada elemento representa o resultado da função `mean` de cada elemento da lista.

```r
x <- 1:10 # Criado um vetor de sequencia de 1 a 10
y <- 2:14 # Criado um vetor de sequencia de 2 a 14
z <- 60:90 # Criado um vetor de sequencia de 60 a 90

lista <- list(x,y,z) # Criando uma lista com os objetos anteriores

sapply(lista, mean)
```

```
## [1]  5.5  8.0 75.0
```

1.6.4 Lapply

A função `lapply` funciona da mesma forma que `sapply`, porém, é retornado uma lista ao invés de um vetor dos resultados. A função é utilizada da seguinte forma:

```
lapply(X, Funcao)
```

Onde `X` representa os dados (de tipo lista) e `Funcao` representa a função a ser aplicada em cada elemento dessa lista.

Exemplo:

Abaixo a função lapply retorna uma lista de comprimento `3`, onde cada elemento representa o resultado da função `mean` de cada elemento da lista.

```
x <- 1:10 # Criado um vetor de sequencia de 1 a 10
y <- 2:14 # Criado um vetor de sequencia de 2 a 14
z <- 60:90 # Criado um vetor de sequencia de 60 a 90

lista <- list(x,y,z) # Criando uma lista com os objetos anteriores

lapply(lista, mean)
```

```
## [[1]]
## [1] 5.5
##
## [[2]]
## [1] 8
##
## [[3]]
## [1] 75
```

1.7 Pacotes

Como o R é **opensource**, a comunidade pode desenvolver e implementar novas funcionalidades que não estão presentes no pacote básico do R, chamadas de pacotes.

Os pacotes podem ser disponibilizados online e baixados pelos usuários dentro do programa, através da função `install.packages("nome do pacote")`.

Após baixar o pacote, é necessário carregá-lo para poder utilizar suas funcionalidades. Para isso, pode-se utilizar duas sintaxes:

`library(nome do pacote)` ou `require(nome do pacote)`

Observação: Para baixar o pacote é necessário que o nome esteja entre aspas, o que não é preciso para carregá-lo.

Exemplo:

```
#baixando e carregando o pacote gráfico ggplot2
install.packages("ggplot2")
library(ggplot2)
```

1.8 Entrada de Dados

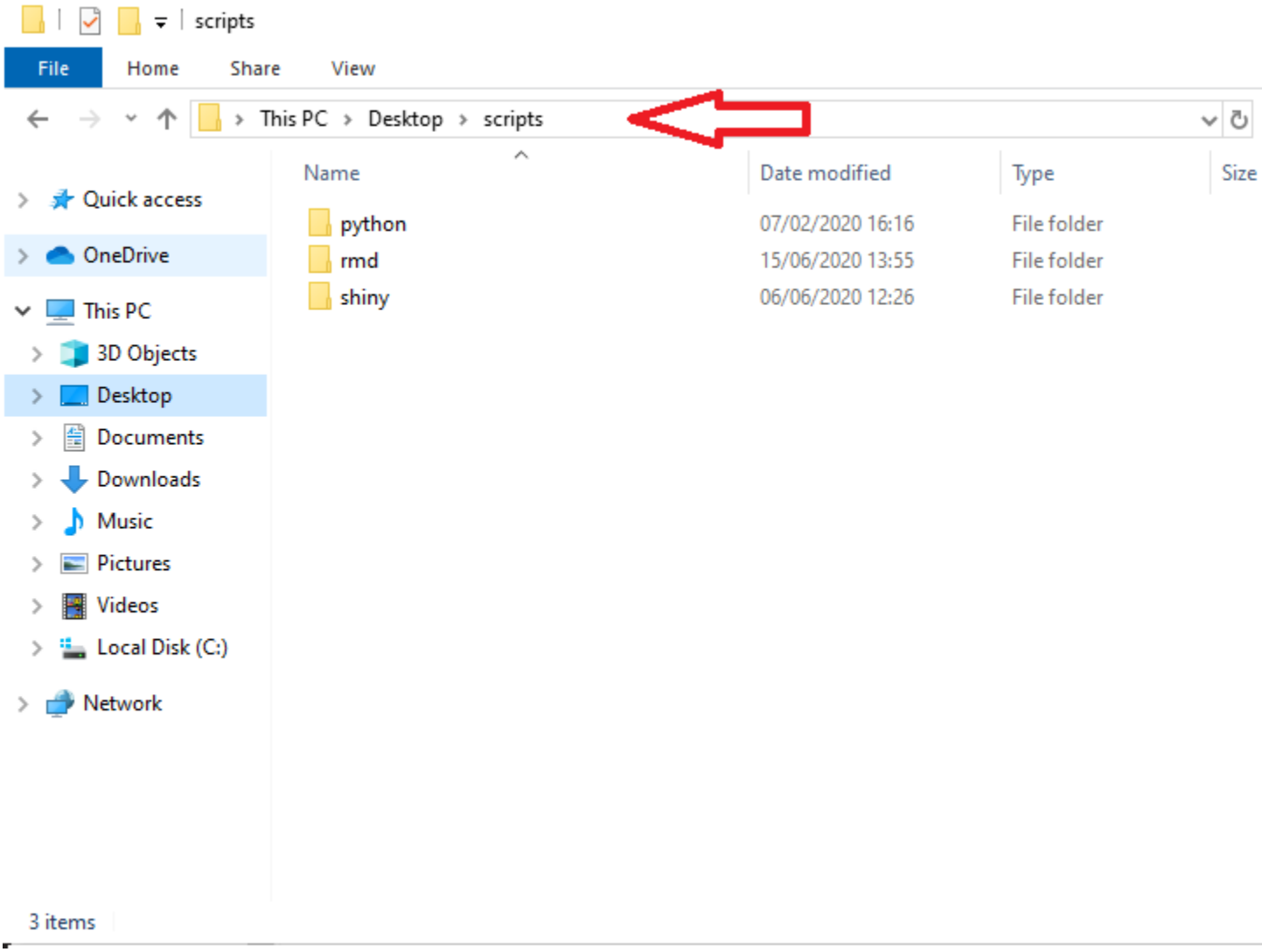
No que se refere à leitura de dados, existem meios para a leitura de bancos de dados externos, bem como carregar bancos disponíveis no próprio R.

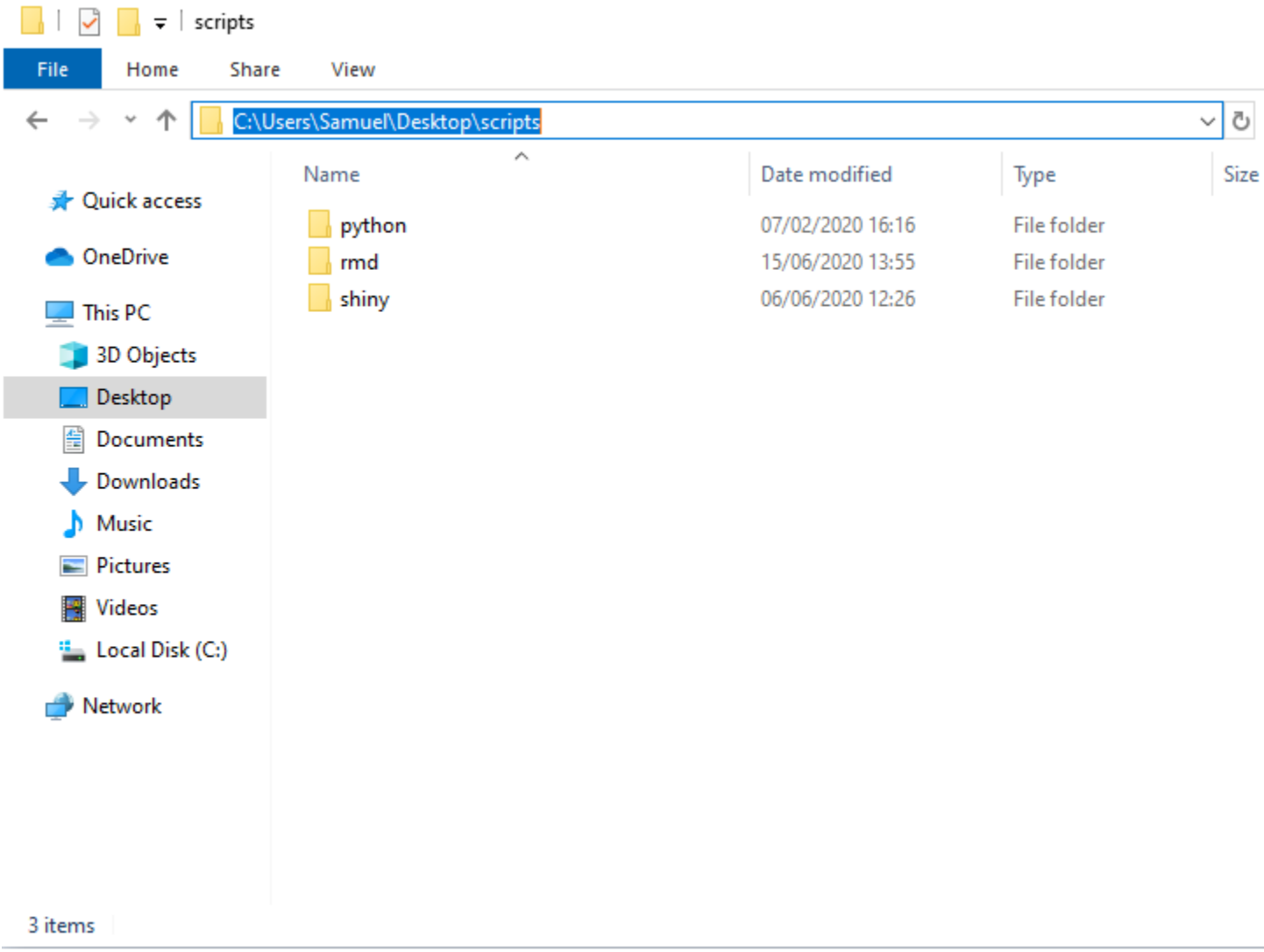
Para carregar bancos de dados externos, primeiro é necessário definir qual será o local (pasta) de trabalho. Para isso, usa-se:

```
setwd("C:\\Usuário\\Local")
```

Observação: Note que é preciso utilizar aspas para indicar o local e também barras duplas (\\) ou barras invertidas (/) para separação.

No Windows, para encontrar o "caminho" da pasta que desja, basta clicar no local indicado na figura a seguir:





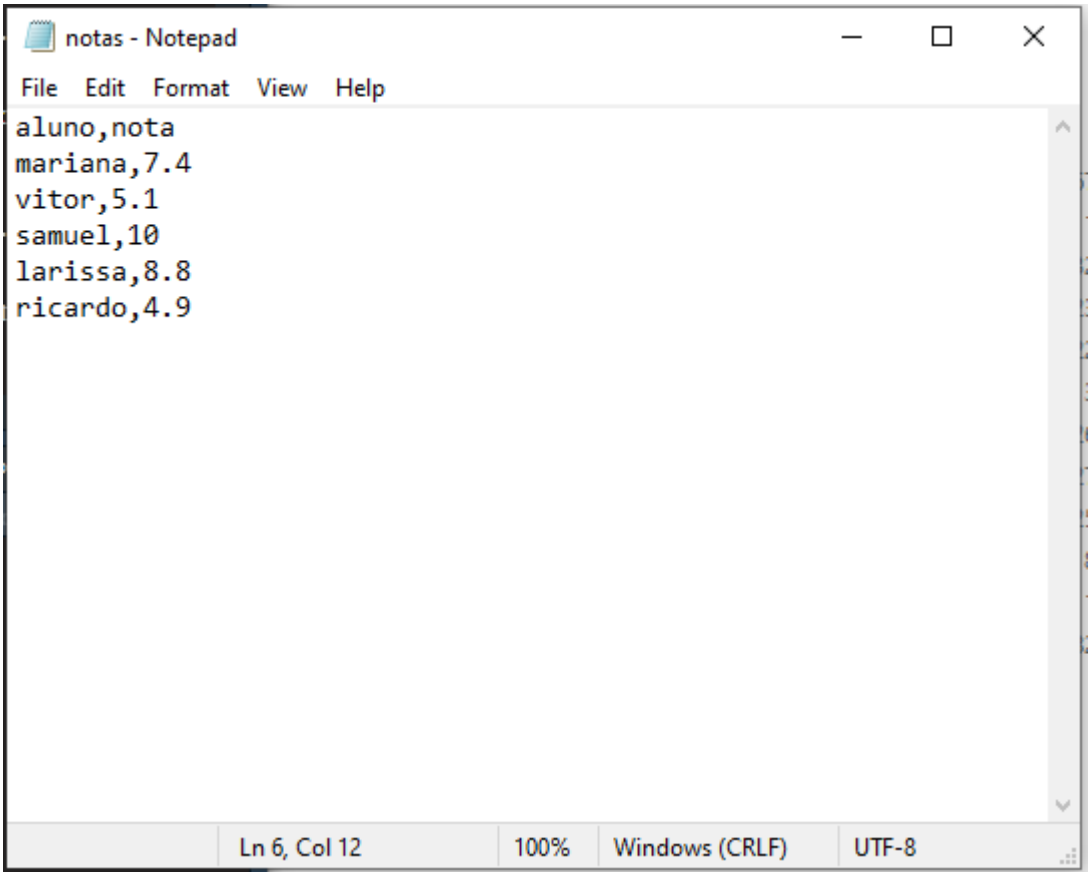
Para verificar qual o local que está definido como pasta de trabalho, utiliza-se a função `getwd()`.

1.8.1 .csv ou .txt

Para ler arquivos do tipo `.csv`(excel) ou `.txt`, pode-se utilizar a função `read.table("dados.csv, ...)`, se atentando para os seguintes detalhes dentro da função:

- `header`: indica se há ou não cabeçalho dentro do banco de dados
- `sep`: indica qual o separador entre as células do banco (indicada entre aspas)
- `dec`: indica qual o indicador decimal das unidades numéricas do banco (indicada entre aspas)

A seguir, vamos ler o banco de dados `notas.txt` como exemplo:



Após definir o local de trabalho, verifica-se que o banco de dados possui cabeçalho, está com os elementos separados por vírgulas, e usa como separador decimal o ponto, com isso, temos:

```
##     aluno nota
## 1 mariana  7.4
## 2   vitor  5.1
## 3  samuel 10.0
## 4 larissa  8.8
## 5 ricardo  4.9
```

Observação: Para arquivos `.csv`, geralmente são utilizados dois separadores: `,` e `;`.

Observação: Para a melhor leitura de arquivos do excel do tipo `.xls` ou `.xlsx` é necessário o uso de pacotes.

1.8.2 Leitura de dados da internet

Também é possível ler arquivos da Web, indicando o endereço (URL) dentro da faunção.

Exemplo:

```
# lendo um banco de dados da Web sobre gatos dosméticos
gatos <- read.table("https://vincentarelbundock.github.io/Rdatasets/csv/boot/catsM.csv", sep = ",", hea
```

1.8.3 Base de dados do R

O R base já vem com alguns bancos de dados que são usados para aprendizado ou exemplificação.

Para ver todas as opções disponíveis utiliza-se a função `data()`.

Para utilizá-los basta atribuirmos o nome do banco escolhido a algum objeto, ou usando pelo próprio nome no R.

```
# lendo o banco de dados iris
data(iris)
```

1.8.4 Conferência dos dados

É recomendável conferir a importação dos dados para evitar erros futuros na análise. Existem comandos utilizados para isso, tais como:

- `head()`: imprime as primeiras observações de um banco de dados
- `tail()`: imprime as últimas observações de um banco de dados
- `View()`: mostra todo o banco de dados em outra janela
- `str()`: imprime o tipo e dimensão de cada variável do banco de dados

Exemplo:

```
# mostrando as primeiras linhas do banco 'iris'
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# mostrando as 3 primeiras linhas do banco 'iris'
head(iris, 3)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
```

```
# mostrando as últimas linhas do banco 'gatos'
tail(gatos)
```

```
##     X Sex Bwt  Hwt
## 92 92   M 3.6 15.0
## 93 93   M 3.7 11.0
## 94 94   M 3.8 14.8
## 95 95   M 3.8 16.8
## 96 96   M 3.9 14.4
## 97 97   M 3.9 20.5
```

```
#mostrando as variáveis do banco 'gatos'
str(gatos)
```

```
## 'data.frame':    97 obs. of  4 variables:
##  $ X  : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Sex: chr  "M" "M" "M" "M" ...
##  $ Bwt: num  2 2 2.1 2.2 2.2 2.2 2.2 2.2 2.2 2.2 ...
##  $ Hwt: num  6.5 6.5 10.1 7.2 7.6 7.9 8.5 9.1 9.6 9.6 ...
```

1.8.5 Funções Adicionais

É possível facilitar o acesso às colunas de uma base de dados, isto é, ao invés de utilizar o operador $ para acessar alguma coluna pode-se utilizar o seu próprio nome. Para fazer isso, utiliza-se o comando `attach(dados)`.

Esse comando irá trazer para a memória do computador cada coluna como um objeto, logo não é recomendável fazer isso com tanta frequência e com uma quantidade grande de dados. Para desfazer esse anexo de dados na memória, usa-se `detach(dados)`.

2 Estatística Básica

2.1 Medida de posição

As medidas descritivas são úteis para a representação dos dados a serem trabalhados e normalmente precedem qualquer análise estatística.

2.1.1 Média

Para se calcular a média de um conjunto de dados utiliza-se a função `mean()`.

Exemplo :

```
{r}
x <- 1:11
mean(x)
```

2.1.2 Mediana

Para se calcular mediana de um conjunto de dados utiliza-se a função `median()`.

Exemplo:

```
{r}
x <- 11:20
median(x)
```

2.1.3 Mínimo e máximo

Assim como quase tudo no R, existe mais de uma maneira de se calcular o mínimo e o máximo em um conjunto de dados. A primeira maneira é utilizando as funções `min()` e `max()`.

Exemplo:

```
{r}
x <- 1:100
min(x)

max(x)
```

A outra forma é utilizando a função `range()`, mas neste caso o R retornará um vetor de 2 posições, sendo o mínimo representado pelo primeiro elemento e o máximo pelo segundo.

Exemplo:

```
{r}
x <- 1:100
range(x)
```

2.1.4 Quantis

Para se obter os quantis de um conjunto de dados utiliza-se a função `quantiles()`.

É possível indicar quais são os quantis de interesse dentro da função. Por padrão, a função retorna os quantis de 0%, 25%, 50%, 75% e 100%, mas é possível achar os qualquer quantil de interesse em uma sequência de valores.

Exemplo:

```
{r}
x <- 1:11 + c(rep(1.2,4),rep(2.3,5),rep(4.3,2))
quantile(x)

quantile(x,c(.05,.95))
```

2.1.5 Moda

No R base não existe uma função específica para se calcular a moda, isto é, o valor mais frequente no conjunto. Então, uma sugestão para se verificar esse valor mais frequente é criar uma tabela de frequência através do comando `table()`, visto anteriormente, e investigar nessa tabela o valor que mais se repete.

Observação 1: Note que este método só é válido para dados de natureza quantitativa discreta. *Observação 2*: Utiliza-se a função `which.max()` para retornar a posição da tabela com o máximo da frequência.

Exemplo:

```
{r}
x=c(2,1,2,2,1,4,4,5,2,6,5,3,2,4,1,6)
tb=table(x)
tb
which.max(tb)
```

2.1.6 Função Summary

A função `summary()` é uma função genérica dentro do R que, quando a aplicada a objetos da classe de vetor numérico, retorna algumas medidas de posição.

Exemplo:

```
{r}
x=c(1.3,1.5,1.3,1.7,1.9,2,1.4)
summary(x)
```

2.2 Medidas de Dispersão

2.2.1 Variância

Uma medida muito importante para se entender o comportamento dos dados é a variância, calculada através da fórmula `var(dados)`.

Exemplo:

```
x <- c(1, 3.5, 7, 5.4, 10)
var(x)
```

```
## [1] 11.672
```

2.2.2 Desvio Padrão

Sabendo que o desvio padrão é a raiz quadrada da variância, é possível achá-lo aplicando a função de raiz quadrada, vista nas sessões anteriores, na função de variância. A sintaxe fica da seguinte forma: `sqrt(var(x))`.

Uma outra forma de se calcular o desvio padrão é utilizando a função `sd(dados)`.

Exemplo:

```
x <- c(1, 3.5, 7, 5.4, 10)

# calculando o desvio padrão atráves da raiz da variância
sqrt(var(x))
```

```
## [1] 3.416431
```

```
# calculando o desvio padrão através da função sd
sd(x)
```

```
## [1] 3.416431
```

2.2.3 Coeficiente de Variação

Para se ter uma métrica da variabilidade relativa dos dados é interessante vermos o coeficiente de variação, dada pela divisão do desvio padrão pela média vezes 100. Escrevendo essa fórmula no R obtém-se a seguinte sintaxe:

```
(sd(dados)/mean(dados)) *100
```

Exemplo:

```
# calculando o coeficiente de variação para o vetor anterior
(sd(x)/mean(x)) *100
```

```
## [1] 63.50243
```

2.2.4 Amplitude

Para se calcular a amplitude basta subtrair o valor mínimo do máximo presente no conjunto de dados.

Exemplo:

```
# calculando a amplitude do vetor anterior
amp <- max(x) - min(x)
amp
```

```
## [1] 9
```

2.3 Correlação

Quando tratamos de variáveis quantitativas, principalmente contínuas, é interessante ter uma medida explicativa a respeito da quantidade de variabilidade compartilhada entre duas variáveis. Essa medida é usualmente o coeficiente de correlação, um valor entre -1 e 1 que explica a força (proximidade de |1|) e direção (positiva ou negativa) da correlação.

Há 3 abordagens diferentes para se calcular essa medida(Pearson, Kendall e Spearman). Em todos os casos, utiliza-se a função `cor()`.

Observação: Caso não seja especificado na função qual método será utilizado, é aplicado o método de pearson.

2.3.1 Coeficiente de Pearson

Para se calcular o coeficiente de Pearson usa-se o comando `cor(..., method = "pearson")`

```
v1 <- 1:7
v2 <- c(2, 6, 7, 9, 3, 5, 1)

cor(v1, v2, method = "pearson")
```

```
## [1] -0.2419215
```

2.3.2 Coeficiente de Kendall

Para se calcular o coeficiente de Kendall usa-se o comando `cor(..., method = "kendall")`

```
v1 <- c(1, 2, 3, 7, 9, 4, 6)
v2 <- c(2, 6, 7, 9, 3, 5, 1)

cor(v1, v2, method = "kendall")
```

```
## [1] 0.04761905
```

2.3.3 Coeficiente de Spearman

Para se calcular o coeficiente de Spearman usa-se o comando `cor(..., method = "spearman")`

```
v1 <- 11:20
v2 <- 20:11

cor(v1, v2, method = "spearman")
```

```
## [1] -1
```

3 Gráficos

3.1 Gráficos para variáveis qualitativas

3.1.1 Gráficos de Barras

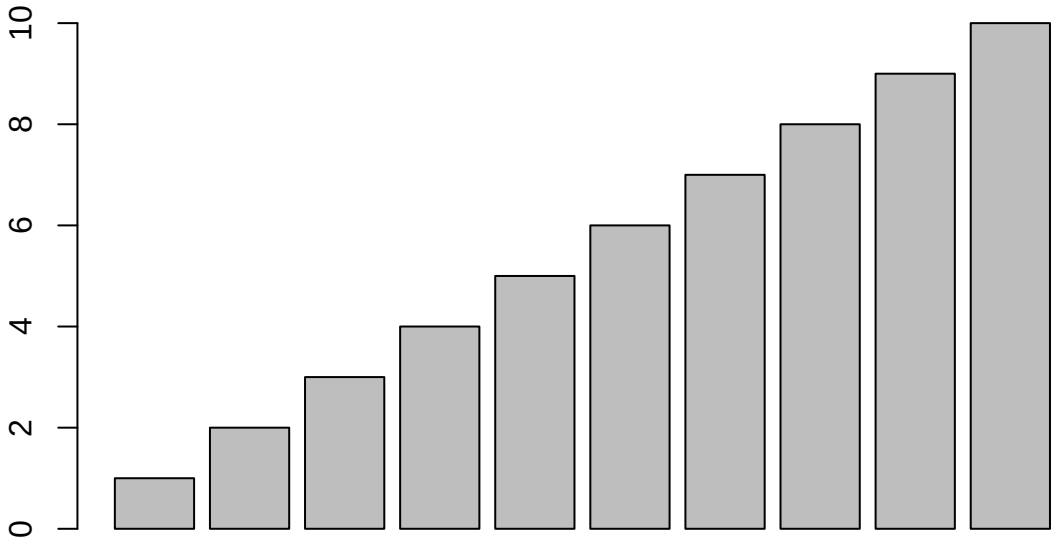
Para criar gráficos de barras no `R`, podemos utilizar a seguinte sintaxe `barplot(dados)`.

Lembrando que o argumento `dados`, deve ser um objeto de tipo vetor ou matriz.

É recomendado utilizar a função `table` ao gerar vetores para gerar gráficos de barras.

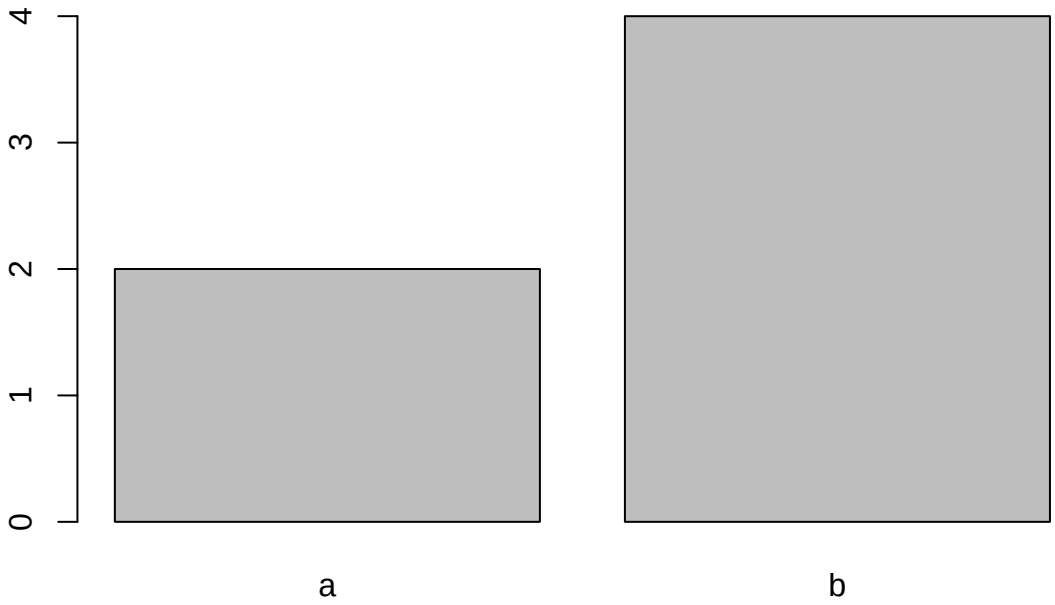
No exemplo abaixo, são criadas 10 barras não nomeadas com valores de 1 a 10.

```r
barplot(1:10)
```



No exemplo abaixo, é utilizado um vetor com elementos nomeados (`a` e `b`), quando geramos um vetor com a função `table` temos o mesmo resultado, e com isso as barras no gráfico também são nomeadas.

```r
barplot(c("a" = 2, "b" = 4))
```



É importante lembrar que isso não se aplica a objetos de tipo lista, como no código abaixo.

```r
barplot(list("a" = 2, "b" = 4))
```

```
## Error in -0.01 * height: non-numeric argument to binary operator
```

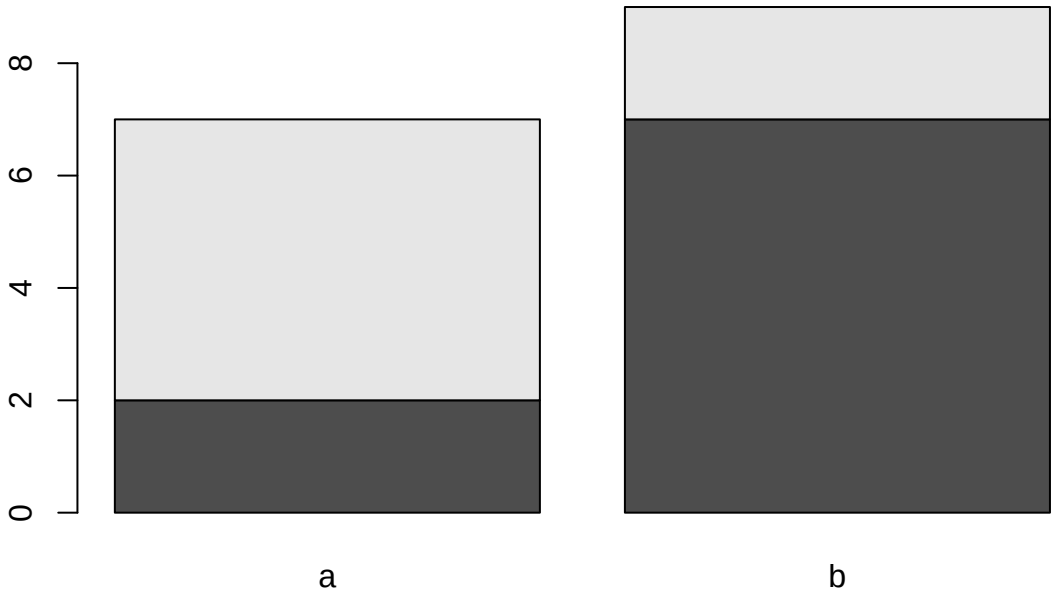
No exemplo abaixo, vemos como objetos de tipo matriz podem ser usados dentro da função `barplot`.

```r
matriz <- matrix(c(2, 5, 7, 2), 2, dimnames = list(c("n1", "n2"), c("a", "b")))
matriz
```

```
##    a b
## n1 2 7
```

```
## n2 5 2
```

```
barplot(matriz)
```

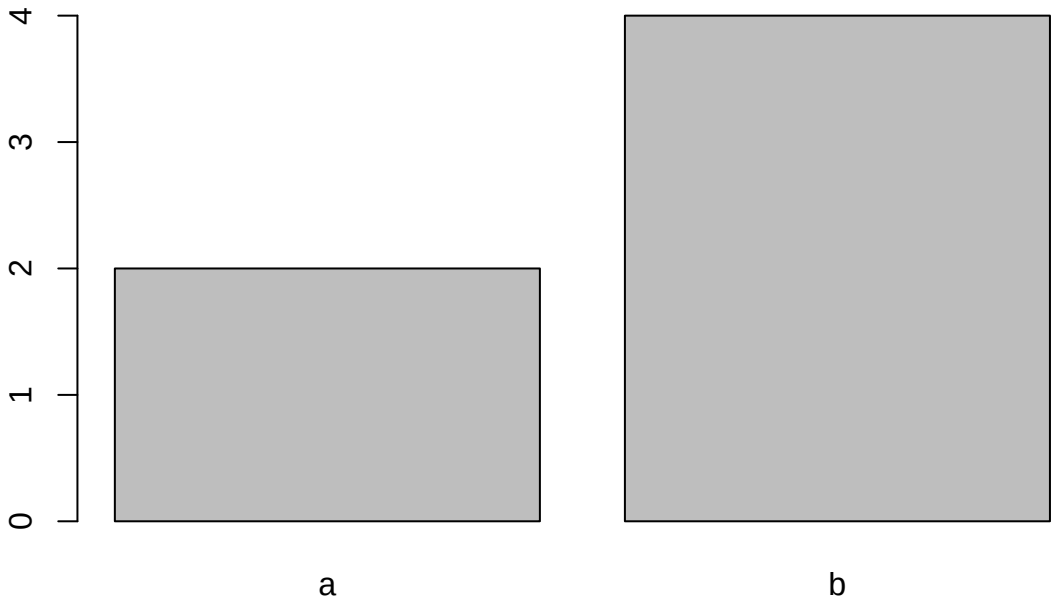


Abaixo, vemos como tabelas podem ser usadas para gerar o gráfico de barras.

```
tabela <- table(c(rep("a", 2), rep("b", 4)))
tabela
```

```
##
## a b
## 2 4
```

```
barplot(tabela)
```



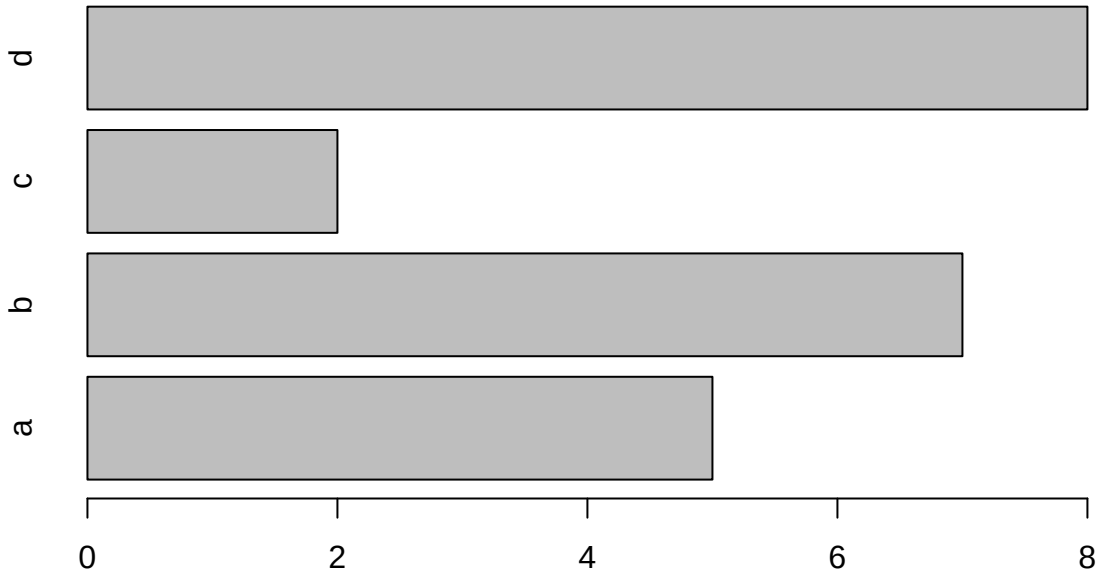
Também é possível utilizar a função `barplot` para gerar um gráfico de colunas horizontais com o argumento `horiz = TRUE`.

Abaixo, vemos também a função `table` sendo usada em conjunto com `data.frame`'s.

```
dados <- data.frame(letra = c(rep("a", 5), rep("b", 7),
                              rep("c", 2), rep("d", 8)))
dados
```

```
##    letra
## 1      a
## 2      a
## 3      a
## 4      a
## 5      a
## 6      b
## 7      b
## 8      b
## 9      b
## 10     b
## 11     b
## 12     b
## 13     c
## 14     c
## 15     d
## 16     d
## 17     d
## 18     d
## 19     d
## 20     d
## 21     d
## 22     d
```

```
barplot(table(dados$letra), horiz = TRUE)
```



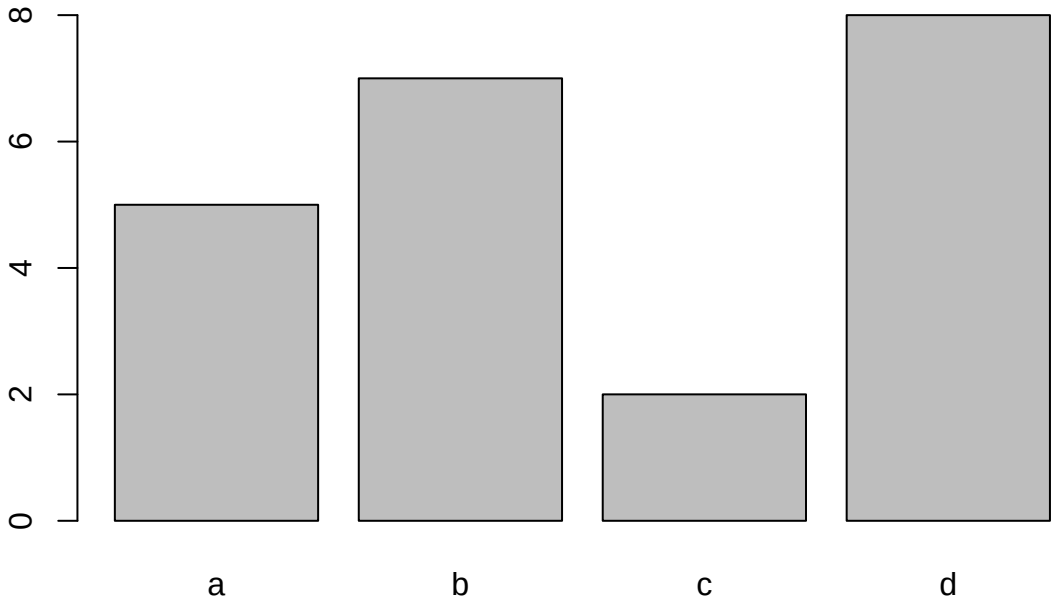
Utilizando `data.frame`'s, podemos usar outra coluna para nomear nossas colunas utilizando o argumento `names.arg`.

```
dados <- data.frame(letra = c("a", "b", "c", "d"), numero = c(5, 7, 2, 8))
dados
```

```
##   letra numero
```

```
## 1     a      5
## 2     b      7
## 3     c      2
## 4     d      8
```

```r
barplot(dados$numero, names.arg = dados$letra)
```

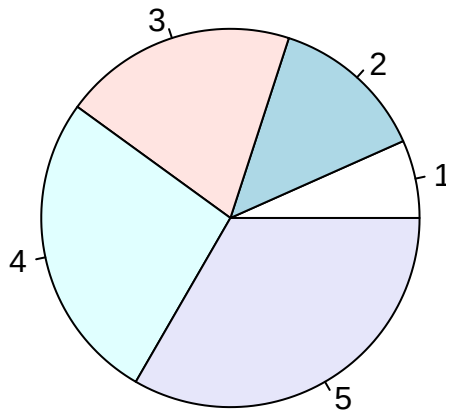


3.1.2 Gráficos de Setores

Para criar gráficos de setores no `R`, podemos utilizar a seguinte sintaxe `pie(dados)`, onde `dados` é um vetor numérico.

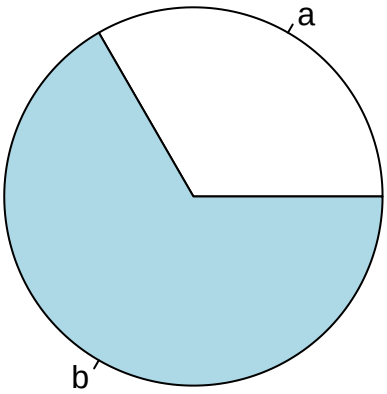
No exemplo abaixo, é criado um gráfico de 5 setores com valores de 1 a 5.

```r
pie(1:5)
```



No exemplo abaixo, é utilizado um vetor com elementos nomeados (`a` e `b`), quando geramos um vetor com a função `table` temos o mesmo resultado, e com isso os setores no gráfico também são nomeadas.

```r
pie(c("a" = 2, "b" = 4))
```



É importante lembrar que isso não se aplica a objetos de tipo lista, como no código abaixo.

```
pie(list("a" = 2, "b" = 4))
```

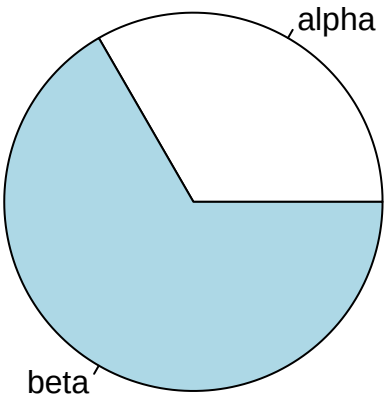
Da mesma forma que com gráficos de barras, podemos utilizar tabelas para gerar o gráfico de setores.

No gráfico de setores também é possível modificar o raio do círculo com o argumento `radius`.

```
tabela <- table(c(rep("alpha", 2), rep("beta", 4)))
tabela
```

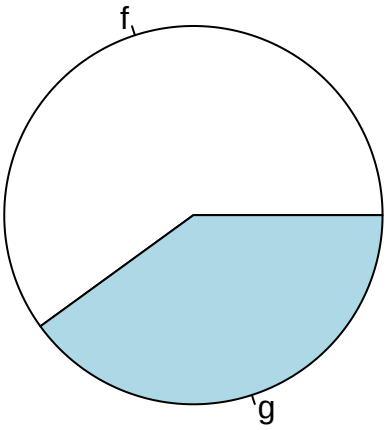
```
##
## alpha  beta
##     2     4
```

```
pie(tabela, radius = 0.8)
```



Também é possível utilizar `data.frame`'s para gerar o gráfico.

```
dados <- data.frame(letra = c("f", "f", "f", "g", "g"))
pie(table(dados$letra))
```

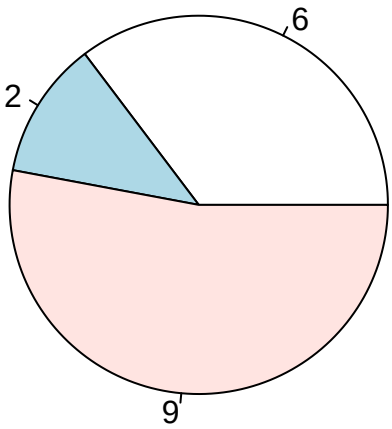


Com gráficos de barras, foi visto como nomear as colunas, no gráfico de setores, também é possível atribuir nomes aos setores utilizando o argumento `labels`.

```
dados <- data.frame(letra = c("f", "g", "h"), numero = c(6, 2, 9))
dados
```

```
##   letra numero
## 1     f      6
## 2     g      2
## 3     h      9
```

```
pie(dados$numero, labels = dados$numero)
```



3.1.3 Histograma

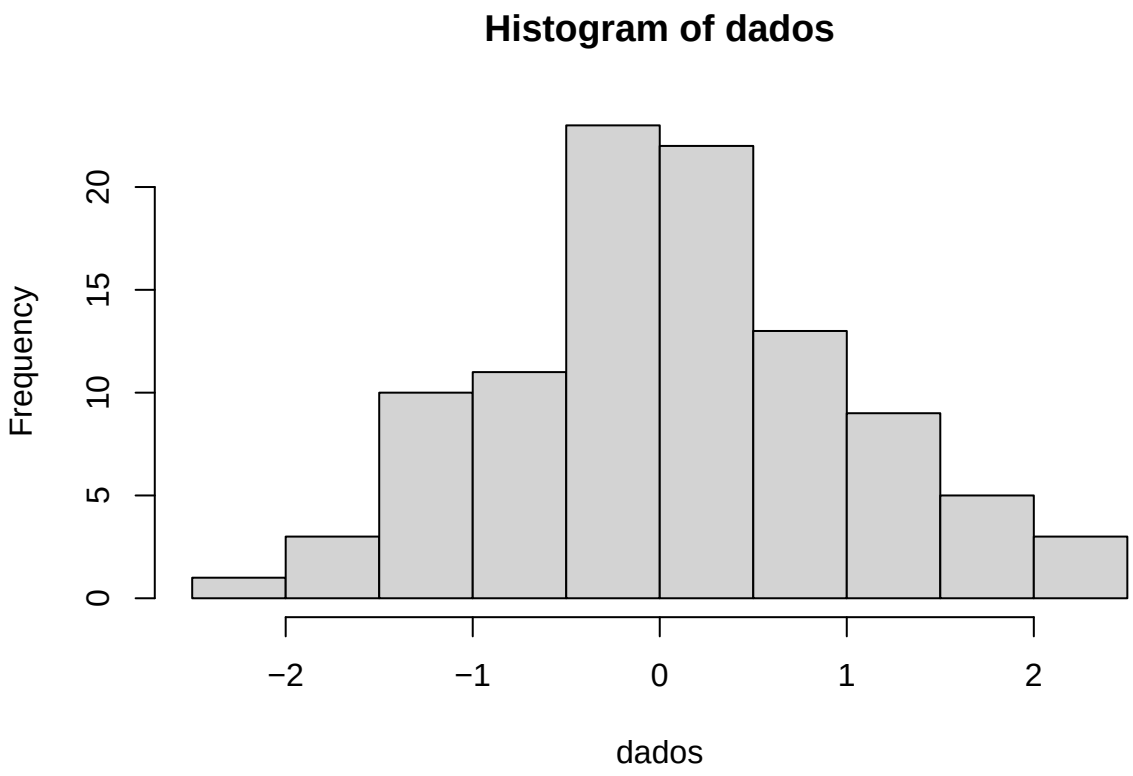
A forma mais utilizada para se representar variáveis quantitativas contínuas é através do histograma.

São dados os comandos abaixo para a construção de um histograma.

`hist(variável)`

Exemplo:

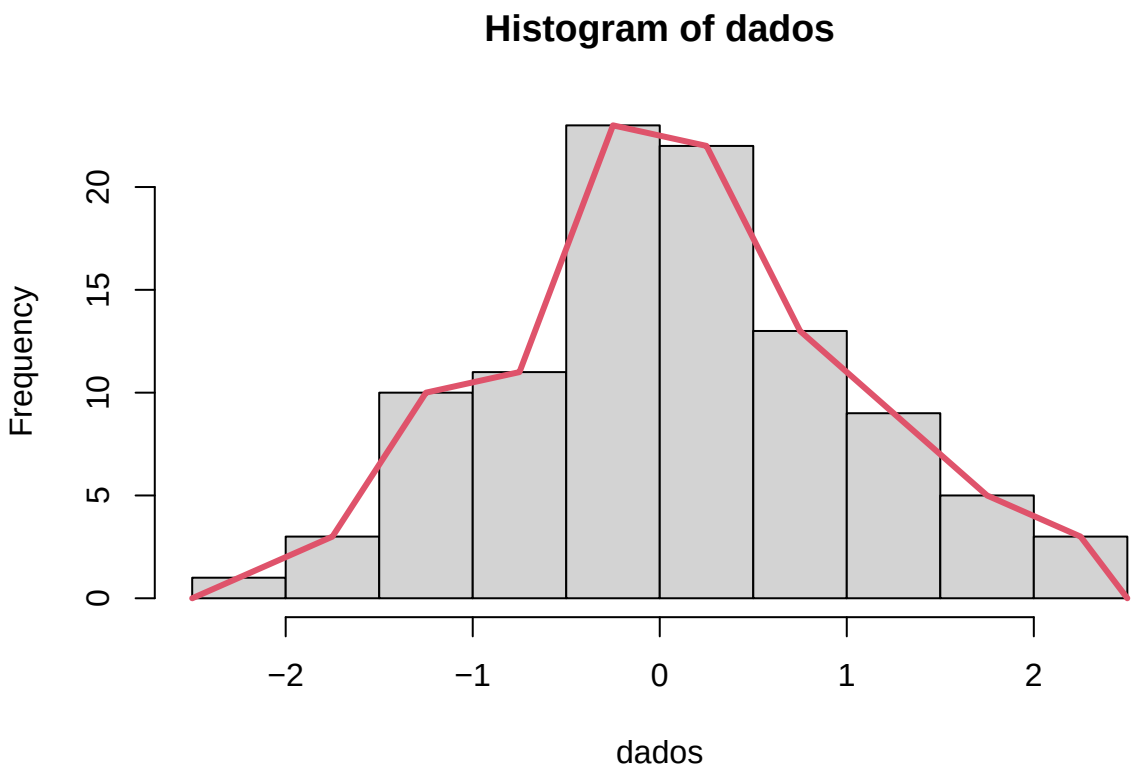
```
set.seed(123)
dados <- rnorm(100)
hist(dados)
```



3.1.4 Polígono de frequência

Para construirmos o polígono de frequência vamos utilizar outras funções gráficas: A função lines() sobrepõe o gráfico com alguma linha, para a qual daremos as coordenadas x e y. No caso x são os pontos médios do histograma e y é a frequência absoluta.

```
set.seed(123)
dados <- rnorm(100)
histograma <- hist(dados)
lines(x=c(histograma$breaks[1], histograma$mids, histograma$breaks[length(histograma$breaks)]),
      y=c(0, histograma$counts,0), col = 2, lwd = 3)
```



Note que atribuímos o histograma a um objeto para conseguir acessar as informações mids (pontos médios) , counts (frequência absoluta) e breaks (intervalos de classe).

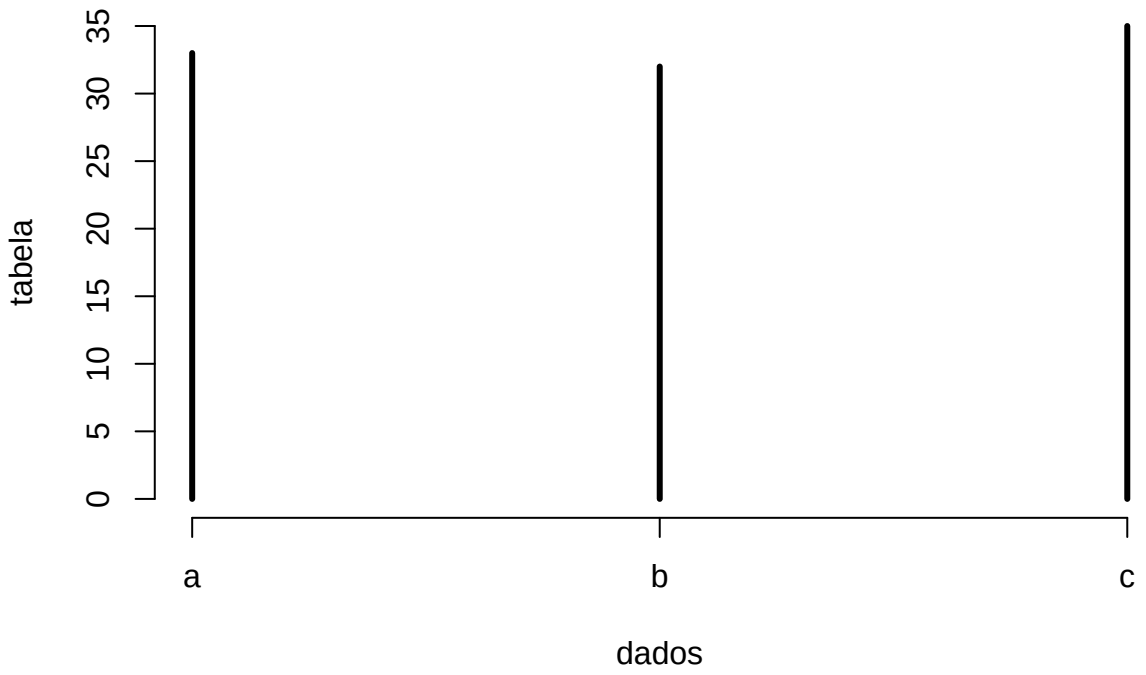
OBS: Um polígono de frequência deve começar na primeira classe e terminar na última, portando o X está variando do primeiro elemento do vetor `histogram$breaks` até o último elemento desse mesmo vetor. O argumento col= informa a cor e lwd= informa a espessura da linha.

3.1.5 Gráfico de bastões (hastes)

Para se representar variáveis de natureza quantitativa discreta é comum utilizarmos o gráfico de bastões ou hastes. Para fazê-lo basta utilizar a função genérica `plot()` nos dados em forma de tabela e ajustar alguns argumentos.

Exemplo:

```
set.seed(123)
dados <- sample(c("a", "b", "c"), 100, TRUE)
tabela <- table(dados)
plot(tabela, type = "h", lwd = 3)
```



A opção type= informa o tipo de gráfico que queremos. Nesse caso, h é o tipo bastão. Entraremos em mais detalhes sobre isso em breve.

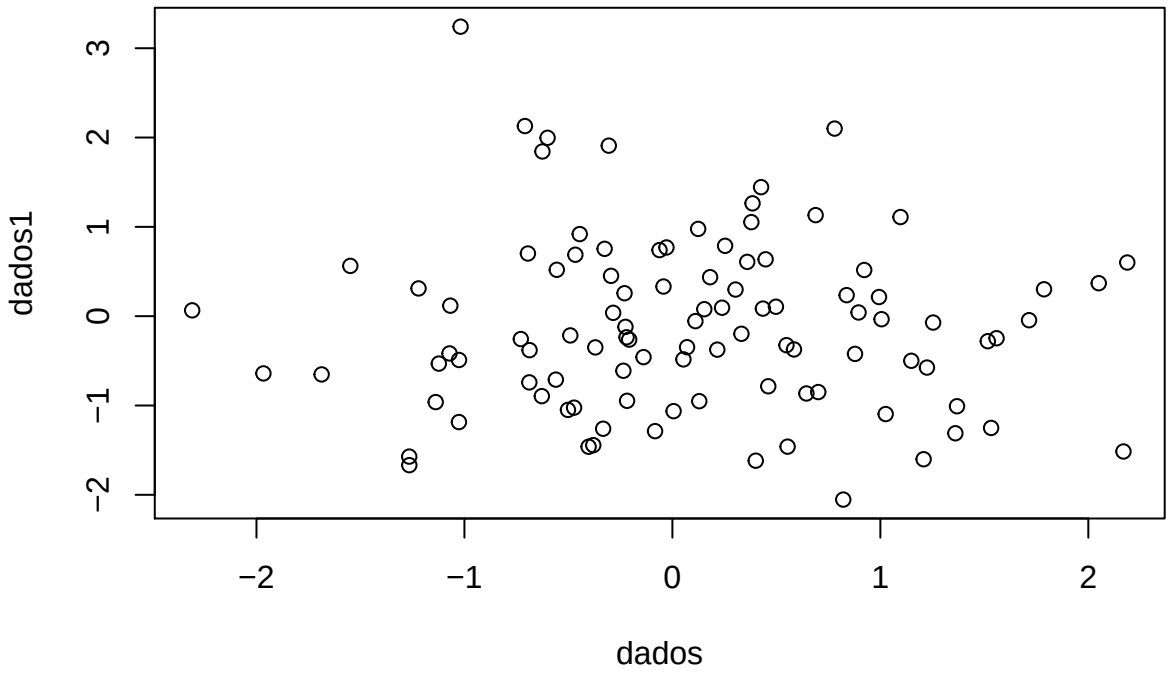
3.1.6 Gráfico de Dispersão

Quando tratamos de duas variáveis quantitativas é interessante observar o comportamento conjunto entre elas. A maneira mais usual para se verificar esse comportamento é o gráfico de dispersão. Para construirmos esse gráfico basta utilizarmos a função genérica plot() do R, dessa forma o comando fica:

```
plot(variável1, variável2)
```

Exemplo

```
set.seed(123)
dados <- rnorm(100)
dados1 <- rnorm(100)
plot(dados, dados1)
```



3.1.7 Gráfico de caixas (Box-plot)

Quando tratamos de variáveis quantitativas é interessante observar, a distribuição dos dados de um modo geral. Uma forma de se verificar isso é pelo gráfico de Caixas, em que conseguimos visualizar os quartis, mediana, limites superior e inferior e outliers.

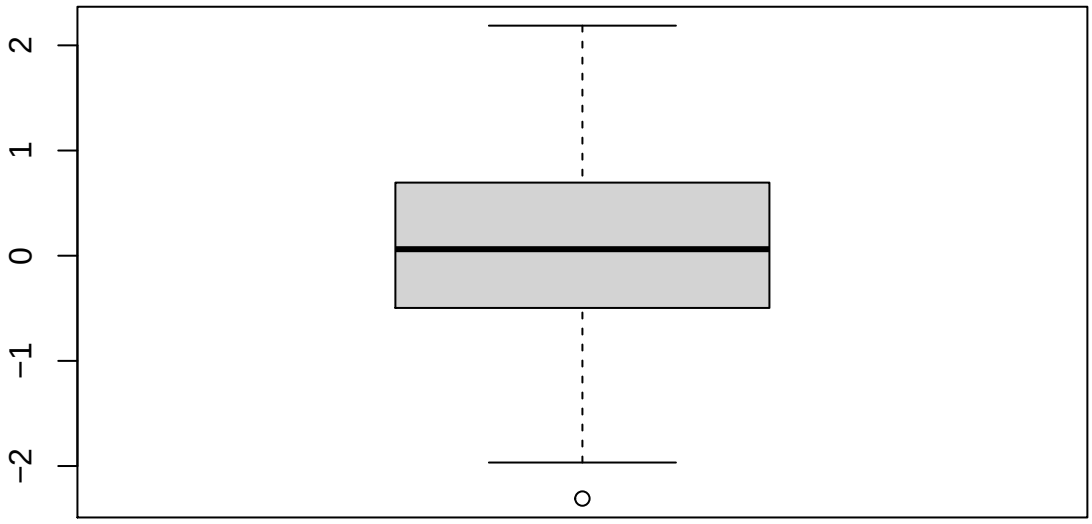
Para construirmos esse gráfico basta utilizarmos a função:

```
boxplot(variável)
```

Exemplos:

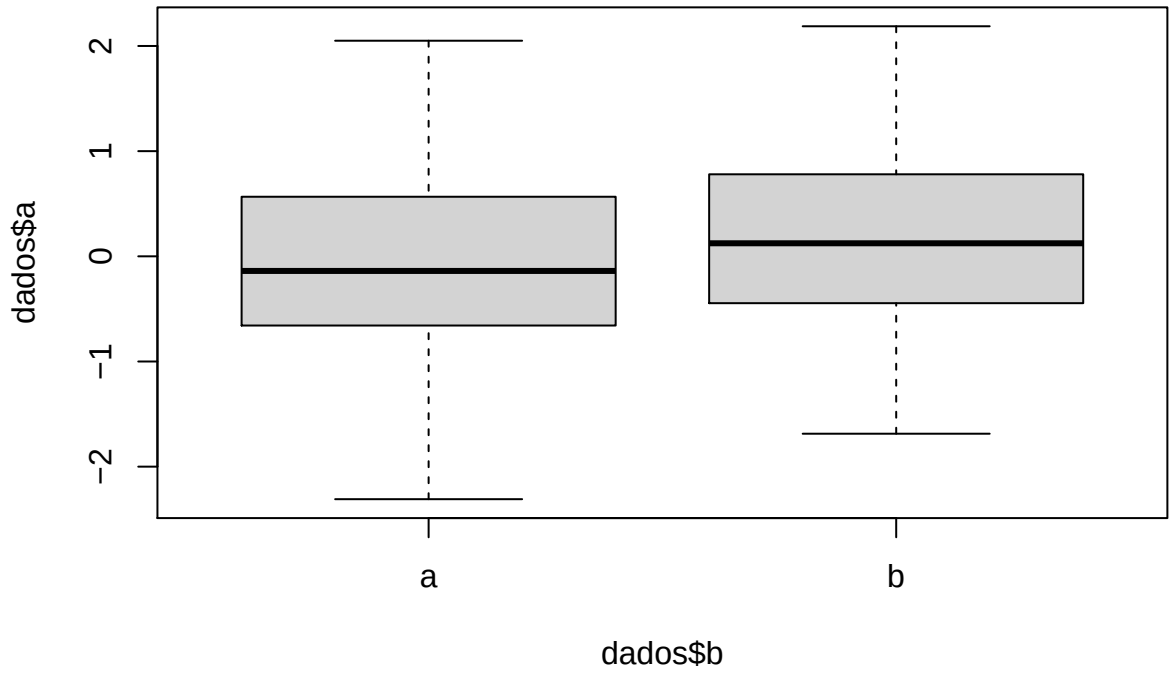
```
set.seed(123)
dados <- data.frame(a = rnorm(100), b = sample(c("a", "b"), 100, TRUE))

boxplot(dados$a)
```



```r
# Criando um Box-plot para a variável altura conforme a variável Sexo.

boxplot(dados$a~dados$b)
```



3.2 Ajustes Gráficos

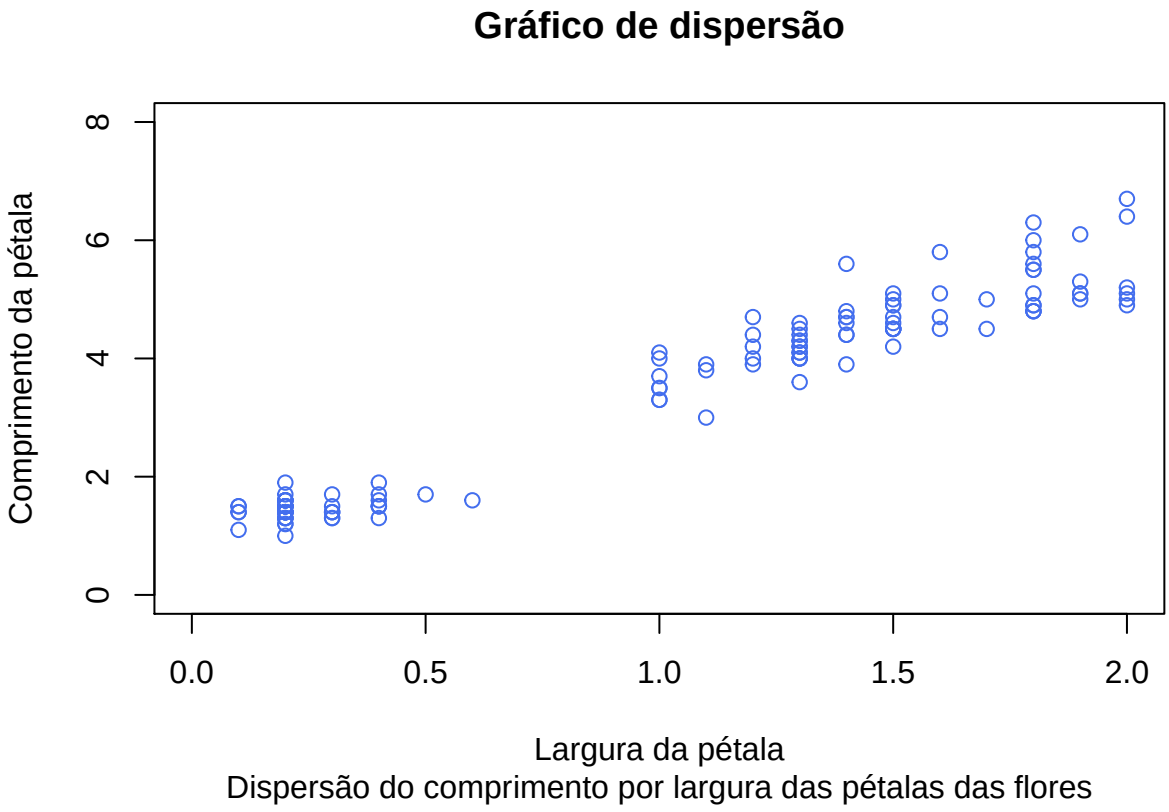
3.2.1 Principais Ajustes

As funções gráficas do R base, por padrão, tem opções em comum. Estas São:

- `main` Define o título do gráfico
- `sub` Define um subtítulo do gráfico
- `xlab` e `ylab` Trocam as legendas dos eixos
- `xlim` e `ylim` Trocam os limites dos eixos
- `col` Define a cor do gráfico

Exemplo:

```r
plot(iris$Petal.Width, iris$Petal.Length,
     main = "Gráfico de dispersão",
     sub = "Dispersão do comprimento por largura das pétalas das flores",
     xlab = "Largura da pétala",
     ylab = "Comprimento da pétala",
     xlim = c(0, 2), ylim = c(0, 8),
     col = "royalblue2")
```



3.2.2 Tipos de gráficos

A função `plot` pode gerar diversos tipos diferentes de gráficos, estes tipos podem ser definidos com o argumento `type`.

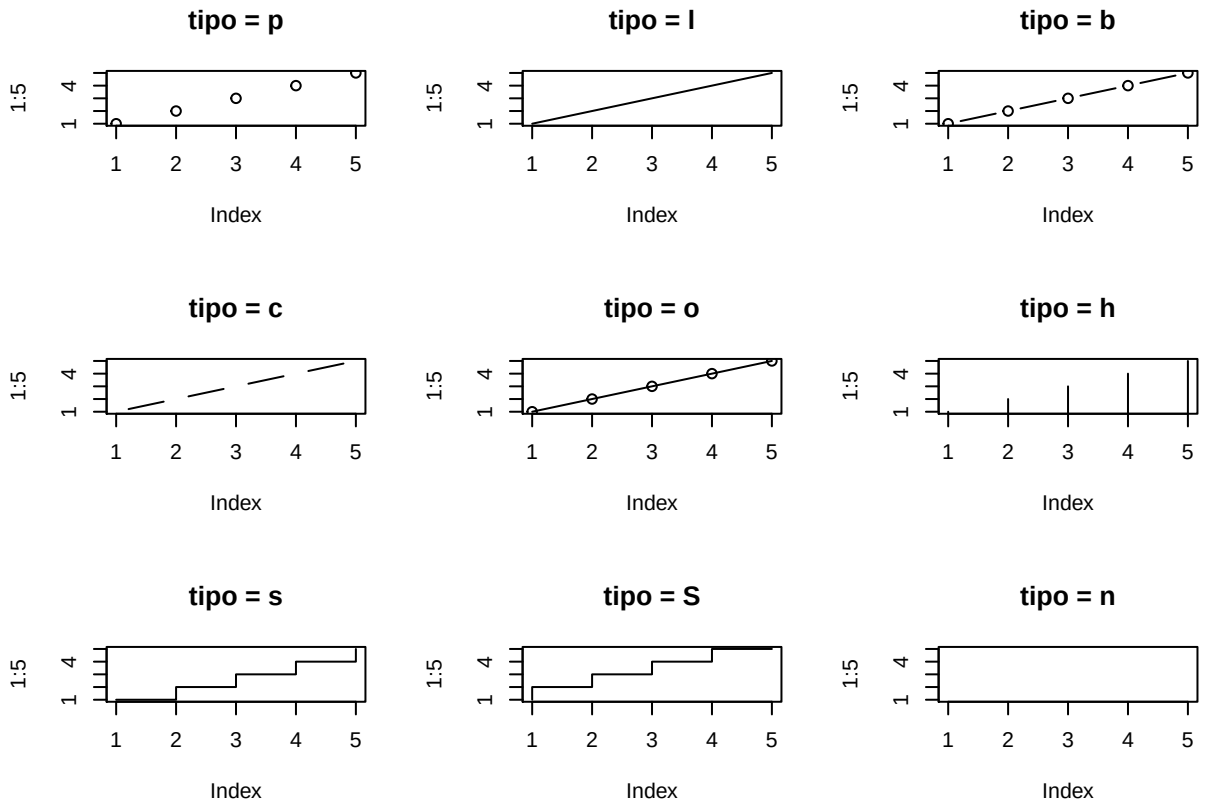
Como por exemplo na seguinte sintaxe `plot(x, y, type = "p")`, que gera um gráfico de pontos.

Abaixo estão listadas todos os tipos de gráficos q a função `plot` pode gerar:

- `p` **P**ontos,
- `l` **L**inhas,
- `b` Pontos e Linhas (**B**oth),
- `c` Somente as linhas da opção `b`,
- `o` Pontos e Linhas com sobreposição (**o**verplotted),
- `h` Linhas verticais como **h**istogramas,
- `s` Gráfico tipo e**s**cada (movendo primeiro horizontalmente),
- `S` Gráfico tipo e**S**cada (movendo primeiro verticalmente),
- `n` **N**enhum gráfico.

No exemplo abaixo temos todos os tipos de gráficos representados.

```
tipos <- c("p", "l", "b", "c", "o", "h", "s", "S", "n")
par(mfrow = c(3, 3))
for(tipo in tipos)
{
  plot(1:5, type = tipo, main = paste("tipo =", tipo))
}
```



A função `par` utilizada no exemplo acima será explicada na seção 3.3.6.

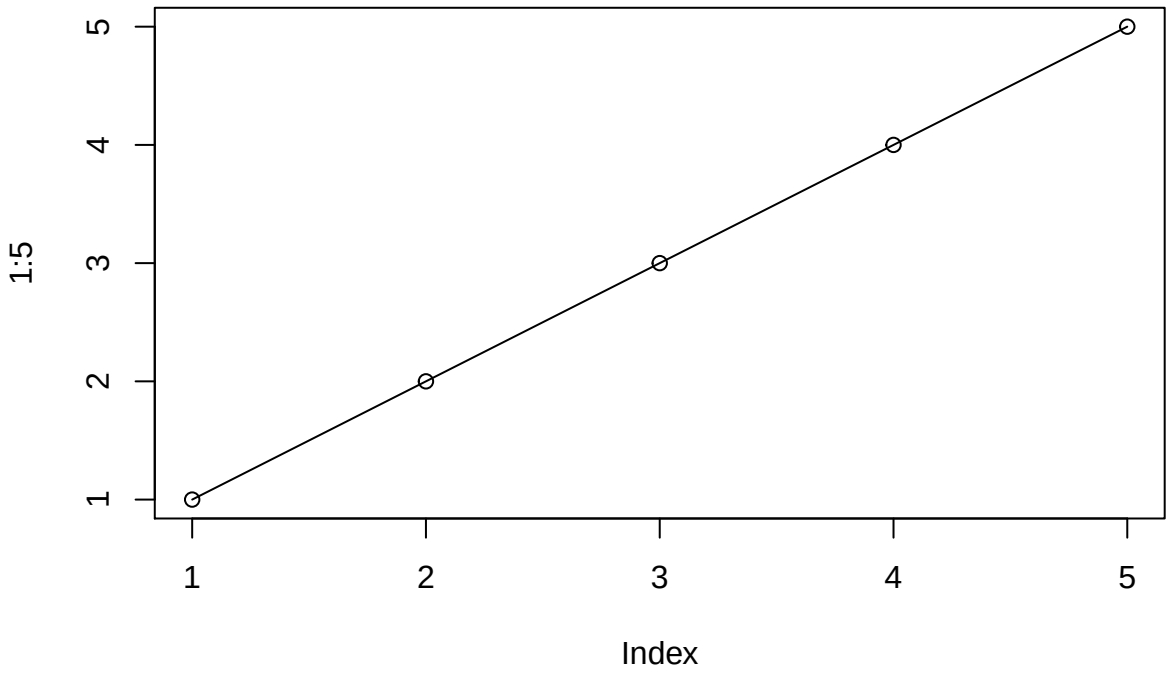
3.2.3 Sobreposição de Gráficos

No `R` existem funções que permitem sobrepor gráficos já feitos. Algumas delas são:

- `lines()`: Cria uma linha contínua sobrepondo o gráfico
- `points()`: Cria pontos sobrepondo o gráfico
- `polygon()`: Cria polígonos sobrepondo o gráfico

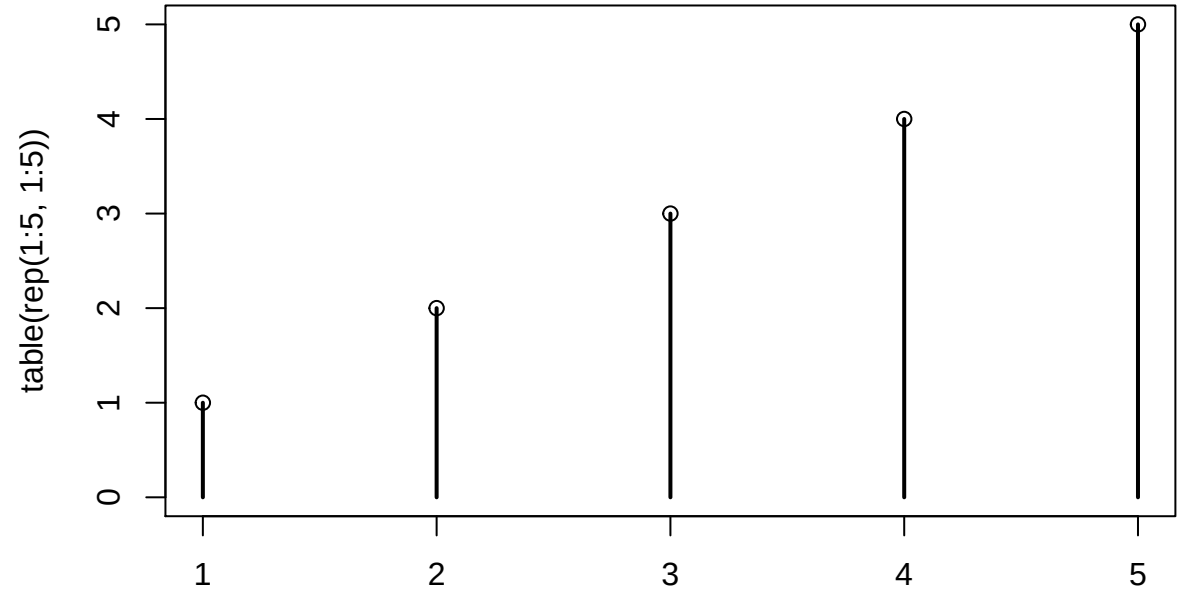
Exemplo utilizando `lines`

```
plot(1:5)
lines(1:5)
```



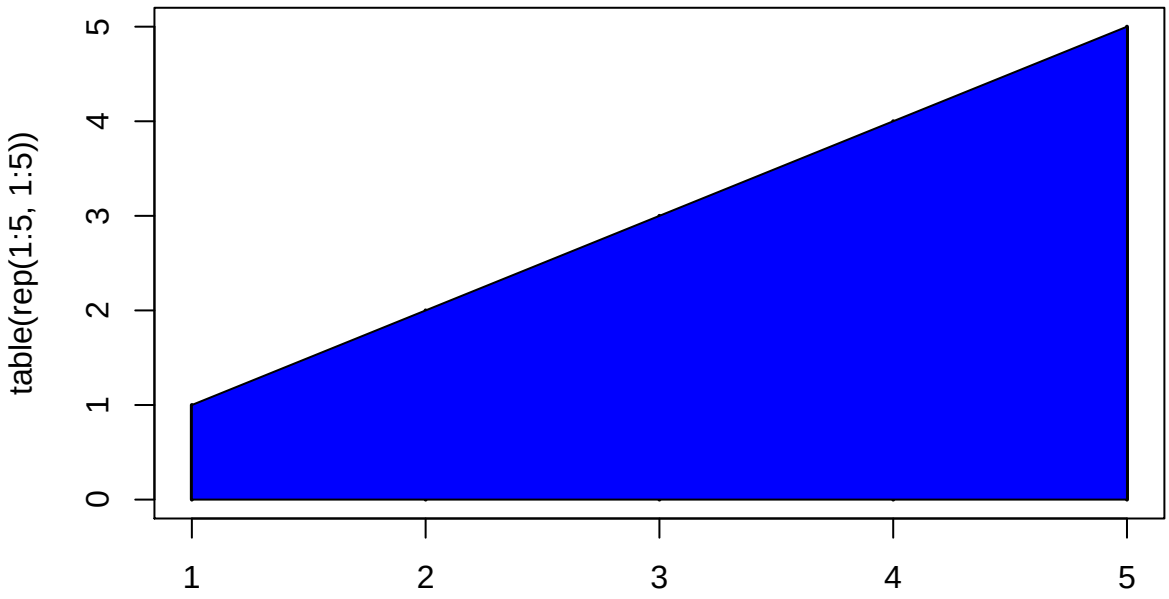
Exemplo utilizando `points`

```
plot(table(rep(1:5, 1:5)))
points(1:5, 1:5)
```



Exemplo utilizando `points`

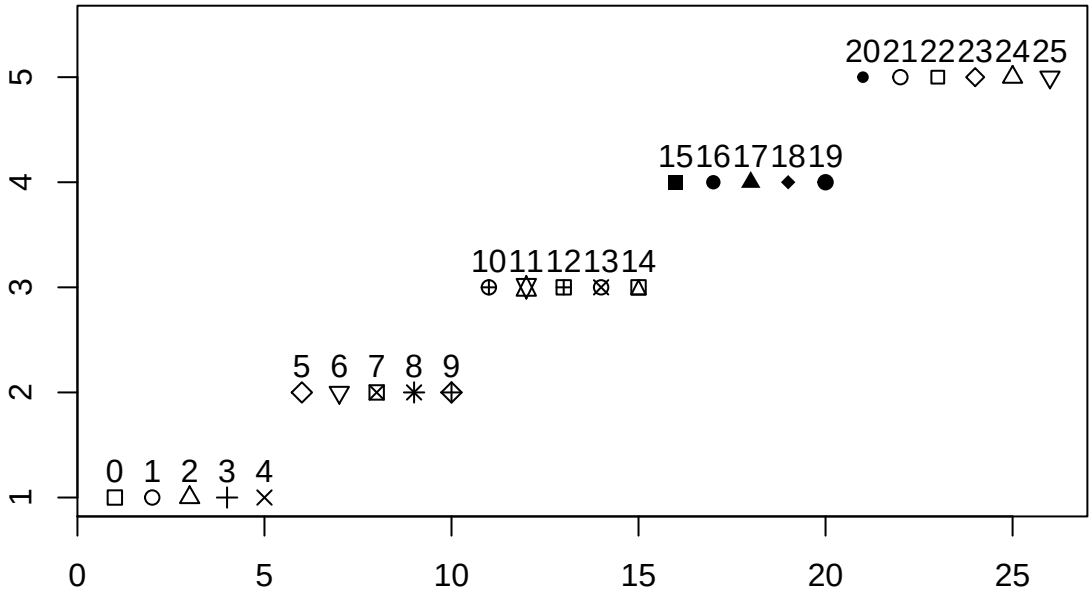
```
plot(table(rep(1:5,1:5)))
polygon(c(1,5,5,1), c(0,0,5,1), col = "blue")
```



3.2.4 Símbolos e Tipos de Linhas

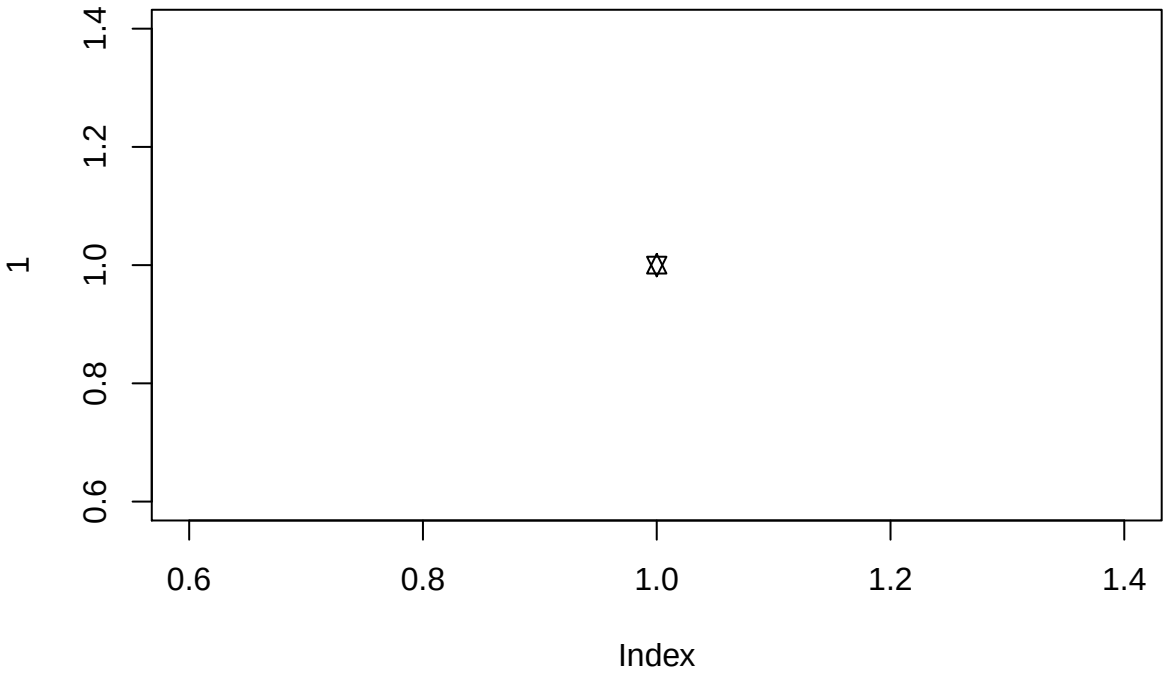
Quando são gerados gráficos de pontos dentro do `R`, é possível modificar os simbolos usados como pontos utilizando o argumento `pch` dentro de funções que gerem gráficos com pontos.

Quando o parâmetro `pch` é definido, ele deve ser um número dentro da tabela de símbolos descrita abaixo (números de 0 a 25), ou uma string com um único símbolo.



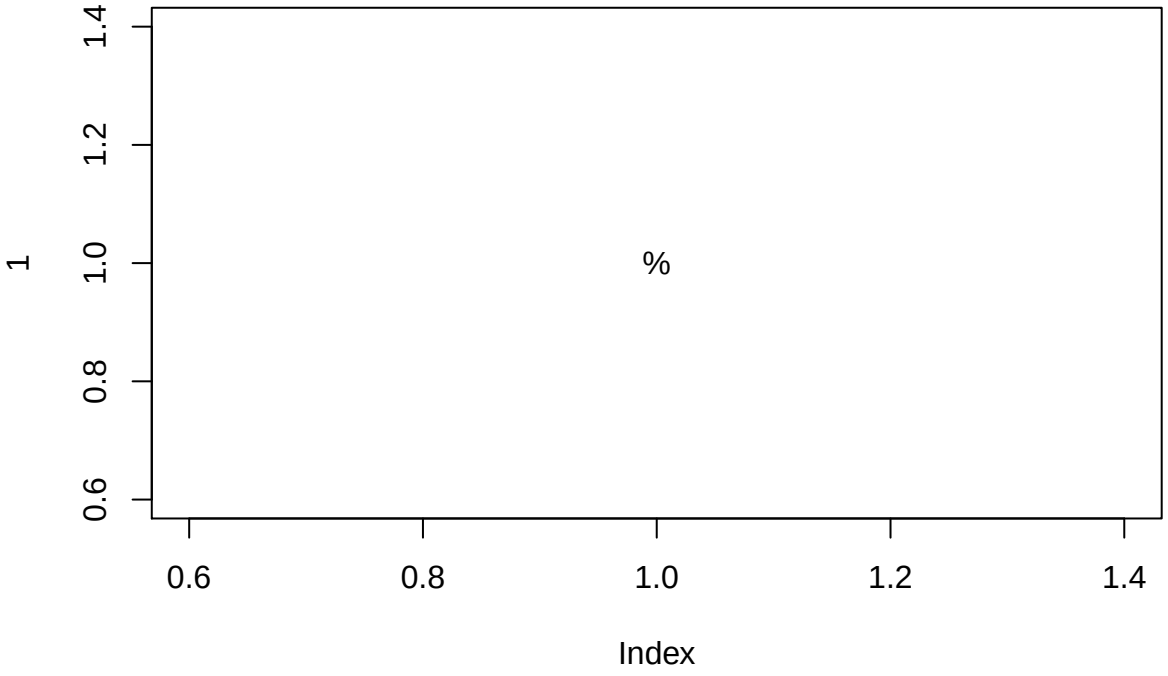
Exemplo de pontos usando números de `pch`

```
plot(1, pch = 11)
```



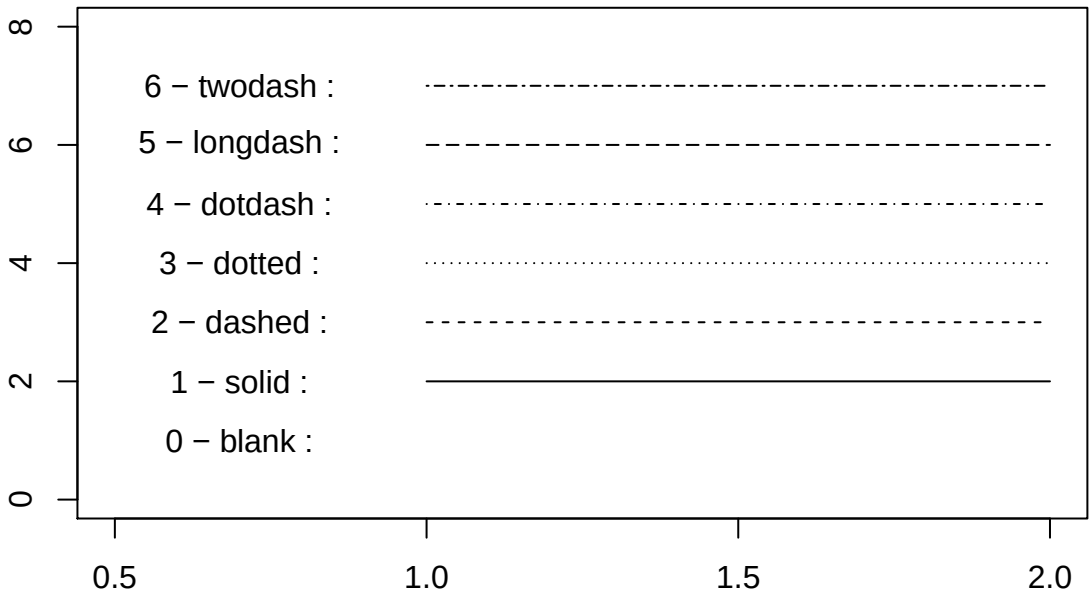
Exemplo de pontos utilizando strings de `pch`

```
plot(1, pch = "%")
```



Da mesma forma que podemos definir símbolos para gráficos de pontos, podemos definir tipos de linhas para gráficos de linhas, para isso, é utilizado o argumento `lty` dentro de funções que gerem gráficos de linhas.

Quando o parâmetro `lty` é definido, ele deve ser um número dentro da tabela de símbolos descrita abaixo (números de 0 a 5), ou strings de seus respectivos nomes.



3.2.5 Texto E Legendas

3.2.6 Função par

A função `par` define os parâmetros dos gráficos a serem gerados, abaixo mostraremos todas as opções dentro da função:

```
xlog, ylog, adj, ann, ask, bg, bty, cex, cex.axis, cex.lab, cex.main, cex.sub, cin, col, col.axis, col.
```

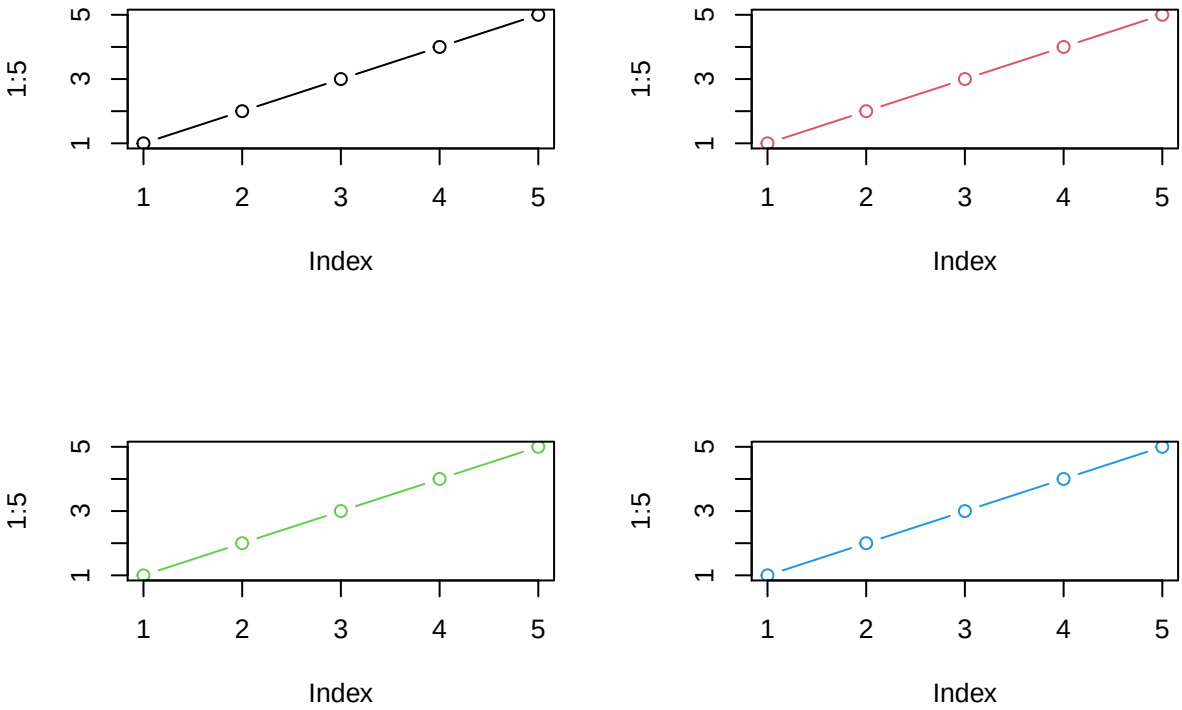
Como mostrado acima, existem diversos parâmetros gráficos a serem definidos, porém, neste curso serão apresentadas aplicações de todas as opções.

3.2.6.1 Concatenando gráficos Utilizando as opções `mfrow` ou `mfcol`, os dispositivos gráficos são divididos em uma matriz, onde cada elemento será preenchido por um novo gráfico gerado.

Quando chamamos uma dessas duas funções, definimos o tamanho da matriz a ser gerada, e está será preenchida por linha quando se utiliza a função `mfrow`, e por coluna quando se utiliza `mfcol`.

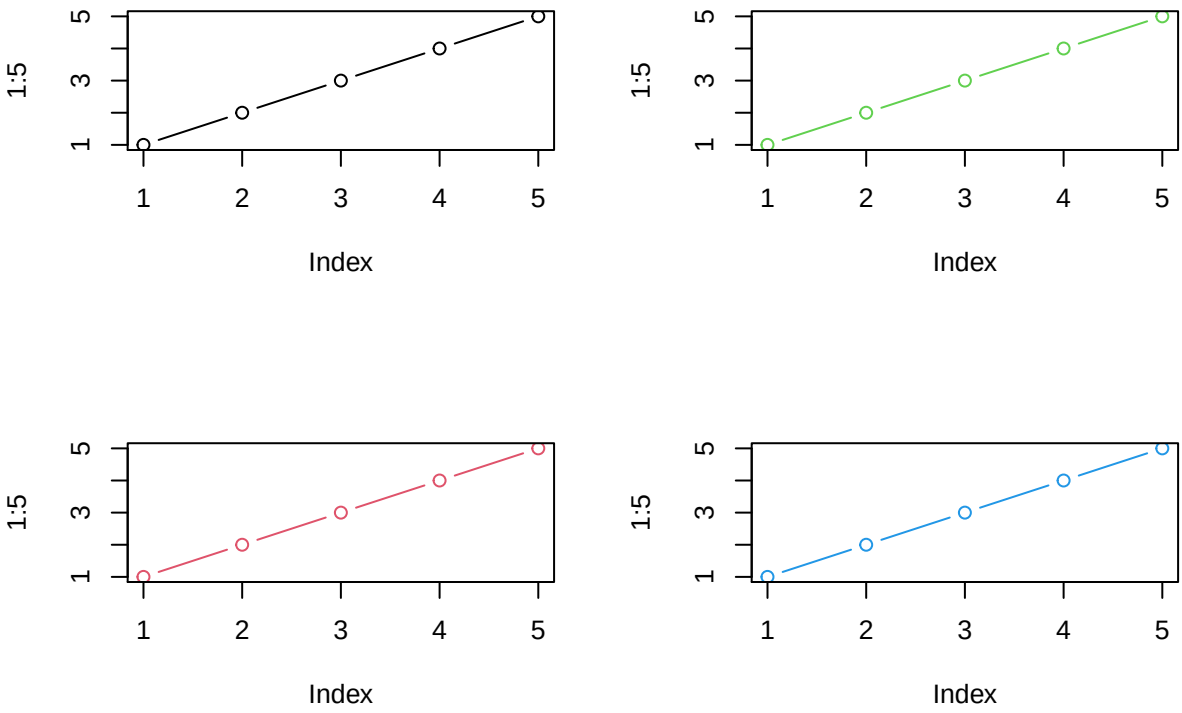
Exemplo utilizando `mfrow`

```
par(mfrow = c(2, 2))
for(i in 1:4) plot(1:5, type = "b", col = i)
```



Exemplo utilizando `mfcol`

```
par(mfcol = c(2, 2))
for(i in 1:4) plot(1:5, type = "b", col = i)
```

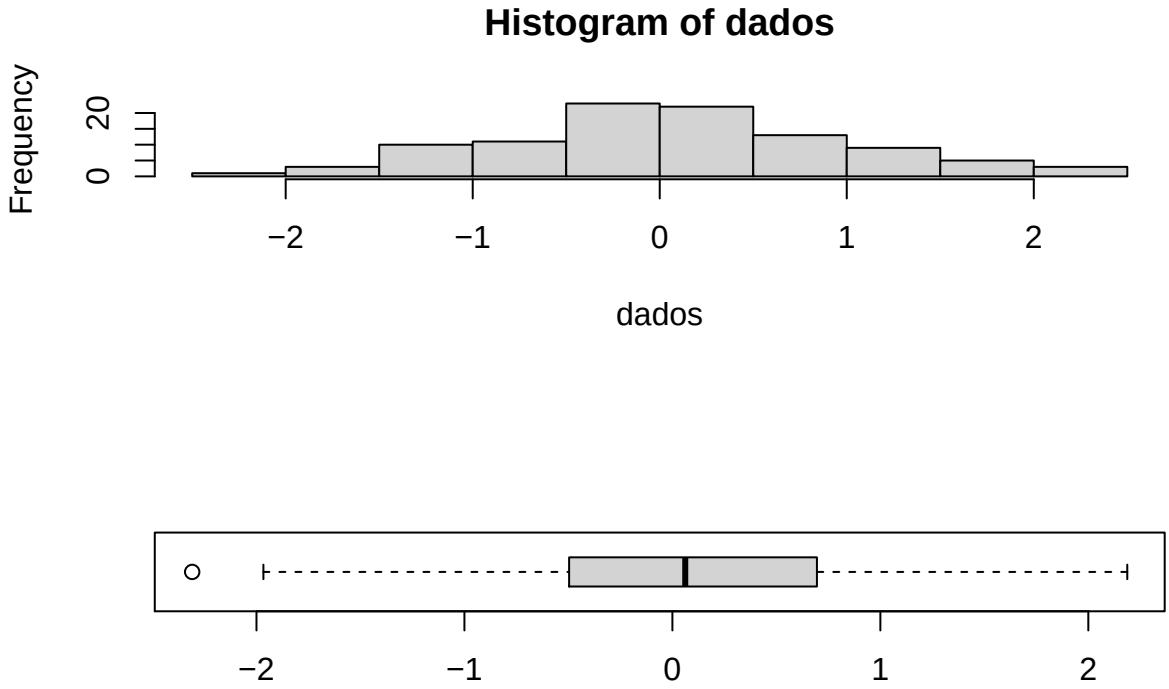


Note que a única diferença na saída das duas funções é a ordem em que os gráficos são posicionados.

Exemplo:

```
set.seed(123) # Semente para reprodutibilidade do código
par(mfcol = c(2, 1))
```

```r
dados <- rnorm(100)
hist(dados)
boxplot(dados, horizontal = TRUE)
```

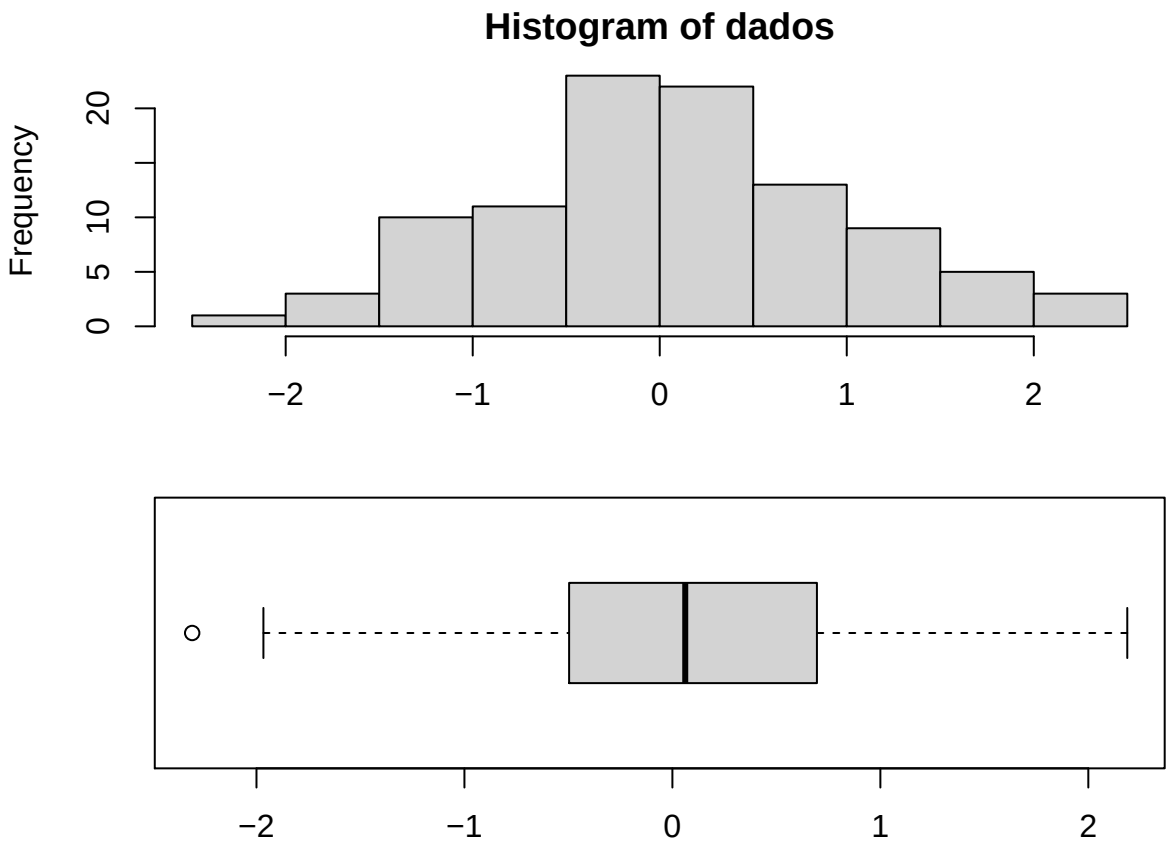


3.2.6.2 Definindo margens Utilizando a opção `mar`, podemos definir o tamanho das margens de cada gráfico a ser feito.

O parâmetro `mar` recebe um vetor númerico com 4 elementos, onde o primeiroa representa a margem inferior, o segundo a margem esquerda, o terceiro e margem de cima e por último a margem da direita, como mostrado a seguir: `par(mar = c(baixo, esquerda, cima, direita))`

Exemplo

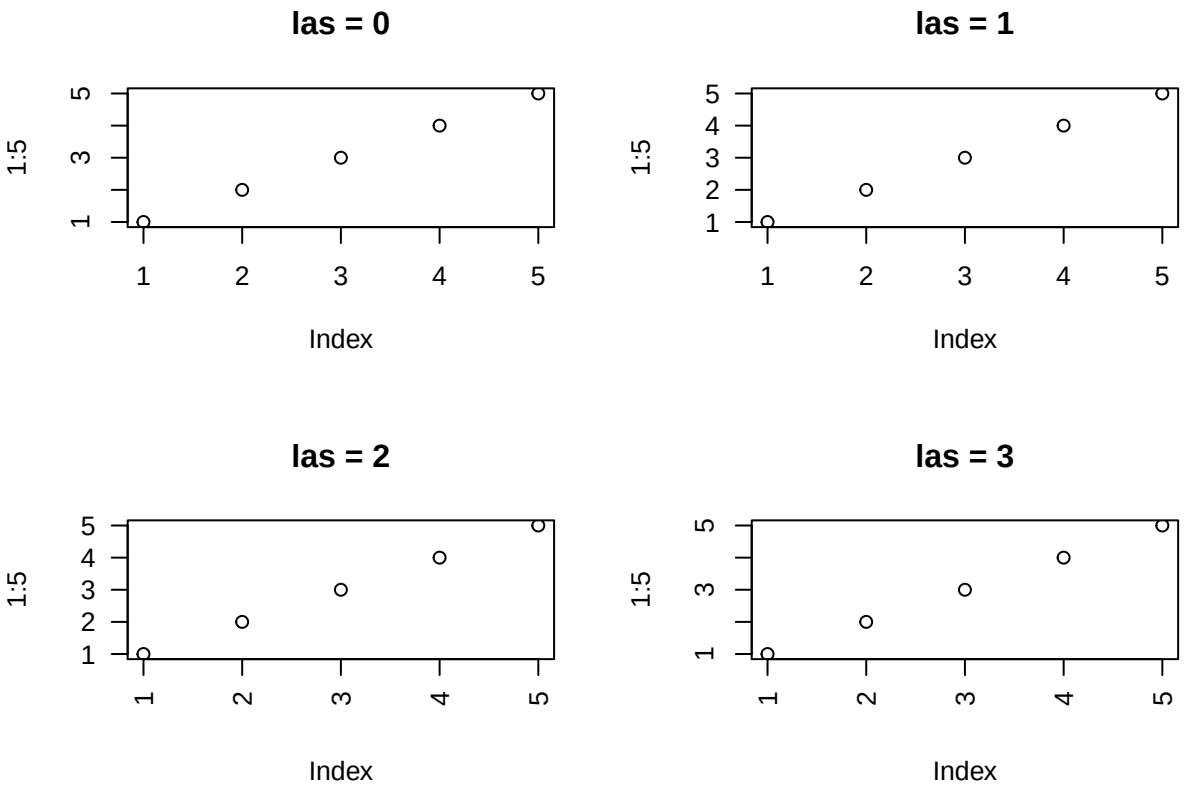
```r
set.seed(123)
dados <- rnorm(100)
par(mfrow = c(2, 1), mar = c(2.1, 4.1, 2.1, 2.1))
hist(dados)
boxplot(dados, horizontal = TRUE)
```



3.2.6.3 Orientação dos eixos Utilizando o parâmetro `las`, é possível definir a orientação do texto nos eixos x e y. Este recebe um número de 0 a 3 para definir tal orientação.

Exemplo descrevendo todas as opções

```
par(mfrow = c(2, 2))
for(i in 0:3)
{
  par(las = i)
  plot(1:5, main = paste("las =", i))
}
```

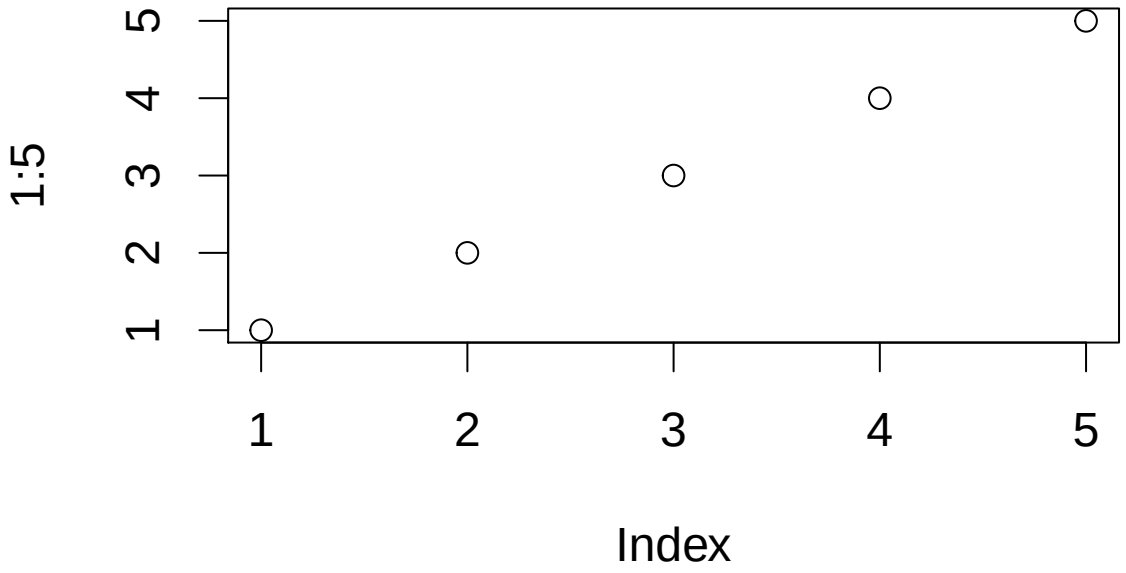


3.2.6.4 Escala de texto Utilizando o parâmetro `cex`, é possível mudar o tamanho de todo o texto presente no gráfico. Além da opção `cex`, existem quatro outra opções para mudar a escala do texto de partes específicas do gráfico. Essas são:

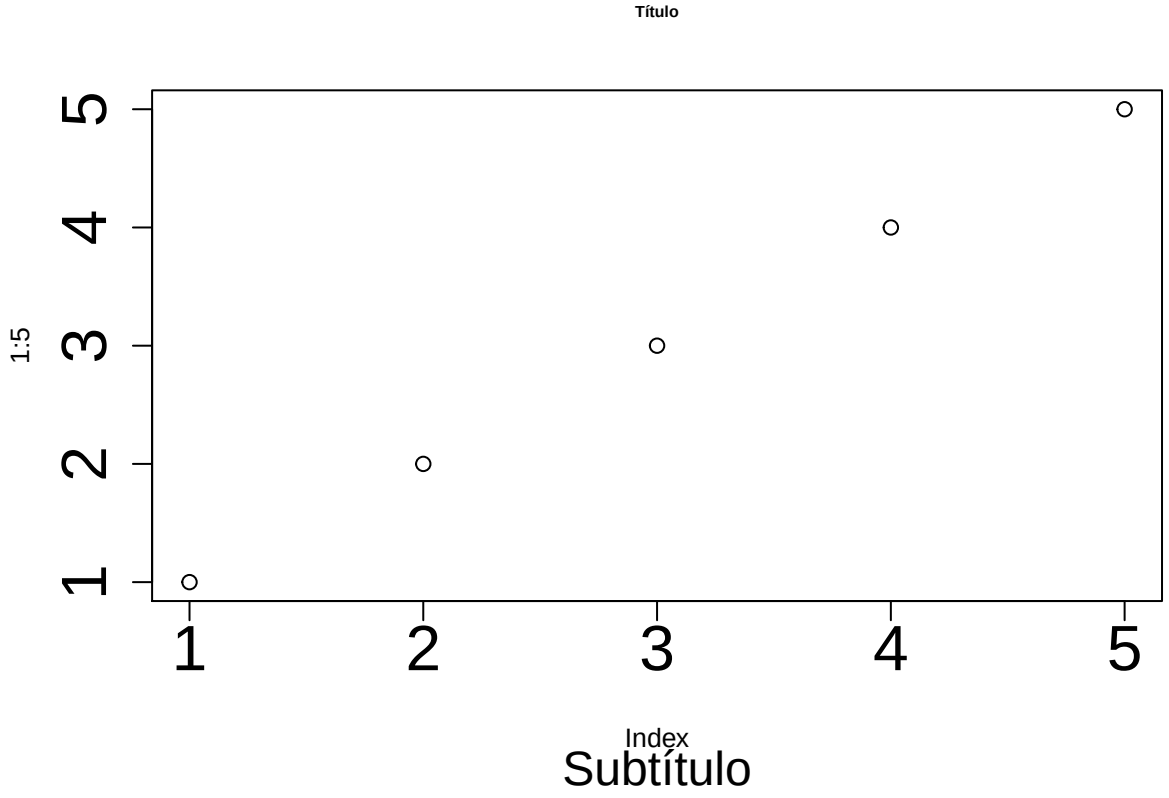
- `cex.axis`: Escala do texto dos eixos
- `cex.lab`: Escala do texto dos nomes dos eixos
- `cex.main`: Escala do texto do título
- `cex.sub`: Escala do texto do subtítulo

Exemplos

```
par(cex = 1.5)
plot(1:5)
```




```r
par(cex = 1, cex.axis = 2, cex.lab = 0.8, cex.main = 0.5, cex.sub = 1.5)
plot(1:5, main = "Título", sub = "Subtítulo")
```



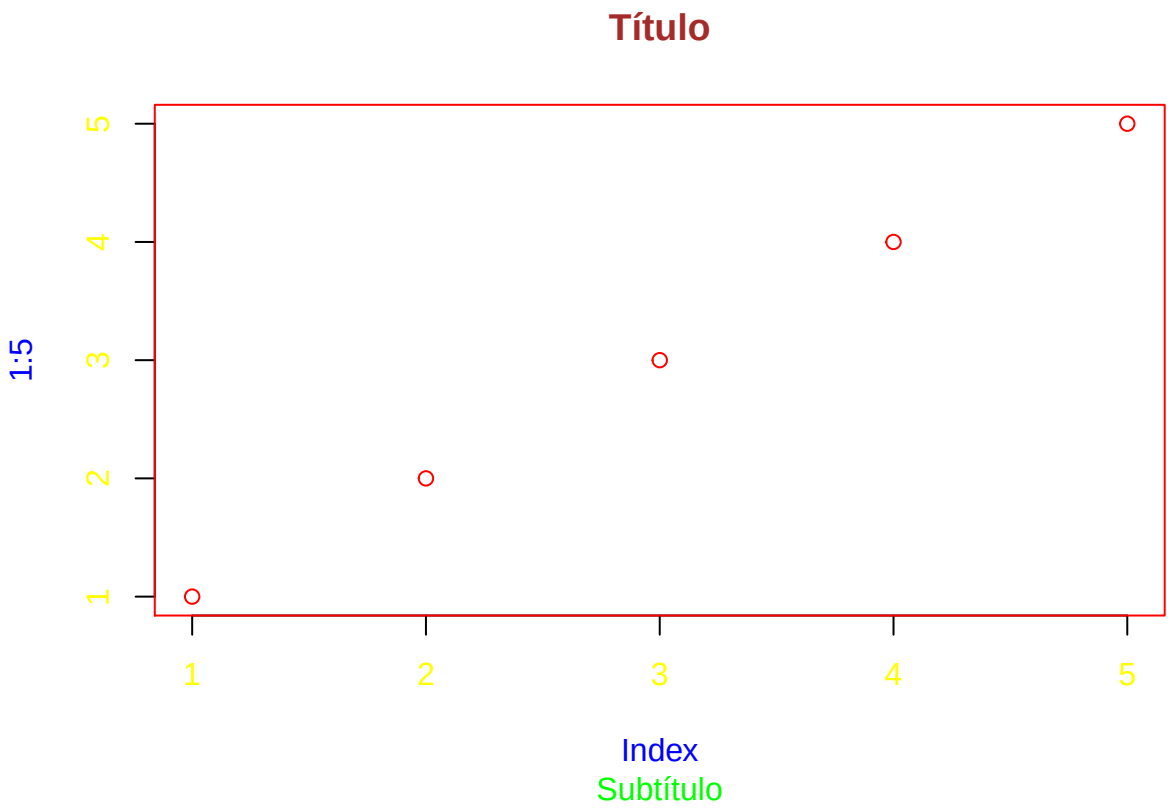
3.2.6.5 Definindo cores Utilizando o parâmetro `col` é possível definir as cores dos próximos gráficos a serem feitos. Como no tópico anterior, temos opções separadas para definir a cor de diferentes partes do gráfico, porém, as opção `col` não define cores para os textos do gráfico, define somente para o gráfico em si (bordas, pontos, linhas, etc.). Portanto todas as opções para definição de cores são:

- `col`: Define a cor do gráfico
- `col.axis`: Define a cor do texto dos eixos
- `col.lab`: Define a cor do texto dos nomes dos eixos

- `col.main`: Define a cor do título
- `col.sub`: Define a cor do subtítulo

Exemplo

```
par(col = "red", col.axis = "yellow", col.lab = "blue",
    col.main = "brown", col.sub = "green", col)
plot(1:5, main = "Título", sub = "Subtítulo")
```



3.2.6.6 Tipos de fonte Utilizando o parâmetro `font`, é possível definir o tipo de texto a ser usado no gráfico. Todos os argumentos `font.*` recebem um número de 1 a 5, onde esses números representam:

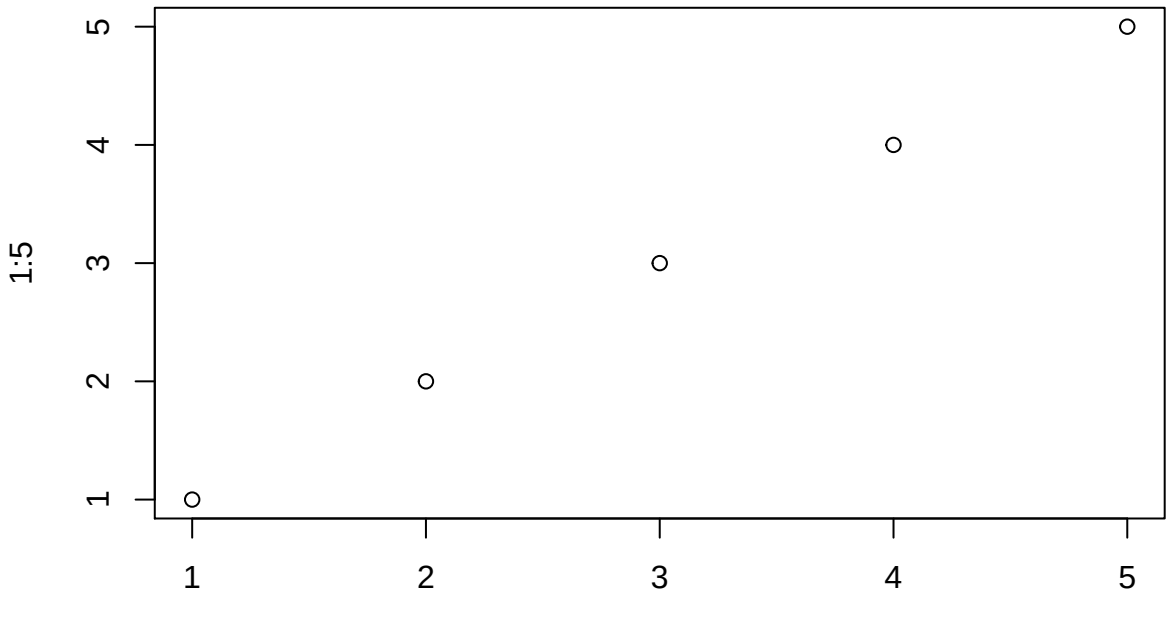
1. Texto padrão
2. Texto Negrito
3. Texto Italico
4. Texto Negrito Italico
5. Codificação de Símbolos Adobe

Como nos tópicos anteriores, temos opções específicas para diferentes partes do gráfico, estas são:

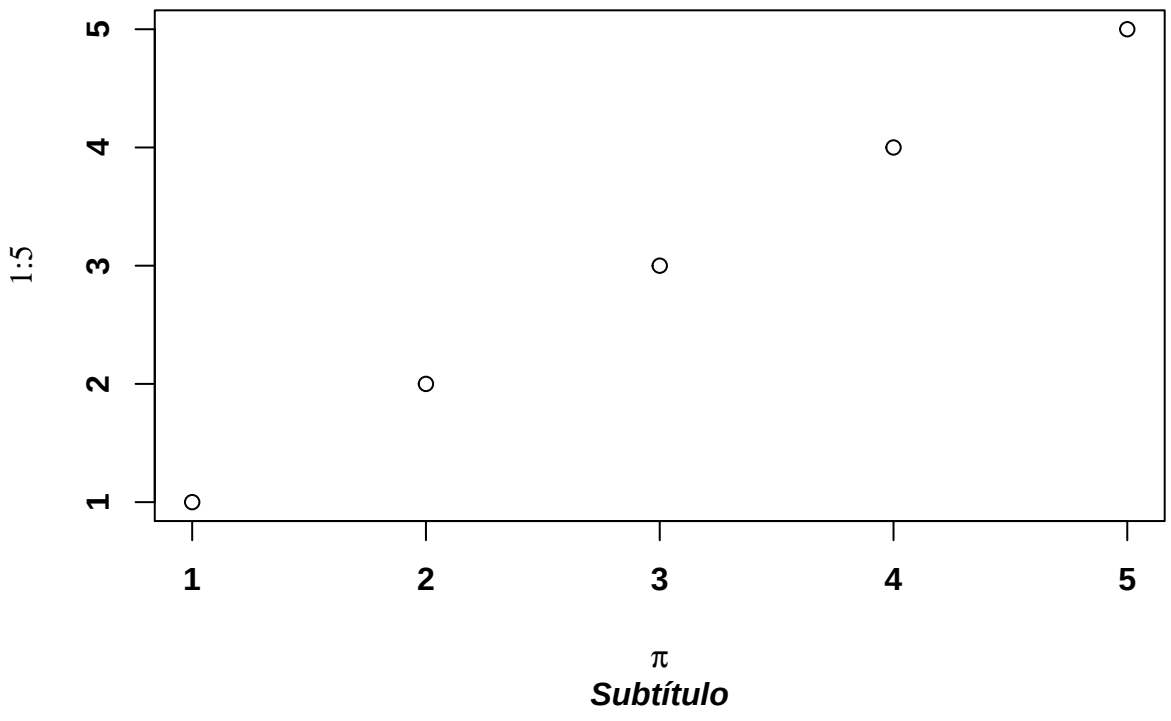
- `font`: Define a fonte do gráfico
- `font.axis`: Define a fonte do texto dos eixos
- `font.lab`: Define a fonte do texto dos nomes dos eixos
- `font.main`: Define a fonte do título
- `font.sub`: Define a fonte do subtítulo

Exemplo

```
par(font = 4)
plot(1:5)
```



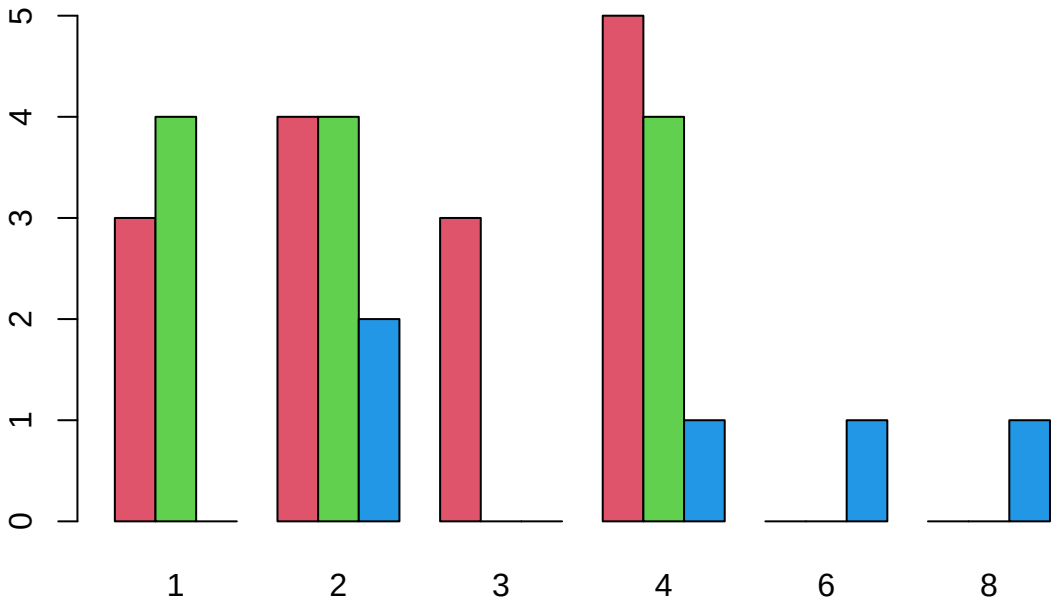

```
par(font = 1, font.axis = 2, font.main = 3, font.sub = 4, font.lab = 5)
plot(1:5, xlab = "\160", main = "Título", sub = "Subtítulo")
```



3.2.7 Cores

3.2.8 Representando tabelas de contingencia

```
barplot(table(mtcars$gear, mtcars$carb), beside = TRUE, col = 2:4)
```

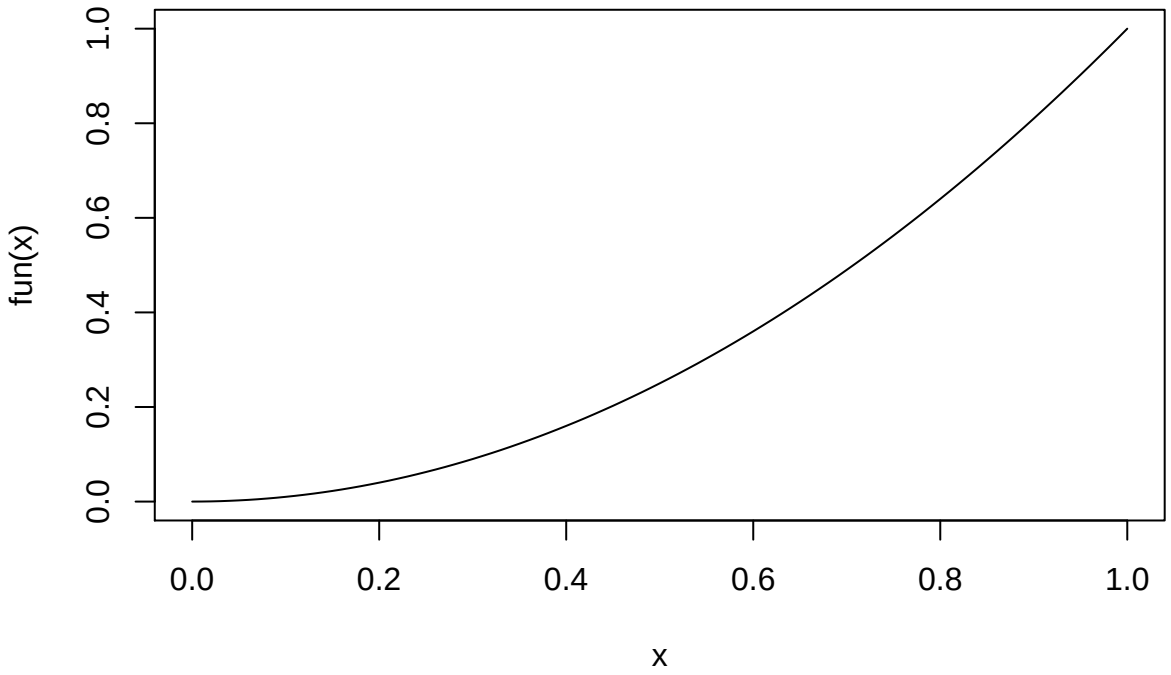


3.2.9 Representando funções matemáticas

É possível representar funções matemáticas no `R` utilizando a função `curve`. Como foi visto na seção 1.5, podemos criar funções dentro do `R`, e essas funções serão utilizadas para que possamos representar funções matemáticas. É importante frisar que a função que será inserida no comando `curve` deve estar em função de uma variável `x`.

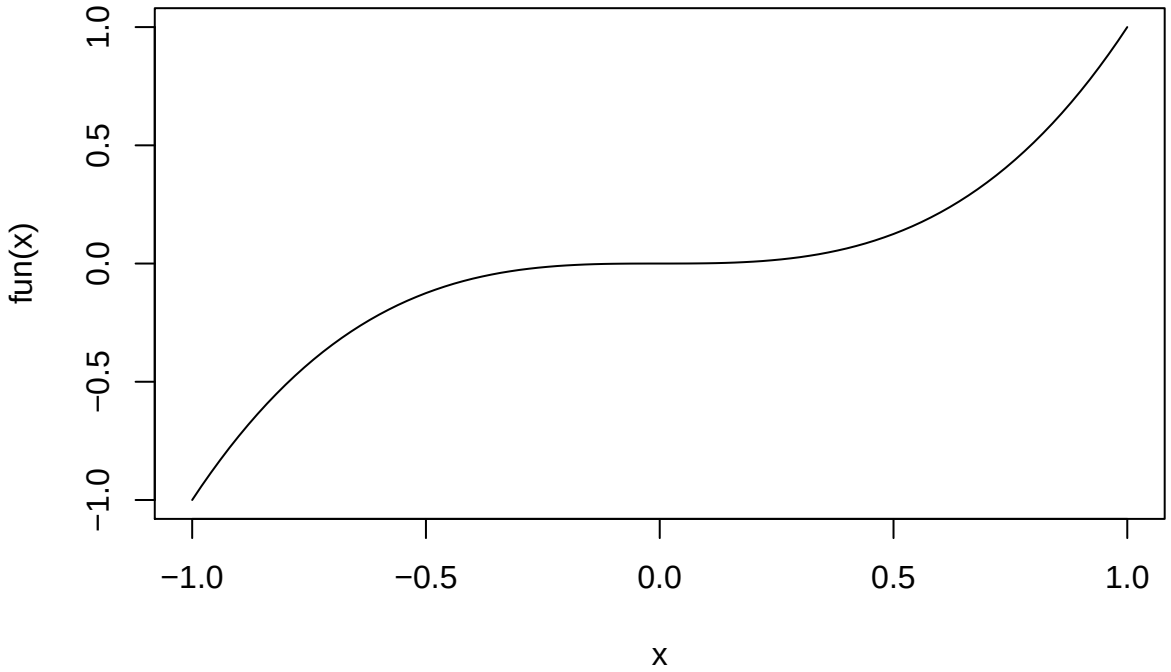
Por exemplo:

```
fun <- function(x) x^2
curve(fun)
```



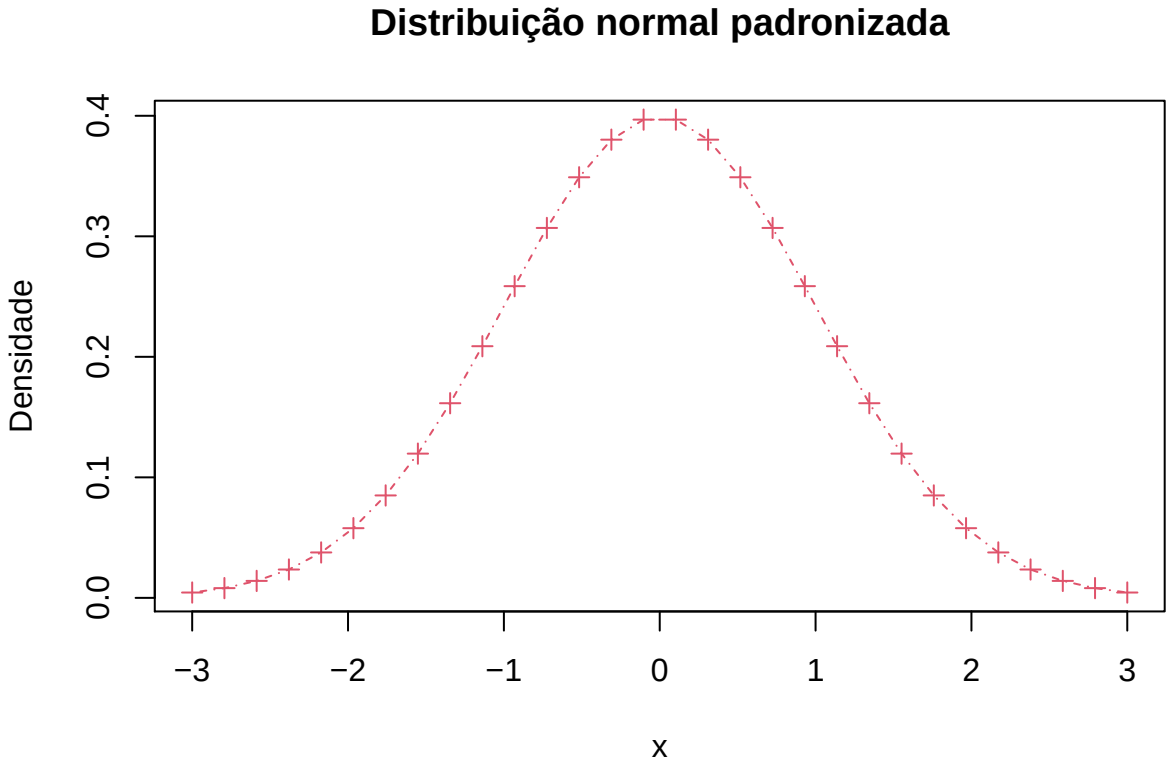
Vemos no gráfico acima que o gráfico da função foi feito utilizando o valor `x` entre 0 e 1. Na função `curve`, podemos definir os argumentos `from` e `to`, que vão definir os valores iniciais e finais de `x`a serem representados no gráfico.

```
fun <- function(x) x^3
curve(fun, from = -1, to = 1)
```



Além dos arumentos `from` e `to`, a função `curve` pode receber o argumento `n` que representa o número de pontos a serem representados no gráfico (por padrão `n = 101`), além de receber também as opções descritas nas seções anteriores.

```r
fun <- function(x) (1/sqrt(2 * pi)) * exp(-0.5*(x^2))
curve(fun, from = -3, to = 3, n = 30, type = "o", pch = 3, lty = 4, col = 2,
      main = "Distribuição normal padronizada", xlab = "x", ylab = "Densidade")
```



3.2.10 Dispositivos Gráficos

Dispositivos gráficos, são interfaces onde os gráficos gerados são exibidos, seja esse um dispositivo gráfico integrado na sua IDE, como por exemplo Rstudio, ou uma janela gráfica aberta no seu sistema operacional, ou até mesmo um arquivo no seu computador. No `R`, quando chamamos uma função que gera um gráfico, automaticamente é criado um dispositivo gráfico padrão, que varia de acordo com a plataforma que você está utilizando. Em sistemas operacionais baseados em Unix o dispositivo gráfico padrão é `X11()`, no sistema operacional Microsoft Windows o dispositivo gráfico padrão é `windows()`, e independente do sistema operacional quando utilizamos Rstudio o dispositivo gráfico padrão é o `RStudioGD()`, que é a janela integrada para gráficos. Ambos `X11()` e `windows()` podem ser chamados a qualquer momento dentro de uma sessão do `R` para criar uma janela externa onde serão exibidos os gráficos. Outros dispositivos gráficos além das janelas externas mencionadas são os dispositivos gráficos de arquivos que serão explicados na próxima seção.

3.2.11 Exportando gráficos

4 Noções de Probabilidade

4.1 Amostragem

O comando `sample()` tem a função de obter amostras de alguma estrutura de dados que for de interesse. Para isso, é necessário informar:

- Os dados
- Tamanho da Amostra
- Se será uma amostra com ou sem reposição
- As probabilidades de cada elemento (Padrão = 1/2)

Exemplos:

```
valores <- 1:10 #dados
letras <- letters

# amostra de 5 elementos com reposição
sample(x = valores, size = 5, replace = T)
```

```
## [1] 4 9 8 6 4
```

```
# amostra de 7 elementos sem reposição
sample(x = letras, size = 7, replace = F)
```

```
## [1] "h" "c" "d" "t" "l" "q" "j"
```

4.2 Análise Combinatória

4.2.1 Permutação

A permutação de n elementos distinos pode ser calculada no R através da função `factorial()`.

Exemplo:

```
factorial(10)
```

```
## [1] 3628800
```

4.2.2 Arranjo

Um Arranjo considera a posição dos elementos além de suas combinações, diferente da combinação, que considera somente os elementos, e não a ordem deles.

Não há uma função pronta no R base para isso, mas é possível implementá-la através da seguinte função:

```
arranjo <- function(n, x){
  return(factorial(n) / factorial(n-x))
}
```

Exemplo:

```
# calculando Arr(5,3)
arranjo(5,3)
```

```
## [1] 60
```

4.2.3 Combinação

Para fazer a combinação, utiliza-se a função `choose()`.

Exemplo:

```
# calculando comb(5,3)
choose(5,3)
```

```
## [1] 10
```

4.3 Distribuições de Probabilidade

Ao trabalhar com variáveis aleatórias, geralmente são associadas distribuições de probabilidade a essas variáveis. E, tendo conhecimento dessas distribuições, é possível obter informações dos dados, a partir da estatística.

No R básico estão implementadas as seguintes distribuições:

- Beta
- Binomial
- Cauchy
- Chi-quadrado
- Exponencial
- F
- Gamma
- Geométrica
- Hipergeométrica
- Log-Normal
- Multinomial
- Binomial Negativa
- Normal
- Poisson
- T de Student
- Uniforme
- Weibull

4.3.1 Geração de valores aleatórios

Para se gerar valores aleatórios que venham de alguma distribuição de probabilidade utiliza-se o prefixo "r" juntamente com a abreviação da distribuição, da seguinte forma:

`r"disribuição"(n,...)`

Para essa função, é necessário informar:

- n: quantidade de valores aleatórios gerados
- $...$: os parâmetros da distribuição

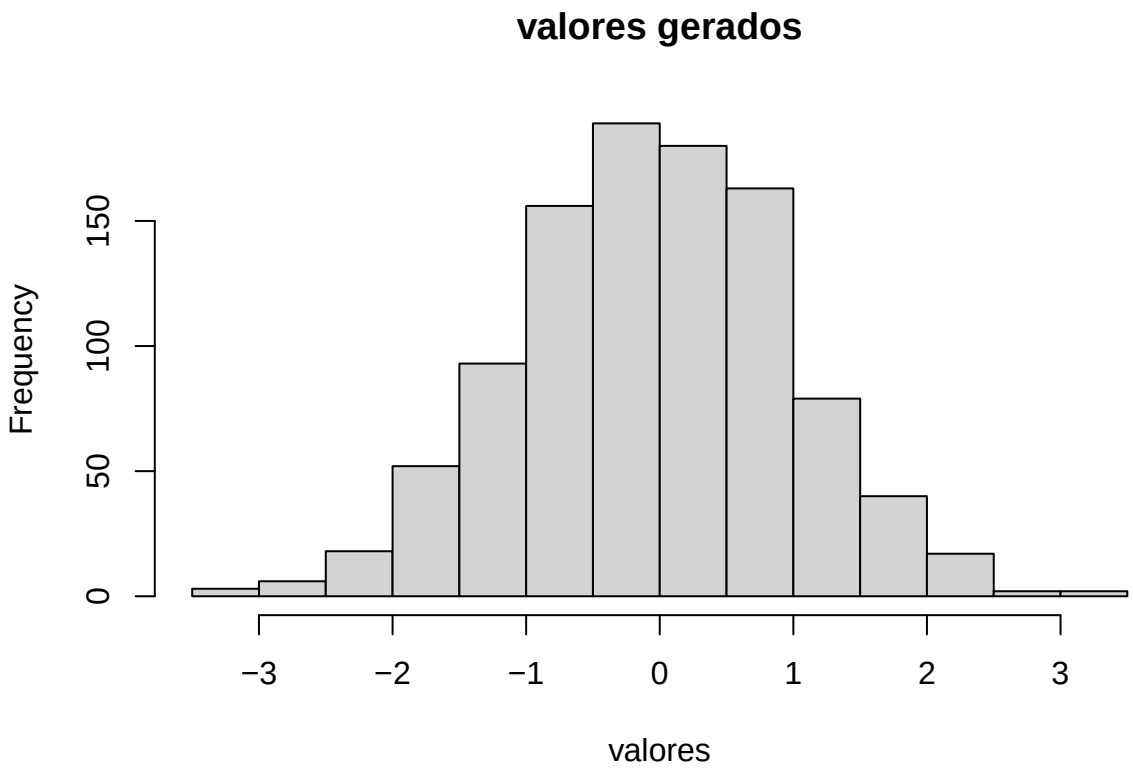
Exemplos:

```
# gerando 10 valores aleatórios baseados em uma normal padrão
rnorm(n = 10, mean = 0, sd = 1)
```

```
##  [1]  1.3974267  1.7636530  0.4856014 -0.2657389  0.1516114  1.3766098
##  [7] -0.1803943 -1.5676751 -0.2607259  0.9618104
```

```
# gerando 1000 valores aleaórios para uma normal com media = 100 e desvio padrão = 5
valores <- rnorm(n = 1000, mean = 0, sd = 1)

# criando um histograma para os dados acima
hist(valores, main = "valores gerados")
```



4.3.2 Função de Densidade / Probabilidade

A função para gerar as probabilidades da função densidade de uma distribuição é semelhante a função anterior, porém, utiliza-se o prefixo "d":

```
d"disribuição"(x,...)
```

Para essa função, é necessário informar:

- x: o quantil associado a probabilidade de interesse
- $\ldots$: os parâmetros da distribuição

Exemplo:

```
# achando P(X = 2) para X ~ Bin(n = 4, p = 0.7)
dbinom(x = 2, size = 4, prob = 0.7)
```

```
## [1] 0.2646
```

4.3.3 Função Acumulada

De forma análoga, a probabilidade da função acumulada de determinada distribuição é calculada ao utilizar o prefixo "p":

```
p"disribuição"(q,...)
```

Para essa função, é necessário informar:

- q: o quantil associado a probabilidade de interesse
- $\ldots$: os parâmetros da distribuição

Exemplo:

```
# achando a probabilidade acumulada para o exemplo anterior
pbinom(q = 2, size = 4, prob = 0.7)
```

```
## [1] 0.3483
```

4.4 Quantis

Para o cálculo dos quartis de determinada distribuição, usa-se o prefixo "q":

```
q"disribuição"(p,...)
```

Para essa função, é necessário informar:

- p: a probabilidade associada ao quantil de interesse
- $\ldots$: os parâmetros da distribuição

Exemplo:

```
# Calculando y tal que P(X<y) = 0.5 para a distribuição usada anteriormente
qbinom(p = 0.5, size = 4, prob = 0.7)
```

```
## [1] 3
```