

Revisão

O Material desta revisão foi extraído do Livro :

Introdução à mineração de dados: com aplicações em R / Leandro Augusto da Silva, Sarajane Marques Peres, Clodis Boscarioli. – 1. ed. – Rio de Janeiro: Elsevier, 2016.



Variáveis

Em *R*, uma variável é definida com o uso do operador `<-`, o qual significa “gets” e, em português, poderia ser traduzido como “atribuição”. Por exemplo, a atribuição do valor 1 à variável *x*, em R, é realizada da seguinte forma:

```
In [130]: x<-1
```

Para verificar o valor atribuído à variável, proceda da seguinte forma: > `x`.

E, após o acionamento da tecla ENTER, o valor da variável será mostrado na tela:

In [131]:

```
x
```

1

Essa notação para atribuição de variável é comumente usada em pseudocódigos e é bastante comum em livros que apresentam discussões sobre algoritmos.

Perceba também que não é necessário definir o tipo da variável, a própria linguagem fará essa definição no momento de execução da atribuição.

Veja outros exemplos de atribuições, considerando diferentes tipos de dados:

```
In [3]: # Atribuindo um conjunto de caracteres (string) à variável x
x <- "Olá mundo!"
x
```

'Olá mundo!'

Tipos de dado

Embora não seja necessário definir o tipo de variável antes de lhe atribuir um valor, existem situações em que se faz necessário saber o tipo de variável.

Em *R*, essa consulta pode ser feita com o uso da função `class()`.

Os tipos de dado retornados pela função `class` são numérico (`numeric`) e caractere (`character`).

Um tipo numérico pode ser tanto um valor real (`float` ou `double`) quanto um valor inteiro (`integer`), ou seja, não se faz a distinção entre esses dois tipos de valor.

A linguagem faz a conversão de valor real para valor inteiro de forma transparente ao usuário. Veja como conhecer o tipo de uma variável por meio das seguintes instruções:

```
In [132]: x <- 3.46  
       class(x)
```

'numeric'

```
In [133]: x <- 10  
class(x)
```

'numeric'

Em situações em que se faz necessário definir uma variável como um valor do tipo inteiro, é preciso realizar a atribuição do valor à variável acompanhado de *L*, ou seja:

```
In [134]: x <- 10L  
         class(x)
```

'integer'

Note que, independentemente da natureza numérica atribuída, a linguagem entende a variável como sendo do mesmo tipo, numeric. Entretanto, caso seja necessário verificar o tipo exato da variável, as funções `is.numeric()` ou `is.integer()` devem ser executadas. Essas funções retornam como resultado os valores lógicos `TRUE` ou `FALSE` para verdadeiro ou falso, respectivamente. Veja o seguinte exemplo:

```
In [135]: x <- 3.46  
is.numeric(x)
```

TRUE

```
In [136]: is.integer(x)
```

FALSE

Para variáveis às quais foi atribuído um conjunto de caracteres, a consulta sobre seu tipo terá como resposta o valor `char` como resposta o valor `char`. Para descobrir a quantidade de caracteres na variável, a função `nchar()` pode ser usada. Veja o exemplo a seguir:

```
In [137]: x <- "Olá mundo!"  
class(x)
```

'character'

```
In [10]: nchar(x)
```

10

Vetores

Vetores são variáveis com um ou mais valores do mesmo tipo: inteiro, real, lógico, caractere (ou string) etc. Em *R*, mesmo uma variável com um único valor (um escalar) é considerada um vetor, nesse caso, um vetor de comprimento 1.

A criação de um vetor em *R* pode ser feita de diferentes maneiras, dependendo do uso que se pretende para a variável.

Se o propósito de criação do vetor é a representação de uma sequência numérica, então é esperada a definição do valor inicial e do valor final do vetor, separados pelo caractere : (dois-pontos).

Por exemplo:

```
In [138]: x <- 1:10  
x
```

1 · 2 · 3 · 4 · 5 · 6 · 7 · 8 · 9 · 10

Também é possível criar uma sequência na qual o intervalo entre os valores não seja unitário. A função `seq()` deve ser usada para esse propósito.

Ela recebe três parâmetros: o *valor inicial da sequência*, o *valor final da sequência* e o *intervalo desejado* entre os valores dessa sequência.

Caso o terceiro parâmetro não seja definido, a função assume o intervalo unitário. O exemplo a seguir gera uma sequência de 1 até 10 com intervalo de 2.

```
In [139]: seq(1,10,2)
```

1 · 3 · 5 · 7 · 9

Caso seja necessário criar uma sequência com valores repetidos, a função `rep()` deve ser usada.

Ela permite repetir um valor ou uma sequência de valores. Veja exemplos da aplicação dessa função a seguir:

```
In [140]: rep(1, 5)
```

1 · 1 · 1 · 1 · 1

```
In [14]: rep(1:3, 2)
```

1 · 2 · 3 · 1 · 2 · 3

Se a necessidade for gerar uma sequência aleatória, o *R* disponibiliza duas possibilidades, sendo uma delas a criação da sequência a partir de uma distribuição uniforme, e a segunda, a partir de uma distribuição normal. A função `runif()` cria a sequência a partir da distribuição uniforme e necessita de três parâmetros:

- o número de elementos na sequência,
- o valor mínimo da sequência e o
- valor máximo da sequência.

A sintaxe e o resultado do uso dessa função são ilustrados a seguir.

```
In [141]: x <- runif(5,1,10)
x
```

```
7.08831313857809 · 1.30808151350357 · 7.94902605284005 ·
7.79995937412605 · 1.01541816093959
```

Para o caso da geração da sequência aleatória a partir de uma distribuição normal, a função `rnorm()` se aplica.

Elá recebe como parâmetros o *número de elementos desejado na sequência*, o *valor médio* e o *desvio da distribuição normal*.

Veja o exemplo:

```
In [142]: x <- rnorm(5, 4, 1)  
x
```

```
4.08065197850378 · 4.64182251965523 · 3.83496684557783 ·  
4.48128569135208 · 2.39432131701429
```

Alternativamente, caso seja necessário definir um vetor com uma sequência numérica específica, há a possibilidade do uso da função concatenação `c()`, como descrito a seguir:

```
In [143]: x <- c(30,10,40,20,80,50,70,100,90)  
x
```

30 · 10 · 40 · 20 · 80 · 50 · 70 · 100 · 90

A quantidade de operações possíveis em um vetor é bastante extensa. Para apresentar as principais, num contexto de utilidade para este curso, um resumo com os nomes, descrições breves e resultados de aplicação (em x , considerando o último exemplo) de algumas funções é apresentado na Tabela a seguir. É importante saber também que, em R , o primeiro elemento do vetor é o elemento de índice 1 e, portanto:

```
In [144]: x[1]
```

30

`min(x)` Encontra o valor mínimo em x.

```
In [145]: min(x)
```

10

`max(x)` Encontra o valor máximo em x.

In [20]: `max(x)`

100

`range(x)` Gera um vetor com os valores de $\min(x)$ e $\max(x)$.

```
In [21]: range(x)
```

10×100

`sort(x)` Gera uma versão ordenada de x.

```
In [22]: sort(x)
```

```
10 · 20 · 30 · 40 · 50 · 70 · 80 · 90 · 100
```

`min(x)` Gera um vetor de inteiros contendo os índices dos elementos de `x`, de acordo com a posição que o elemento assume em uma sequência ordenada.

Veja que 100 é o maior valor e assume a última posição em uma sequência ordenada e, por isso, ele recebe o maior índice, 9.

```
In [23]: rank(x)
```

```
3 · 1 · 4 · 2 · 7 · 5 · 6 · 9 · 8
```

```
In [24]: x
```

```
30 · 10 · 40 · 20 · 80 · 50 · 70 · 100 · 90
```

`order(x)` Um vetor de inteiros contendo os índices dos elementos de `x`, considerando a ordem crescente dos valores desses elementos. O primeiro inteiro no vetor resultante é 2, pois o menor elemento no vetor `x` (o número 10) está na segunda posição. Para usar a ordem decrescente, faça: `order(x, decreasing=TRUE)` .

```
In [25]: order(x)
```

2 · 4 · 1 · 3 · 6 · 7 · 5 · 9 · 8

```
In [26]: order(x, decreasing=TRUE)
```

8 · 9 · 5 · 7 · 6 · 3 · 1 · 4 · 2

`length(x)` Fornece o total de elementos do vetor x

```
In [27]: length(x)
```

9

`sum(x)` Calcula a soma de todos os elementos de x.

```
In [28]: sum(x)
```

490

`mean (x)` Calcula a média aritmética dos elementos de x.

In [29]: `mean (x)`

54.4444444444444

`median(x)` Calcula a mediana de x.

```
In [30]: median(x)
```

50

`var(x)` Calcula a variância de x.

In [31]: `var(x)`

1027.77777777778

`quantile(x)` Gera um vetor contendo o valor mínimo, quartil inferior, mediana, quartil superior e valor máximo de x.

In [32]: `quantile(x)`

0%: 10 25%: 30 50%: 50 75%: 80 100%: 100

Ainda é possível criar vetores com conjuntos de caracteres. Nesse caso, cada elemento deve estar entre aspas “ ” e separados por vírgula. Por exemplo:

```
In [33]: cardapio <- c("Filé à parmegiana", "Feijoada", "Batatas Fritas", "Lasanha à Bolonhesa",  
"Salada Caprese")
```

```
In [34]: cardapio
```

```
'Filé à parmegiana' · 'Feijoada' · 'Batatas Fritas' · 'Lasanha à Bolonhesa' ·  
'Salada Caprese'
```

O acesso a cada elemento de um vetor com conjuntos de caracteres pode ser feito de maneira semelhante ao de vetores numéricos. Ou seja, para acessar o elemento de número 5, tem-se:

```
In [35]: cardapio[5]
```

```
'Salada Caprese'
```

É possível fazer algumas manipulações em um vetor de conjuntos de caracteres. Por exemplo, para descobrir o número de caracteres em cada elemento do vetor, faça:

```
In [36]: nchar(cardapio)
```

```
17 · 8 · 14 · 19 · 14
```

Algumas das funções apresentadas na podem ser aplicadas a vetores de conjuntos de caracteres, como é o caso da função `length()`, para descobrir o tamanho do vetor, a função `sort()`, para obter uma ordenação em ordem alfabética, ou mesmo a função `order()`, para ter a posição dos elementos em ordem alfabética. Veja a execução desses três exemplos a seguir:

```
In [37]: length(cardapio)
```

```
5
```

```
In [38]: sort(cardapio)
```

```
'Batatas Fritas' · 'Feijoada' · 'Filé à parmegiana' · 'Lasanha à Bolonhesa' ·  
'Salada Caprese'
```

```
In [39]: cardapio
```

```
'Filé à parmegiana' · 'Feijoada' · 'Batatas Fritas' · 'Lasanha à Bolonhesa' ·  
'Salada Caprese'
```

```
In [40]: order(cardapio)
```

```
3 · 2 · 1 · 4 · 5
```

Data frames

A função `data.frame()`, que resulta em um tipo de dado especial chamado `data.frame`, permite estruturar dados em um formato semelhante ao que se encontra em planilhas eletrônicas.

Esse recurso permite que vetores de diferentes tipos sejam arranjados em uma estrutura de linhas e colunas, na qual cada coluna pode ser nomeada.

Por exemplo, considere que, para o exemplo anterior (o cardápio), se deseja acrescentar os preços de cada um dos pratos.

Um vetor de preços é criado como especificado a seguir:

```
In [146]: precos <- c(32.50, 44.00, 12.00, 35.50, 27.00)
```

A função `data.frame()` permite relacionar o vetor `cardapio` com o vetor `precos` em uma estrutura como uma tabela.

Assim, é possível ter dois vetores de tipos diferentes em uma mesma estrutura de dados, mas que, sobretudo, referem-se à mesma unidade observacional, o cardápio do restaurante. Em um `data.frame`, ainda é possível especificar um nome para cada coluna da tabela, passando parâmetros para a função formados por pares `nome = vetor`, como mostrado a seguir.

```
In [147]: data.frame(Cardápio = cardapio, Preços = precos)
```

A data.frame: 5 × 2

| Cardápio | Preços |
|---------------------|--------|
| <chr> | <dbl> |
| Filé à parmegiana | 32.5 |
| Feijoada | 44.0 |
| Batatas Fritas | 12.0 |
| Lasanha à Bolonhesa | 35.5 |
| Salada Caprese | 27.0 |

Importando dados de arquivos

Até este ponto, todos os exemplos discutidos fizeram uso de dados gerados na própria linha de comando.

Entretanto, na prática, é comum a necessidade de usar dados armazenados em planilhas, arquivos textos ou em bases de dados.

A importação de arquivos em R pode ser feita basicamente de duas maneiras: a partir da leitura de arquivo ou a partir da leitura de bases de dados.

A linguagem *R* permite, inclusive, a chamada de consultas expressas em **SQL**(*Structured Query Language*).

```
In [148]: iris <- read.table("https://gist.githubusercontent.com/netj/8836201/raw/6f9306ad213  
98ea43cba4f7d537619d0e07d5ae3/iris.csv",  
header=TRUE, sep=',')
```

In [149]: iris

A data.frame: 150 × 5

| sepal.length | sepal.width | petal.length | petal.width | variety |
|--------------|-------------|--------------|-------------|-----------|
| <dbl> | <dbl> | <dbl> | <dbl> | <chr> |
| 5.1 | 3.5 | 1.4 | 0.2 | Setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | Setosa |
| 4.7 | 3.2 | 1.3 | 0.2 | Setosa |
| 4.6 | 3.1 | 1.5 | 0.2 | Setosa |
| 5.0 | 3.6 | 1.4 | 0.2 | Setosa |
| 5.4 | 3.9 | 1.7 | 0.4 | Setosa |
| 4.6 | 3.4 | 1.4 | 0.3 | Setosa |
| 5.0 | 3.4 | 1.5 | 0.2 | Setosa |
| 4.4 | 2.9 | 1.4 | 0.2 | Setosa |
| 4.9 | 3.1 | 1.5 | 0.1 | Setosa |
| 5.4 | 3.7 | 1.5 | 0.2 | Setosa |
| 4.8 | 3.4 | 1.6 | 0.2 | Setosa |
| 4.8 | 3.0 | 1.4 | 0.1 | Setosa |
| 4.3 | 3.0 | 1.1 | 0.1 | Setosa |
| 5.8 | 4.0 | 1.2 | 0.2 | Setosa |
| 5.7 | 4.4 | 1.5 | 0.4 | Setosa |
| 5.4 | 3.9 | 1.3 | 0.4 | Setosa |
| 5.1 | 3.5 | 1.4 | 0.3 | Setosa |
| 5.7 | 3.8 | 1.7 | 0.3 | Setosa |
| 5.1 | 3.8 | 1.5 | 0.3 | Setosa |
| 5.4 | 3.4 | 1.7 | 0.2 | Setosa |
| 5.1 | 3.7 | 1.5 | 0.4 | Setosa |
| 4.6 | 3.6 | 1.0 | 0.2 | Setosa |
| 5.1 | 3.3 | 1.7 | 0.5 | Setosa |
| 4.8 | 3.4 | 1.9 | 0.2 | Setosa |
| 5.0 | 3.0 | 1.6 | 0.2 | Setosa |
| 5.0 | 3.4 | 1.6 | 0.4 | Setosa |
| 5.2 | 3.5 | 1.5 | 0.2 | Setosa |
| 5.2 | 3.4 | 1.4 | 0.2 | Setosa |
| 4.7 | 3.2 | 1.6 | 0.2 | Setosa |
| ... | ... | ... | ... | ... |
| 6.9 | 3.2 | 5.7 | 2.3 | Virginica |
| 5.6 | 2.8 | 4.9 | 2.0 | Virginica |
| 7.7 | 2.8 | 6.7 | 2.0 | Virginica |
| 6.3 | 2.7 | 4.9 | 1.8 | Virginica |

| sepal.length | sepal.width | petal.length | petal.width | variety |
|--------------|-------------|--------------|-------------|-----------|
| <dbl> | <dbl> | <dbl> | <dbl> | <chr> |
| 6.7 | 3.3 | 5.7 | 2.1 | Virginica |
| 7.2 | 3.2 | 6.0 | 1.8 | Virginica |
| 6.2 | 2.8 | 4.8 | 1.8 | Virginica |
| 6.1 | 3.0 | 4.9 | 1.8 | Virginica |
| 6.4 | 2.8 | 5.6 | 2.1 | Virginica |
| 7.2 | 3.0 | 5.8 | 1.6 | Virginica |
| 7.4 | 2.8 | 6.1 | 1.9 | Virginica |
| 7.9 | 3.8 | 6.4 | 2.0 | Virginica |
| 6.4 | 2.8 | 5.6 | 2.2 | Virginica |
| 6.3 | 2.8 | 5.1 | 1.5 | Virginica |
| 6.1 | 2.6 | 5.6 | 1.4 | Virginica |
| 7.7 | 3.0 | 6.1 | 2.3 | Virginica |
| 6.3 | 3.4 | 5.6 | 2.4 | Virginica |
| 6.4 | 3.1 | 5.5 | 1.8 | Virginica |
| 6.0 | 3.0 | 4.8 | 1.8 | Virginica |
| 6.9 | 3.1 | 5.4 | 2.1 | Virginica |
| 6.7 | 3.1 | 5.6 | 2.4 | Virginica |
| 6.9 | 3.1 | 5.1 | 2.3 | Virginica |
| 5.8 | 2.7 | 5.1 | 1.9 | Virginica |
| 6.8 | 3.2 | 5.9 | 2.3 | Virginica |
| 6.7 | 3.3 | 5.7 | 2.5 | Virginica |
| 6.7 | 3.0 | 5.2 | 2.3 | Virginica |
| 6.3 | 2.5 | 5.0 | 1.9 | Virginica |
| 6.5 | 3.0 | 5.2 | 2.0 | Virginica |

O conjunto de dados flor Iris ou conjunto de dados Iris de Fisher é um conjunto de dados multivariados introduzido pelo estatístico e biólogo britânico Ronald Fisher em seu artigo de 1936, O uso de múltiplas medições em problemas taxonômicos, como um exemplo de análise discriminante linear .

Às vezes, é chamado de conjunto de dados da íris de Anderson porque Edgar Anderson coletou os dados para quantificar a variação morfológica das flores da íris de três espécies relacionadas. Duas das três espécies foram coletadas na Península de Gaspé, "todas do mesmo campo, colhidas no mesmo dia e medidas ao mesmo tempo pela mesma pessoa com a mesma aparelho".

O conjunto de dados consiste em 50 amostras de cada uma das três espécies de Iris (**Iris setosa, Iris virginica e Iris versicolor**).

Quatro variáveis foram medidas em cada amostra: o **comprimento (length)** e a **largura (width)** das **sépalas** e **pétalas**, em centímetros.

Com base na combinação dessas quatro características, Fisher desenvolveu um modelo discriminante linear para distinguir as espécies umas das outras.

Iris Setosa



Iris Versicolor

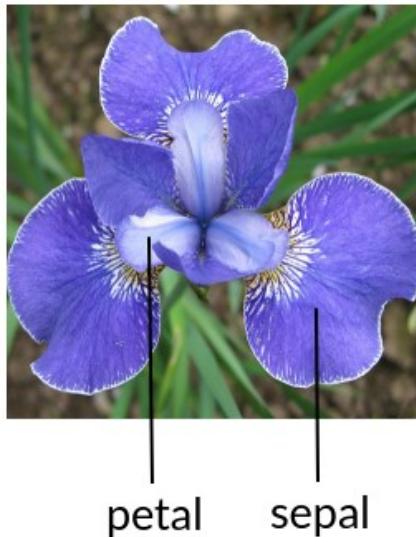


Iris Vergínica

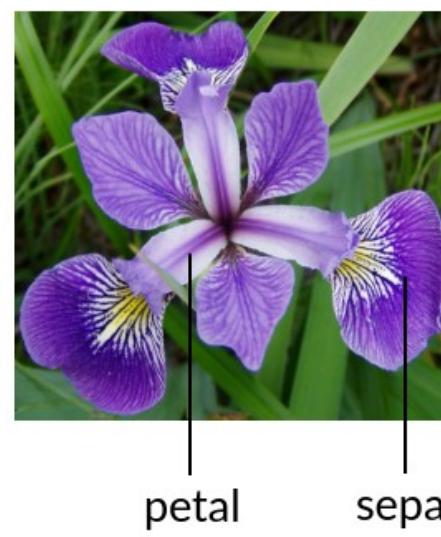


Sépulas (*Sepal*) e Pétulas (*Petal*)

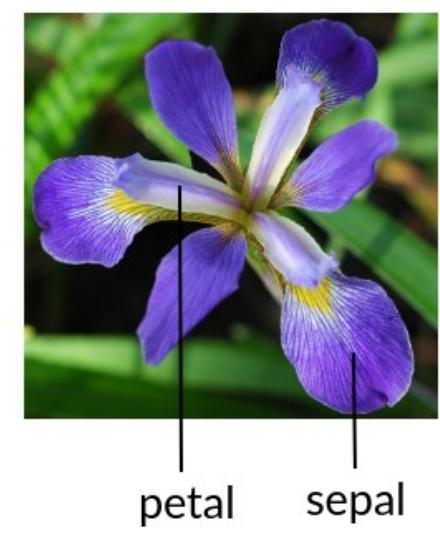
iris setosa



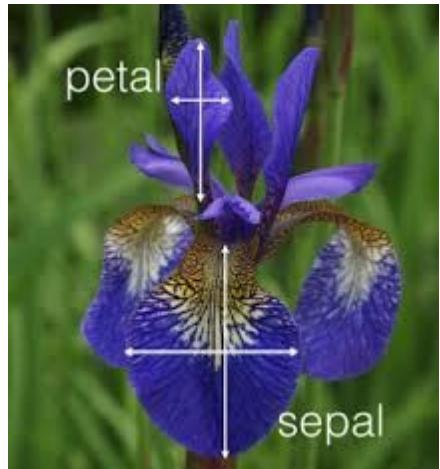
iris versicolor



iris virginica



Sépulas (*Sepal*) e Pétulas (*Petal*) e *Length* e *Width*



```
In [150]: head(iris)
```

A data.frame: 6 × 5

| | sepal.length | sepal.width | petal.length | petal.width | variety |
|---|--------------|-------------|--------------|-------------|---------|
| | <dbl> | <dbl> | <dbl> | <dbl> | <chr> |
| 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 |

```
In [47]: tail(iris)
```

A data.frame: 6 × 5

| | sepal.length | sepal.width | petal.length | petal.width | variety |
|-----|--------------|-------------|--------------|-------------|-----------|
| | <dbl> | <dbl> | <dbl> | <dbl> | <chr> |
| 145 | 6.7 | 3.3 | 5.7 | 2.5 | Virginica |
| 146 | 6.7 | 3.0 | 5.2 | 2.3 | Virginica |
| 147 | 6.3 | 2.5 | 5.0 | 1.9 | Virginica |
| 148 | 6.5 | 3.0 | 5.2 | 2.0 | Virginica |
| 149 | 6.2 | 3.4 | 5.4 | 2.3 | Virginica |
| 150 | 5.9 | 3.0 | 5.1 | 1.8 | Virginica |

Noções Básicas de Gráficos

R inclui funções para a criação de vários tipos de gráficos, como gráfico de barras, de setores, de linhas, de dispersão etc. Essas funções estão disponíveis no pacote `graphics`, nativo da linguagem.

Há ainda outros tipos de gráficos possíveis, mais sofisticados, cujas funções para criação estão disponíveis no pacote `lattice`, por exemplo.

Nesta seção, são apresentados os principais gráficos úteis no entendimento dos conceitos apresentados neste curso. A seguir apresenta um resumo das funções para criação dos gráficos apresentadas nesta aula.

`plot()` Cria um gráfico de dispersão ou um gráfico de linhas

`barplot()` Cria um gráfico de colunas ou de barras

`pie()` Cria um gráfico de setores ou do tipo pizza

`boxplot()` Cria um gráfico de caixas

Gráfico de dispersão

Um gráfico de dispersão (scatter plot), ou espalhamento, permite a visualização da distribuição dos exemplares de um conjunto de dados.

A função `plot()` constrói esse gráfico a partir dos valores atribuídos a uma série de parâmetros (os principais estão listados na sequência).

De maneira genérica, considerando que dois vetores serão usados na plotagem dos gráficos, a função é definida como:

```
plot(x, y = NULL, type = "p", xlim = NULL, ylim = NULL, log = "",  
main = NULL, sub = NULL, xlab = NULL, ylab = NULL, col="blue")
```

`x, y` Os dados a serem plotados, que devem estar separados em dois vetores.

`type` Tipo de marcador do gráfico: `p`, para pontos, `l`, para linhas, `o`, para pontos e linhas.

`xlim` Um vetor com os limites inferior e superior do eixo x.

`ylim` Um vetor com os limites inferior e superior do eixo y.

`log` Uma sequência de caracteres deve conter:

- `x` caso se queira que o eixo x esteja em escala logarítmica,
- `y` para o eixo y se quiser que ele esteja em escala logarítmica ou,
- no caso em que ambos os eixos devam estar nessa escala, usar a sequência `xy` ou `yx`.

`main` O título principal do gráfico.

`sub` O subtítulo do gráfico.

`xlab` O rótulo do eixo x.

`ylab` O rótulo para o eixo y.

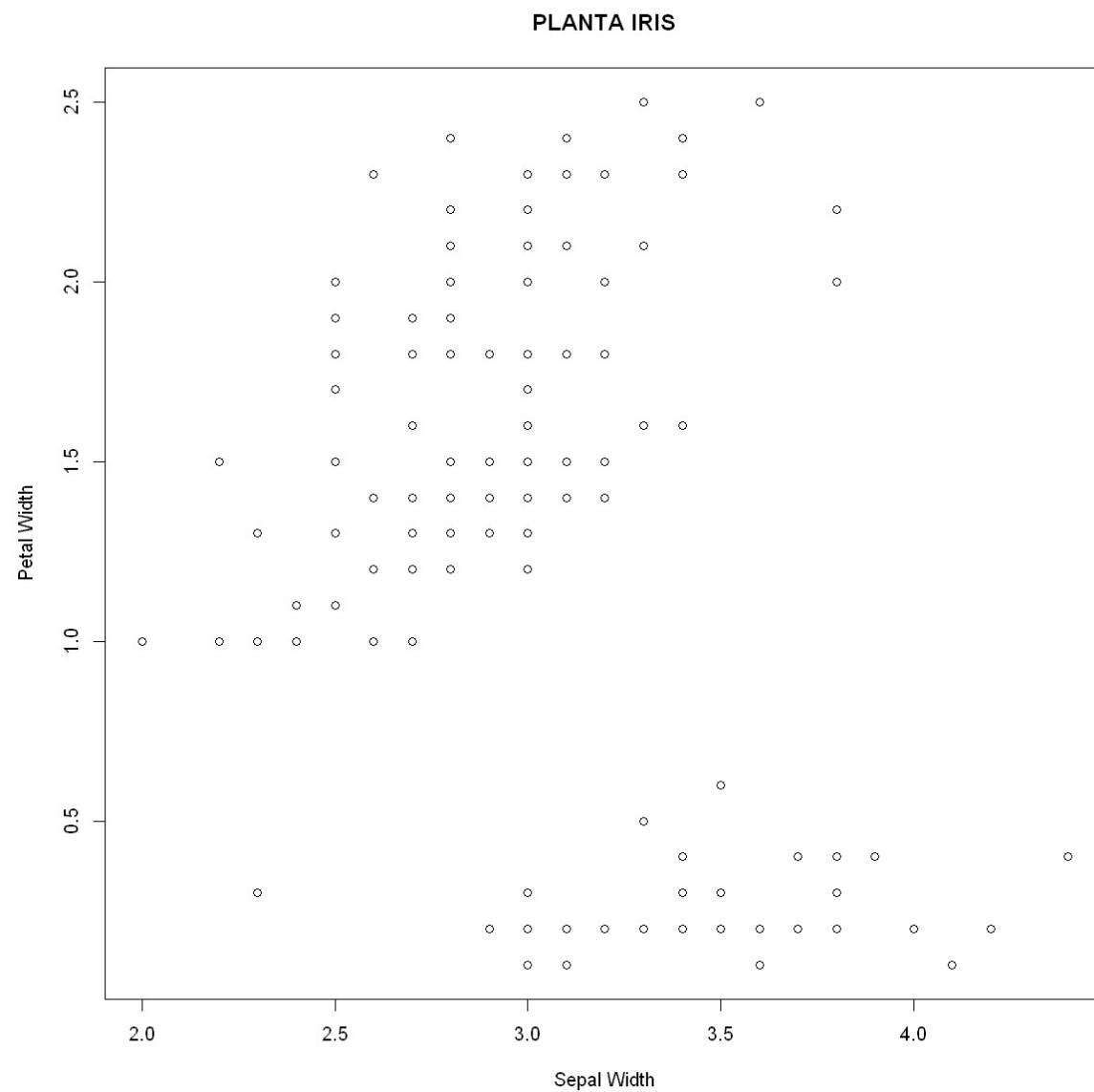
`col` Cor dos marcadores: red, blue ou purple.

Para exemplificar o uso da função `plot()` na construção de um gráfico de dispersão, considere dois atributos descritivos do conjunto de dados Iris:

- `Sepal.Width` e
- `Petal.Width`.

Inicialmente, esses atributos devem ser colocados em duas variáveis do tipo vetor (`x` e `y`), e, em seguida, a função pode ser executada.

```
In [151]: x <- iris$sepal.width  
y <- iris$petal.width  
plot(x,y,type='p',main = "PLANTA IRIS",xlab = "Sepal Width",ylab= "Petal Width")
```



No caso do conjunto de dados Iris, a informação referente ao rótulo de cada exemplar está disponível, e pode ser interessante usar essa informação na criação dos gráficos.

Duas estratégias para uso dessa informação são apresentadas aqui.

A primeira delas é com o uso da função `text()`, que permite atribuir a informação de rótulo a cada ponto no gráfico (cada ponto no gráfico corresponde a um exemplar do conjunto de dados).

Os principais parâmetros dessa função estão detalhados na sequencia

`x, y` Os dados a serem plotados, que devem estar separados em dois vetores.

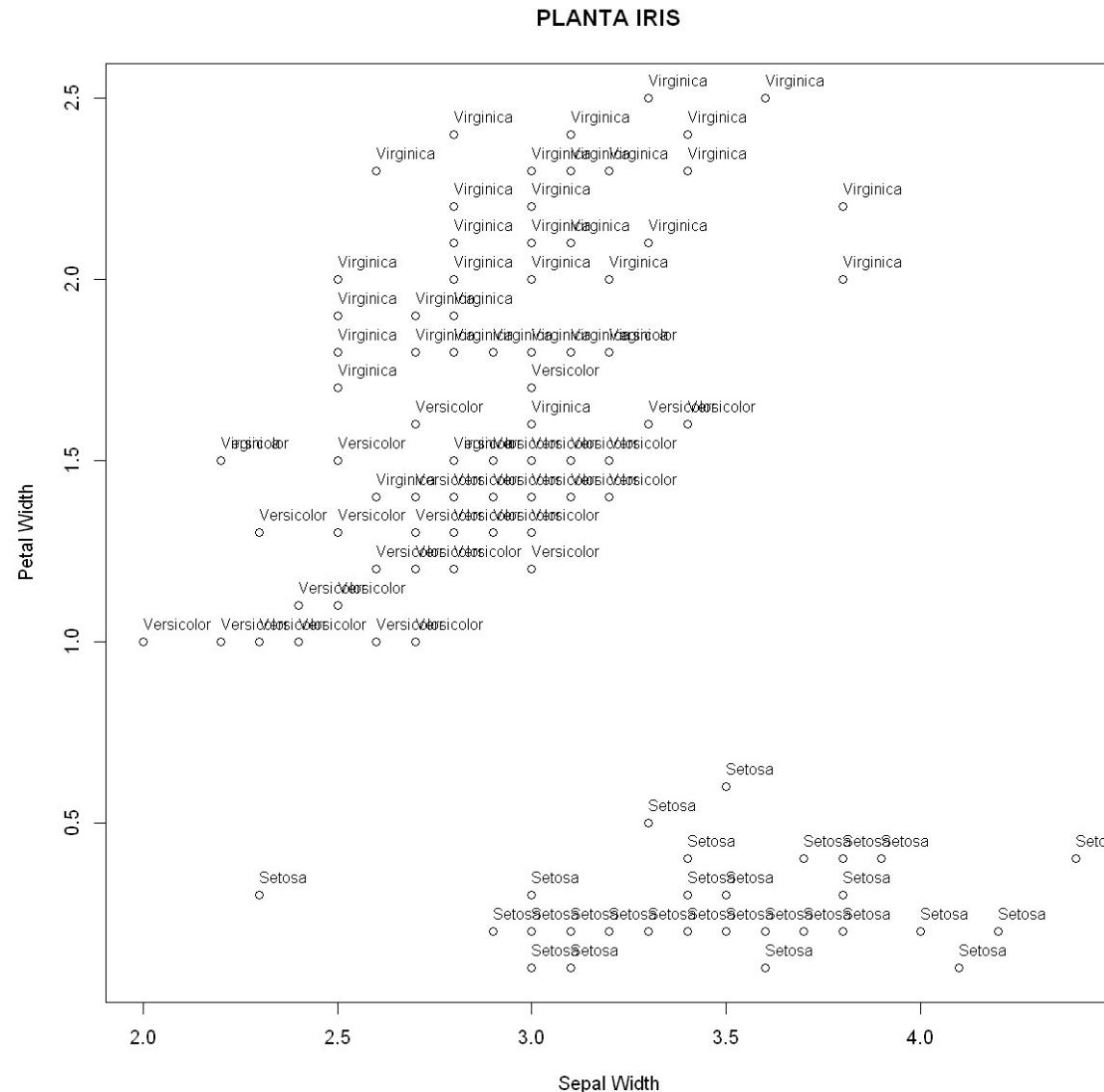
`labels` Uma variável com as informações de rótulo de cada exemplar do conjunto de dados.

`cex` Tamanho do texto que será plotado.

`adj` Posicionamento do texto (para que fique posicionado de forma adjacente ao marcador).

```
In [152]: x <- iris[, 'sepal.width']
y <- iris[, 'petal.width']
especie <- iris[, 'variety']
```

```
In [103]: library(repr) # A Biblioteca repr serve para ajusta o tamanho dos gráficos em R
options(repr.plot.width=10,repr.plot.height=10) # Ajuste de um gráfico para 10 de largura e 10 de altura
plot(x,y,type='p',main = 'PLANTA IRIS',xlab = 'Sepal Width',ylab= 'Petal Width')
text(x,y,labels=especie,cex=0.8,adj=c(0,-1))
```



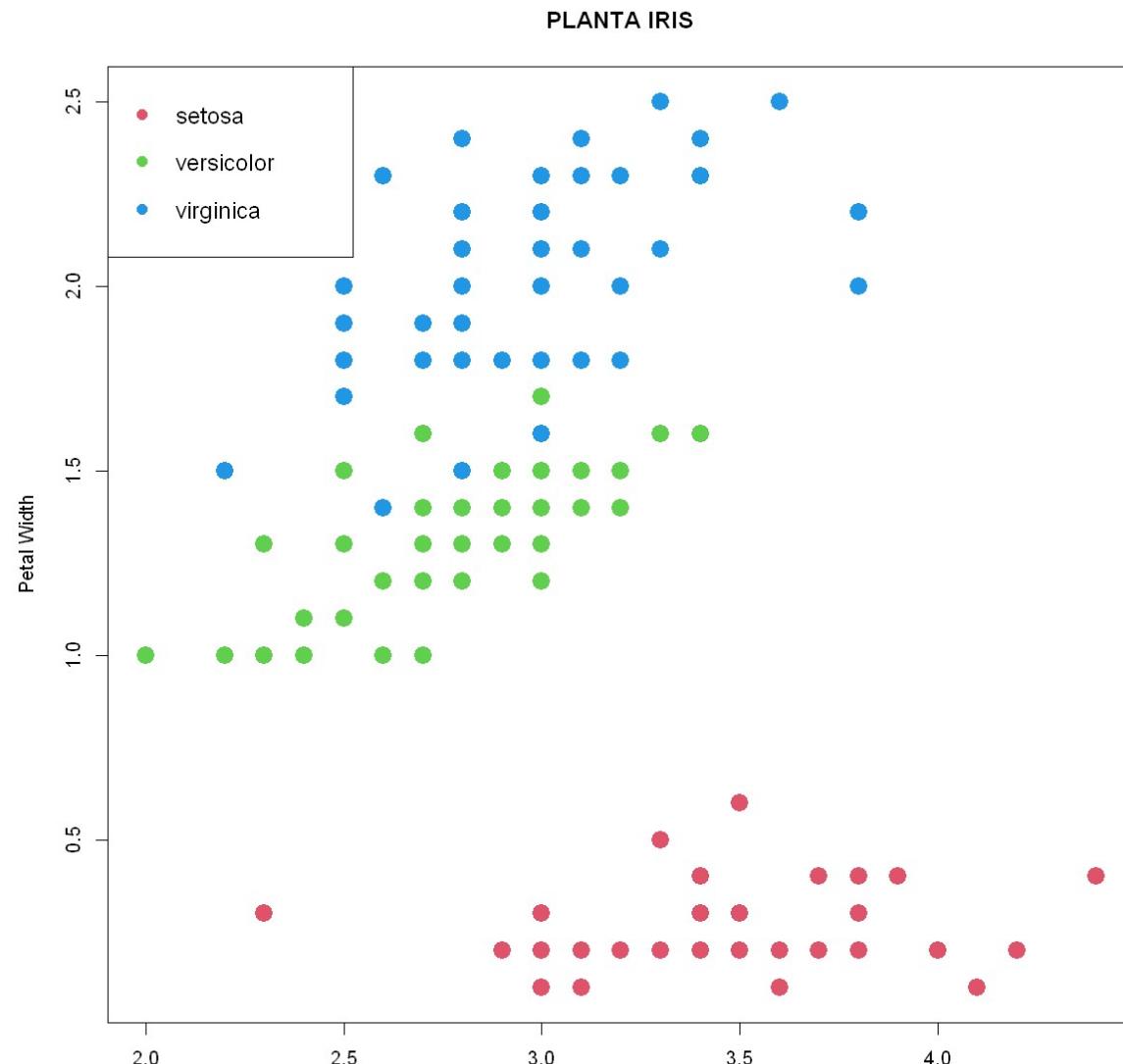
Uso da Função Points

A segunda estratégia para explorar a informação sobre rótulos é por meio do uso da função `points()`. Com essa função, é possível explorar recursos referentes a cores e marcadores. A tabela a seguir mostra os valores de parâmetros da função `points`.

Valores para os parâmetros que usam cores, neste caso, exemplificando o `col` da função

| Cor | Nome | Inteiro | Hexadecimal |
|----------|-------|---------|-------------|
| Preta | black | 1 | 000000 |
| Vermelha | red | 2 | FF0000 |
| Verde | green | 3 | 00FF00 |
| Azul | blue | 4 | 0000FF |

```
In [104]: plot(x,y, type='n', main = 'PLANTA IRIS',xlab = 'Sepal Width',ylab= 'Petal Width')
points(x[1:50],y[1:50],col='2',cex=2,pch=19)
points(x[51:100],y[51:100],col='3',cex=2,pch=19)
points(x[101:150],y[101:150],col='4',cex=2,pch=19)
legend("topleft",
       legend = c('setosa','versicolor','virginica'),
       pch=c(19,19,19),
       cex = 1.2,
       text.col = "black",
       col=c('2','3','4'))
```



Para explorar o uso da função `plot()`, algumas informações adicionais sobre o conjunto de dados Iris precisam ser consideradas. Esse conjunto de dados contém 150 exemplares, distribuídos uniformemente em três classes:

- 50 exemplares da classe setosa
- 50 exemplares da classe versicolor e
- 50 exemplares da classe virgínica

Além disso, será necessário fazer uma plotagem inicial, sem exibição dos marcadores de forma adequadamente com a adição opcional de uma legenda (função `legend()` - veja um resumo das opções de marcadores).

| Parâmetros | Descrição |

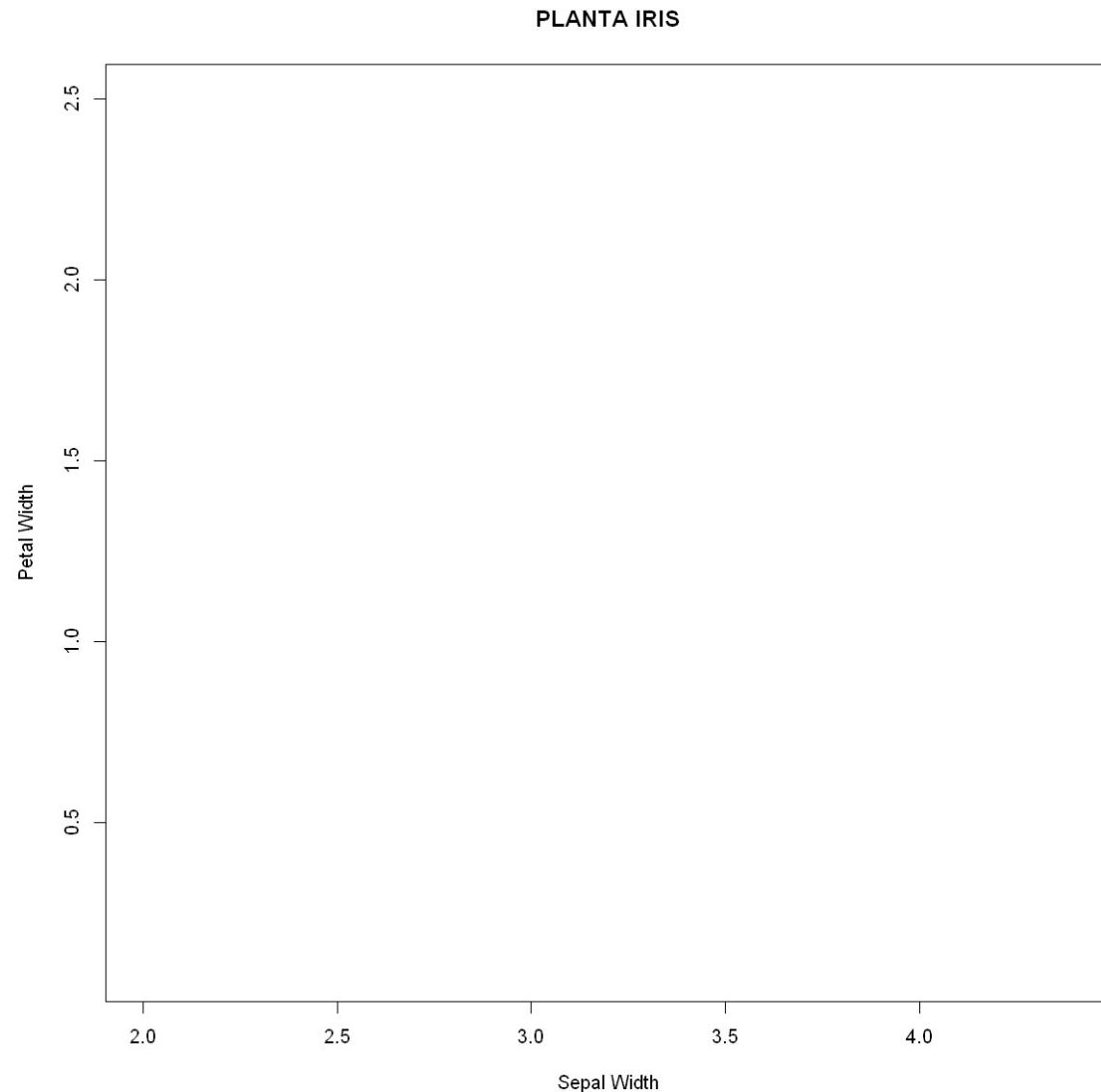
|-----|-----|
| pos ou locator | A posição pode ser um vetor de coordenadas (x,y) ou então pode ser desejado da legenda. | | names | Os nomes que aparecerão na legenda. Podem ser definidos para cada marcador em uso. Uma indicação por rótulo utilizado. | | Col | Parâmetro que indica a cor de

|

As três sequências de comandos a seguir se referem a esse procedimento, e os gráficos resultantes de cada uma são ilustrados a seguir

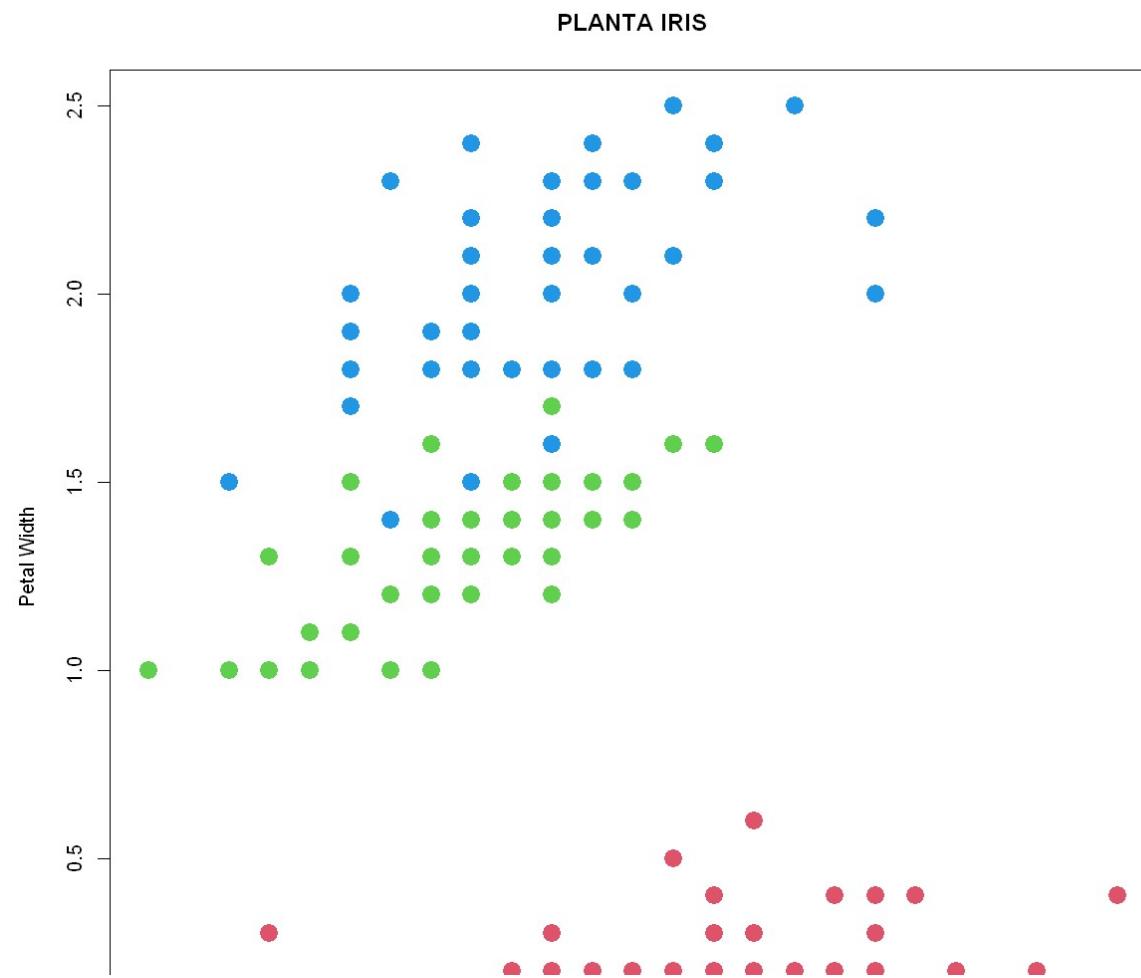
Sequencia 1 plot() de base

```
In [105]: plot(x,y, type='n', main = 'PLANTA IRIS',xlab = 'Sepal Width',ylab= 'Petal Width')
```



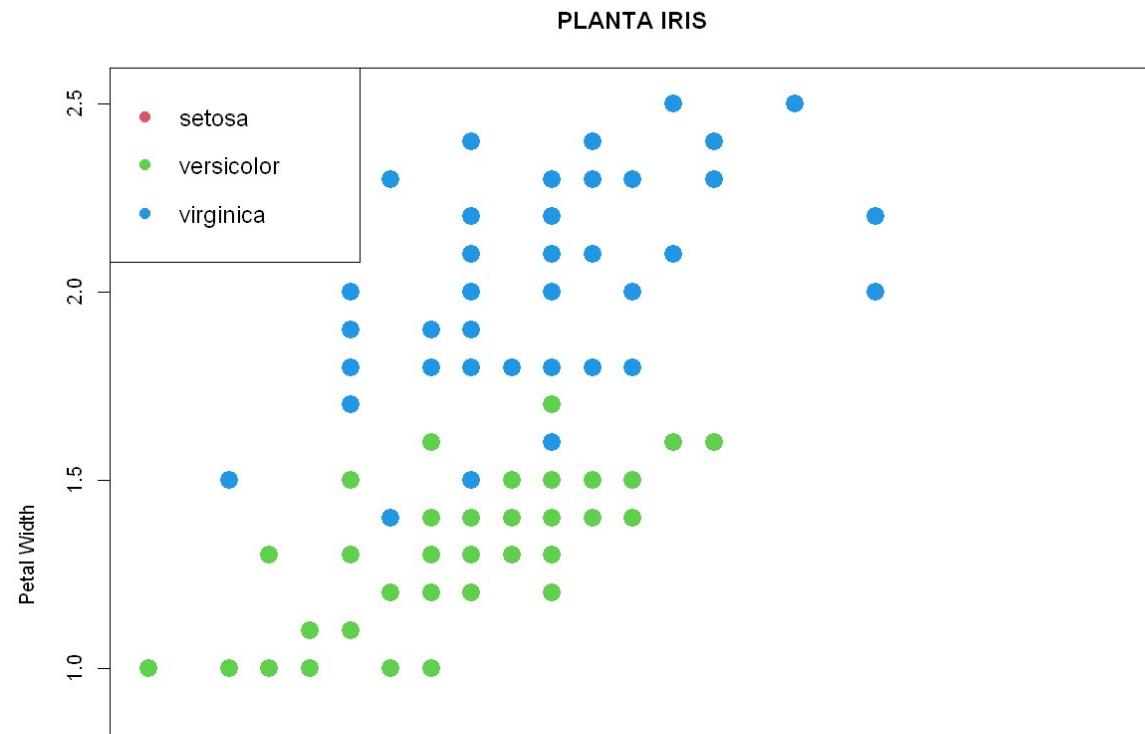
Sequencia 2 plot () + points () de base

```
In [106]: plot(x,y, type='n', main = 'PLANTA IRIS',xlab = 'Sepal Width',ylab= 'Petal Width')
points(x[1:50],y[1:50],col='2',cex=2,pch=19)
points(x[51:100],y[51:100],col='3',cex=2,pch=19)
points(x[101:150],y[101:150],col='4',cex=2,pch=19)
```



Sequencia 3 `plot () + points () + legend ()` de base

```
In [107]: plot(x,y, type='n', main = 'PLANTA IRIS',xlab = 'Sepal Width',ylab= 'Petal Width')
points(x[1:50],y[1:50],col='2',cex=2,pch=19)
points(x[51:100],y[51:100],col='3',cex=2,pch=19)
points(x[101:150],y[101:150],col='4',cex=2,pch=19)
legend("topleft",
       legend = c('setosa','versicolor','virginica'),
       pch=c(19,19,19),
       cex = 1.2,
       text.col = "black",
       col=c('2','3','4'))
```



Mudando o formato dos Points

```
In [108]: plot(x,y, type='n', main = 'PLANTA IRIS',xlab = 'Sepal Width',ylab= 'Petal Width')
points(x[1:50],y[1:50],col='2',cex=2,pch=1)
points(x[51:100],y[51:100],col='3',cex=2,pch=2)
points(x[101:150],y[101:150],col='4',cex=2,pch=3)
legend("topleft",
       legend = c('setosa','versicolor','virginica'),
       pch=c(19,19,19),
       cex = 1.2,
       text.col = "black",
       col=c('2','3','4'))
```

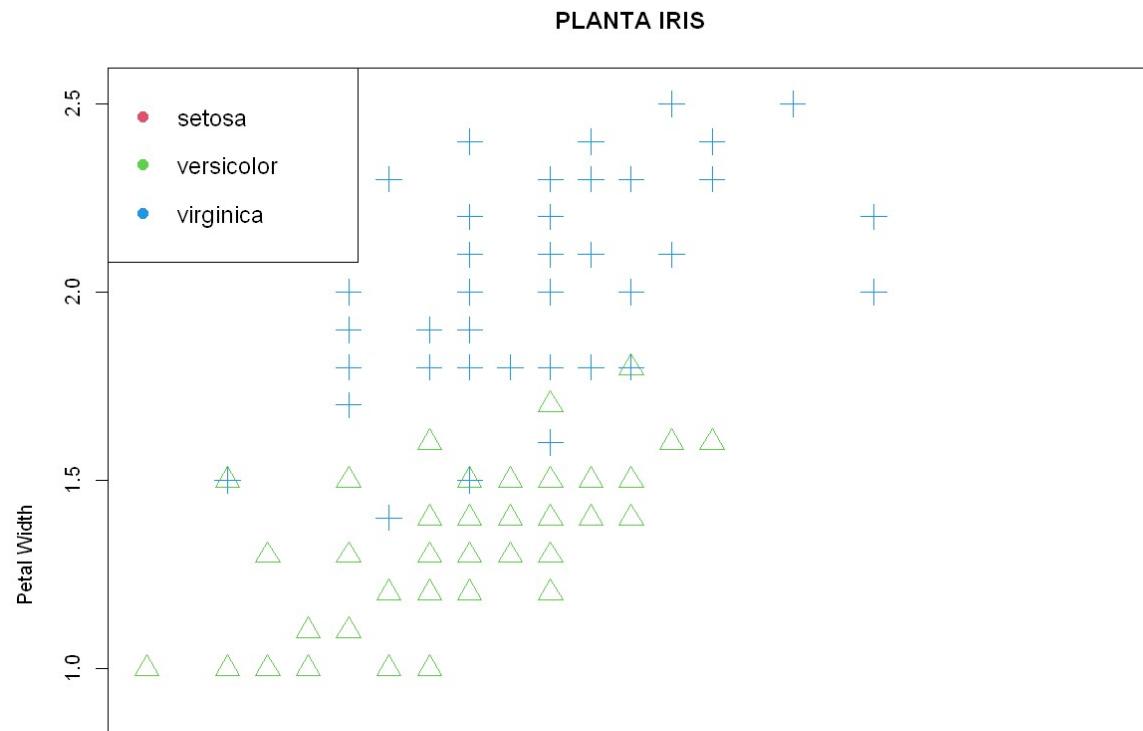


Gráfico genérico com diversos tipo de points

In [109]:

```
x1 <- 1:10
y1 <- 1:10
plot(x1, y1, type = "n")
points(1:10, rep(8, 10), pch = 0:9, col = 1:10, cex = 1.5)
points(1:10, rep(6, 10), pch = 10:19, col = 1:10, cex = 1.5)
points(1:10, rep(4, 10), pch = c(".", "a", "o", "v", "s", "c", "&", "$", "@", "%"),
col = 1:10, cex = 1.5)
```

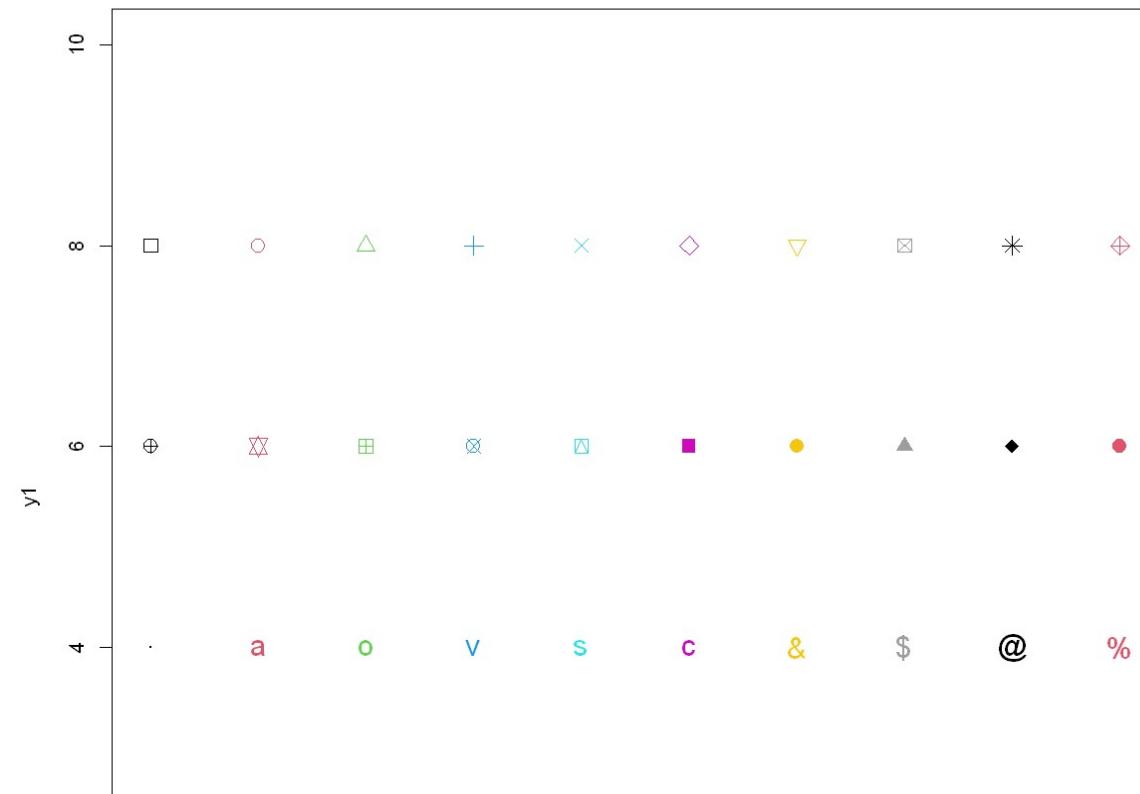


Gráfico genérico com diversos tipo de points - aumentando o tamanho

```
In [110]: x1 <- 1:10  
y1 <- 1:10  
plot(x1, y1, type = "n")  
points(1:10, rep(8, 10), pch = 0:9, col = 1:10, cex = 3.5)  
points(1:10, rep(6, 10), pch = 10:19, col = 1:10, cex = 3.5)  
points(1:10, rep(4, 10), pch = c(".", "a", "o", "v", "s", "c", "&", "$", "@", "%"),  
col = 1:10, cex = 3.5)
```

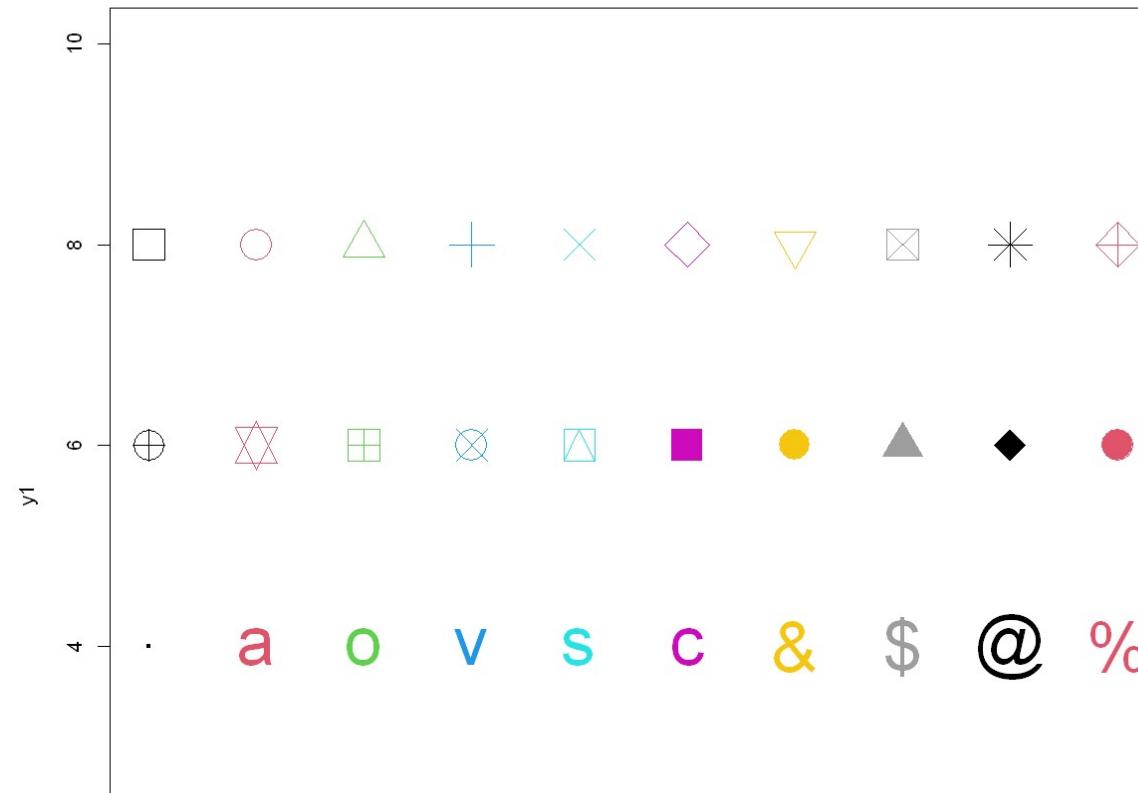


Gráfico de linhas

Com a função `plot()`, também é possível imprimir gráficos de linhas.

Um gráfico de linhas pode ser usado para suportar a análise de atributos individualmente.

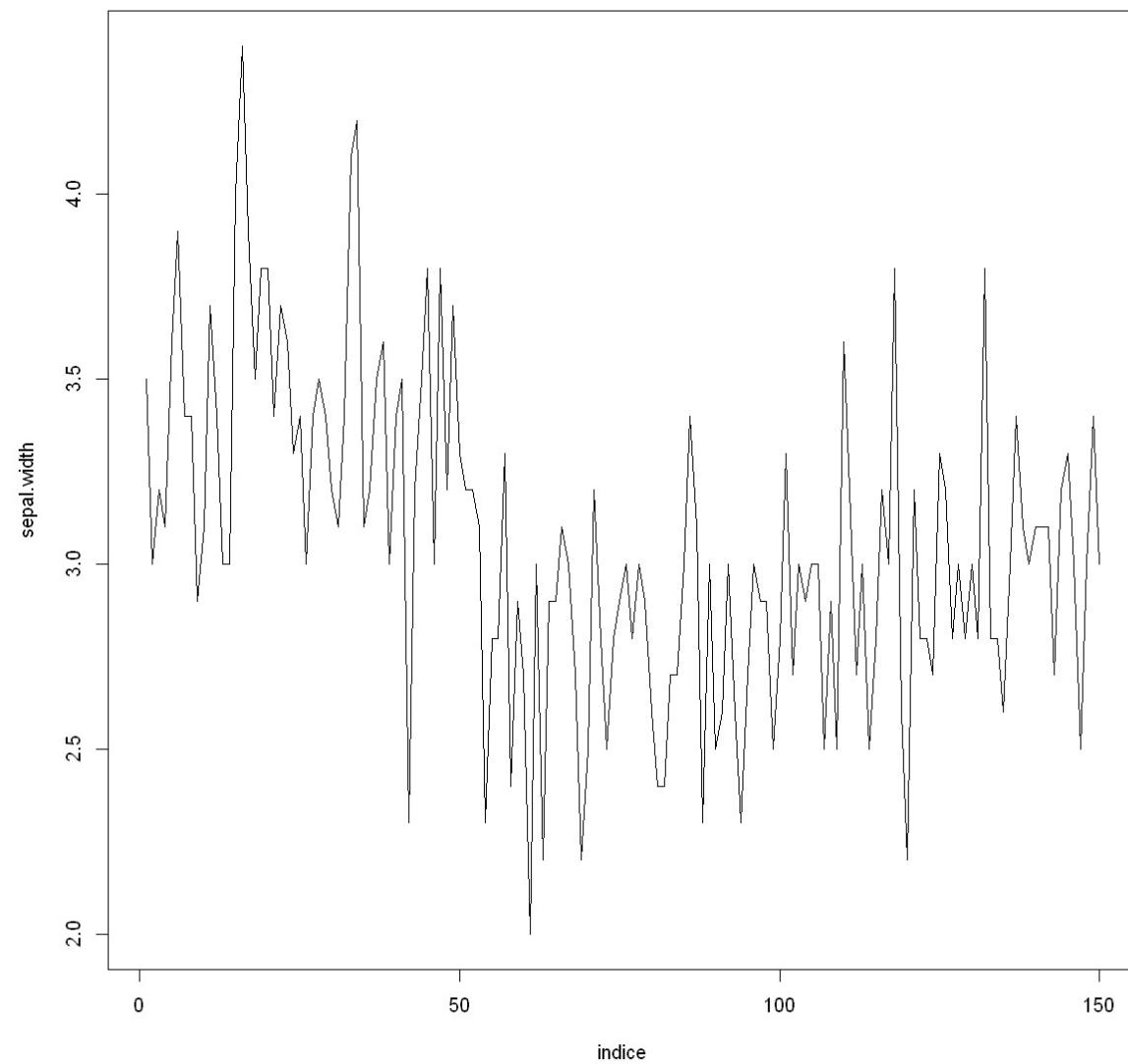
Em geral, no eixo x de um gráfico de linhas, estão os índices dos exemplares do conjunto de dados ou os tempos em que foram coletados, caso o conjunto de dados venha de uma série temporal de valores.

Para exemplificar um gráfico de linhas, o atributo **Sepal Width** será utilizado.

Assim como no gráfico de dispersão, inicialmente, é necessário executar a função `plot()` com `type = n`, e, então, imprimir com a função `lines()` a série de valores que darão origem à linha no gráfico.

Veja a sequência de comandos necessários a seguir, e o gráfico gerado.

```
In [112]: plot(x,type='n',xlab='indice',ylab='sepal.width')
lines(iris['sepal.width'])
```

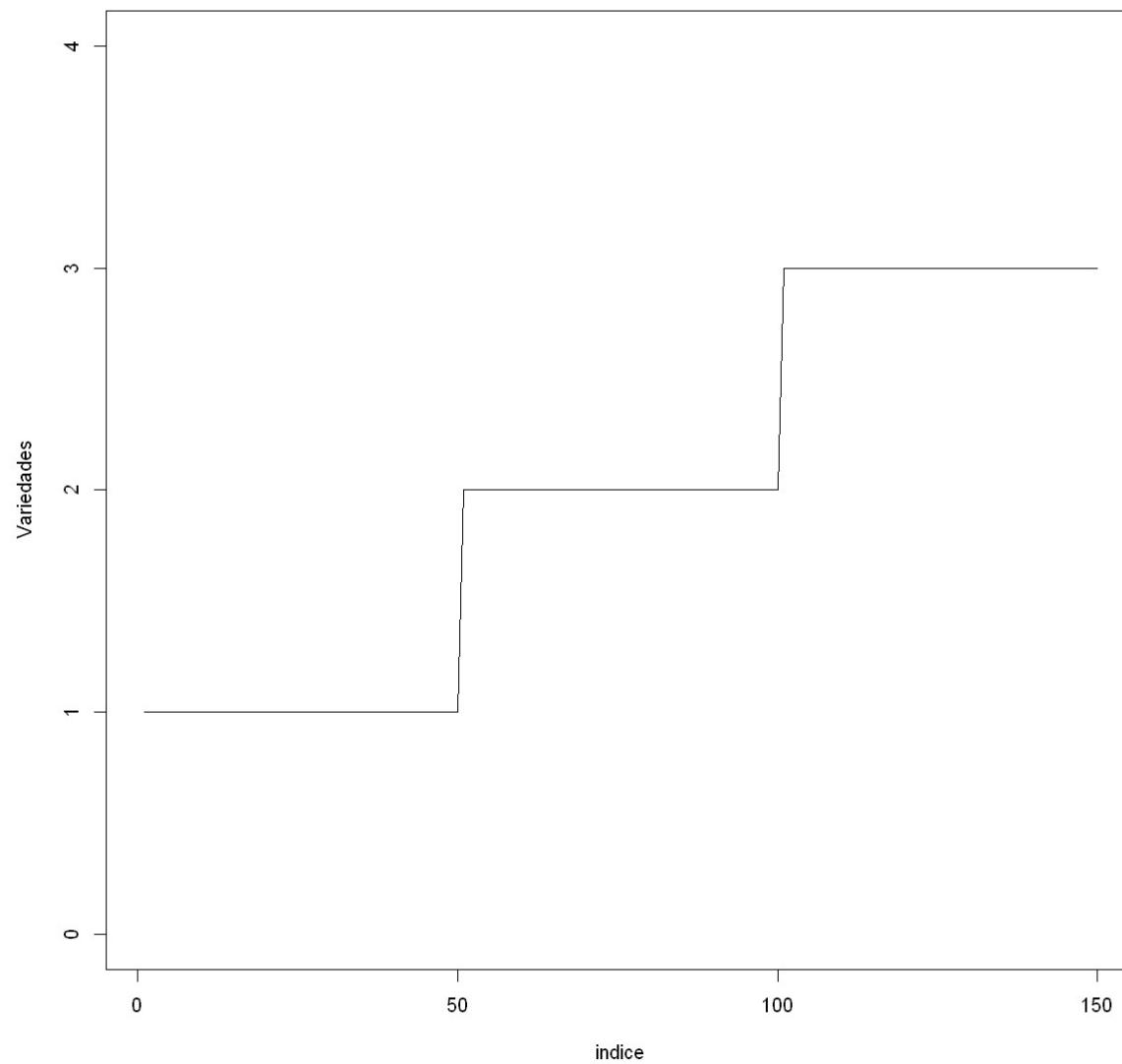


A plotagem do comportamento assumido por valores de atributos categóricos também é possível.

O gráfico apresentado na Figura abaixo ilustra a impressão do atributo de rótulo (**variety**) do conjunto de dados Iris. No gráfico, os nomes das espécies foram transformados em números apenas para fins de diferenciação (não há, porém, nenhuma relação de ordem entre as espécies).

Com essa visualização, é possível perceber a distribuição uniforme das classes, como discutido anteriormente neste texto. Os comandos necessários para obter esse gráfico são apresentadas a seguir:

```
In [117]: plot(x,type='n',xlab='indice',ylab='Variedades',ylim=c(0,4))  
lines(iris['variety'])
```



Na chamada da função `plot()` desse exemplo, foi utilizado um novo parâmetro, o `ylim`. Esse parâmetro permite definir os valores mínimos e máximos do eixo y. Para a mesma parametrização do eixo x, o parâmetro `xlim` deve ser usado. Ainda há outros tipos de linhas que podem ser usados nesses gráficos. A configuração do tipo da linha deve ser realizada por meio do parâmetro `lty` (**line type**); veja alguns dos tipos disponíveis na Figura abaixo. Adicionalmente ao tipo de linha, é possível parametrizar sua espessura por meio do parâmetro `lwd` (**line weight***).

Grafico Genérico com diversos tipos de linhas

In [118]:

```
x2 <- 1:9
y2 <- 1:9
plot(x2, y2, type = "n")
lines(c(2, 8), c(8, 8), lwd = 6)
lines(c(2, 8), c(7, 7), lty = 2, lwd = 6)
lines(c(2, 8), c(6, 6), lty = 3, lwd = 6)
lines(c(2, 8), c(5, 5), lty = 4, lwd = 6)
lines(c(2, 8), c(4, 4), lty = 5, lwd = 6)
lines(c(2, 8), c(3, 3), lty = 6, lwd = 6)
```

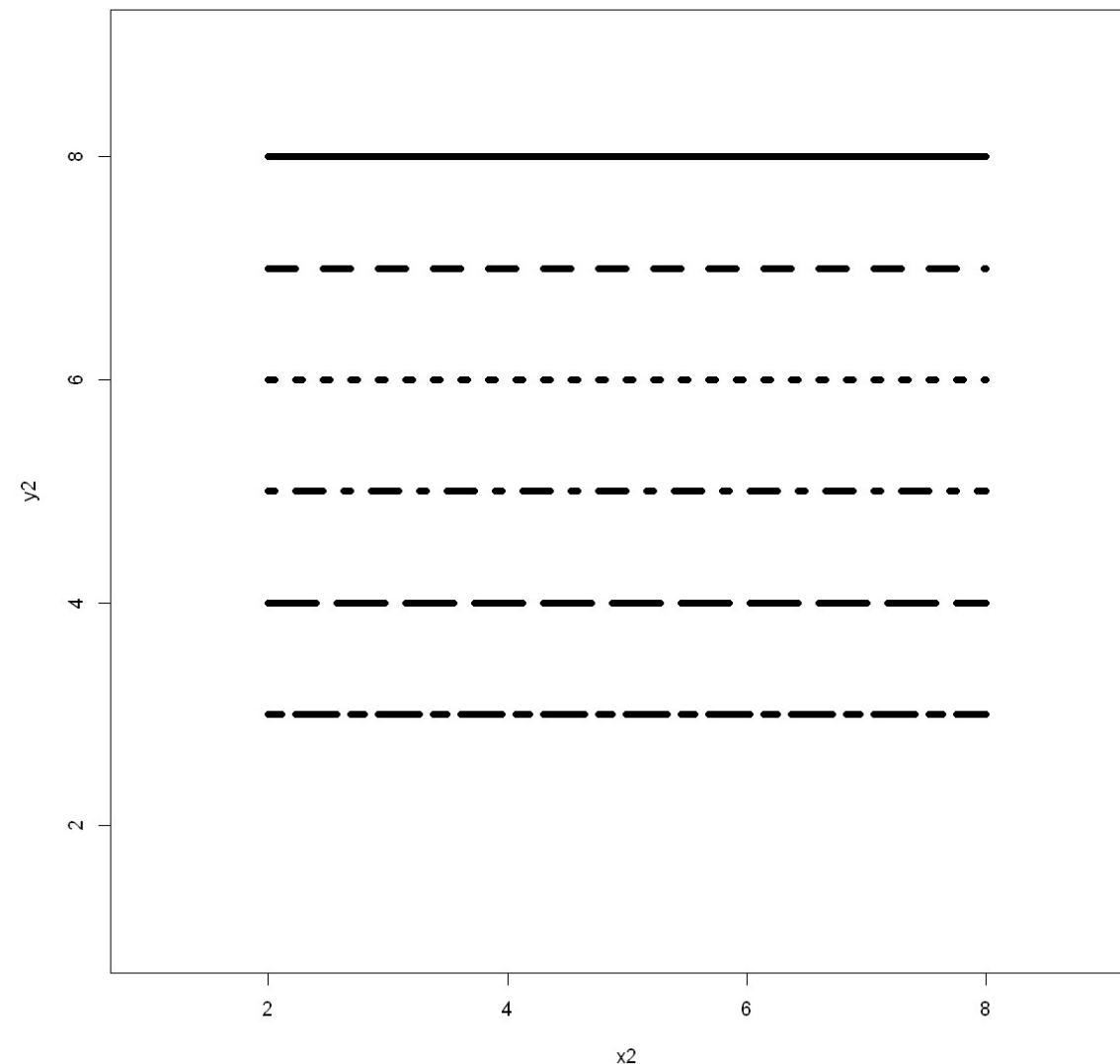


Gráfico de barras

Alternativamente ao uso do gráfico de linhas, a observação da quantidade de exemplares que possuem determinado valor para um atributo descritivo categórico pode ser feita usando um gráfico de barras .

A utilidade de ambos, gráfico de linhas e gráfico de barras, é similar, mas, no caso do gráfico de barras, as quantidades são agregadas em uma contagem.

Para criação desse gráfico em *R*, é preciso que os valores estejam organizados em uma matriz, sendo que, em uma das colunas, estão os índices, e, em outra, estão as quantidades agregadas.

Em geral, os valores a serem observados por meio de um gráfico de barras devem passar por algum tipo de transformação, como uma agregação.

Agregações podem ser feitas com uso de funções de soma, média, mediana, contagem etc. Por exemplo, considere a contagem de exemplares pertencentes a cada categoria do atributo de rótulo **variety**.

A etapa de transformação dos valores pode ser realizada com a aplicação da função `tapply()`.

Os principais parâmetros dessa função estão definidos na Tabela a seguir : Parâmetros da função **TAPPLY**

| Parâmetros | Descrição |
|-----------------------|--|
| <code>index</code> | Índices da série de valores. |
| <code>x</code> | Vetor com os valores. |
| <code>function</code> | Agregação a ser aplicada nos valores do vetor, como <code>sum</code> , <code>mean</code> , <code>max</code> e <code>min</code> |

Para criar o gráfico, a seguinte sequência de comandos deve ser executada. O gráfico resultante está ilustrado na Figura a seguir

```
In [121]: quantidade <- tapply(rep(1,150),iris['variety'],sum)
barplot(quantidade,xlab='Espécies',ylab='Frequência')
```

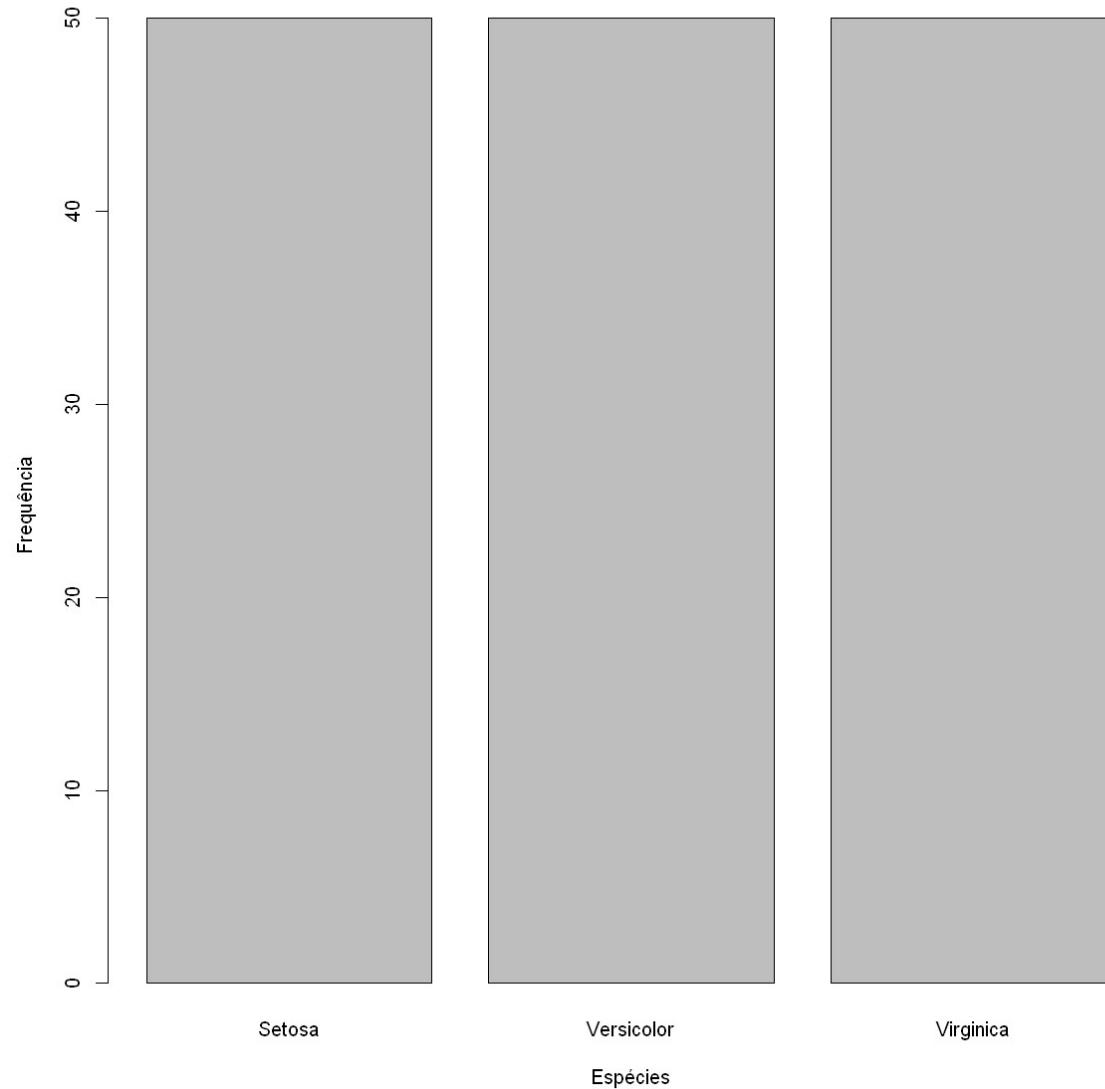
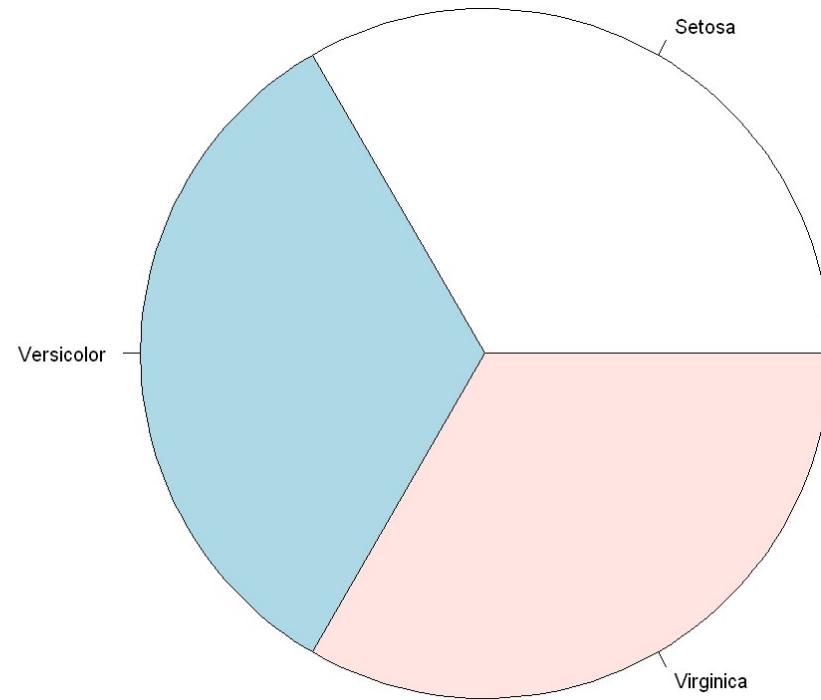


Gráfico de setores

O gráfico de setores , ou gráfico de pizza, como é popularmente chamado, tem o mesmo princípio do gráfico de barras, porém, a apresentação gráfica é diferente.

O gráfico de setores para a variável quantidade, criada para o exemplo anterior, pode ser impresso com o uso da função pie(), por meio da execução do comando:

```
In [122]: pie(quantidade)
```



Note que ambos os tipos de gráfico, gráfico de barras e gráfico de setores, permitem que sejam adicionadas informações como nomes, cores, legenda etc., usando as mesmas opções já discutidas anteriormente para a função `plot()`.

Gráfico de caixas

O gráfico de caixas, ou **boxplot**, é um instrumento que permite identificar a variação dos valores de uma variável, apresentando como resultados a localização dos 50% valores centrais, mediana e valores extremos (mínimo e máximo).

Esses resultados podem ser vistos em R pelo uso da função `summary()` e por meio da plotagem de um gráfico via função `boxplot()`.

Veja, a seguir, um exemplo da execução da função `summary()` sobre o conjunto de dados Iris.

```
In [123]: summary(iris)
```

| | sepal.length | sepal.width | petal.length | petal.width |
|---------|--------------|-------------|--------------|-------------|
| Min. | :4.300 | :2.000 | :1.000 | :0.100 |
| 1st Qu. | :5.100 | :2.800 | :1.600 | :0.300 |
| Median | :5.800 | :3.000 | :4.350 | :1.300 |
| Mean | :5.843 | :3.057 | :3.758 | :1.199 |
| 3rd Qu. | :6.400 | :3.300 | :5.100 | :1.800 |
| Max. | :7.900 | :4.400 | :6.900 | :2.500 |

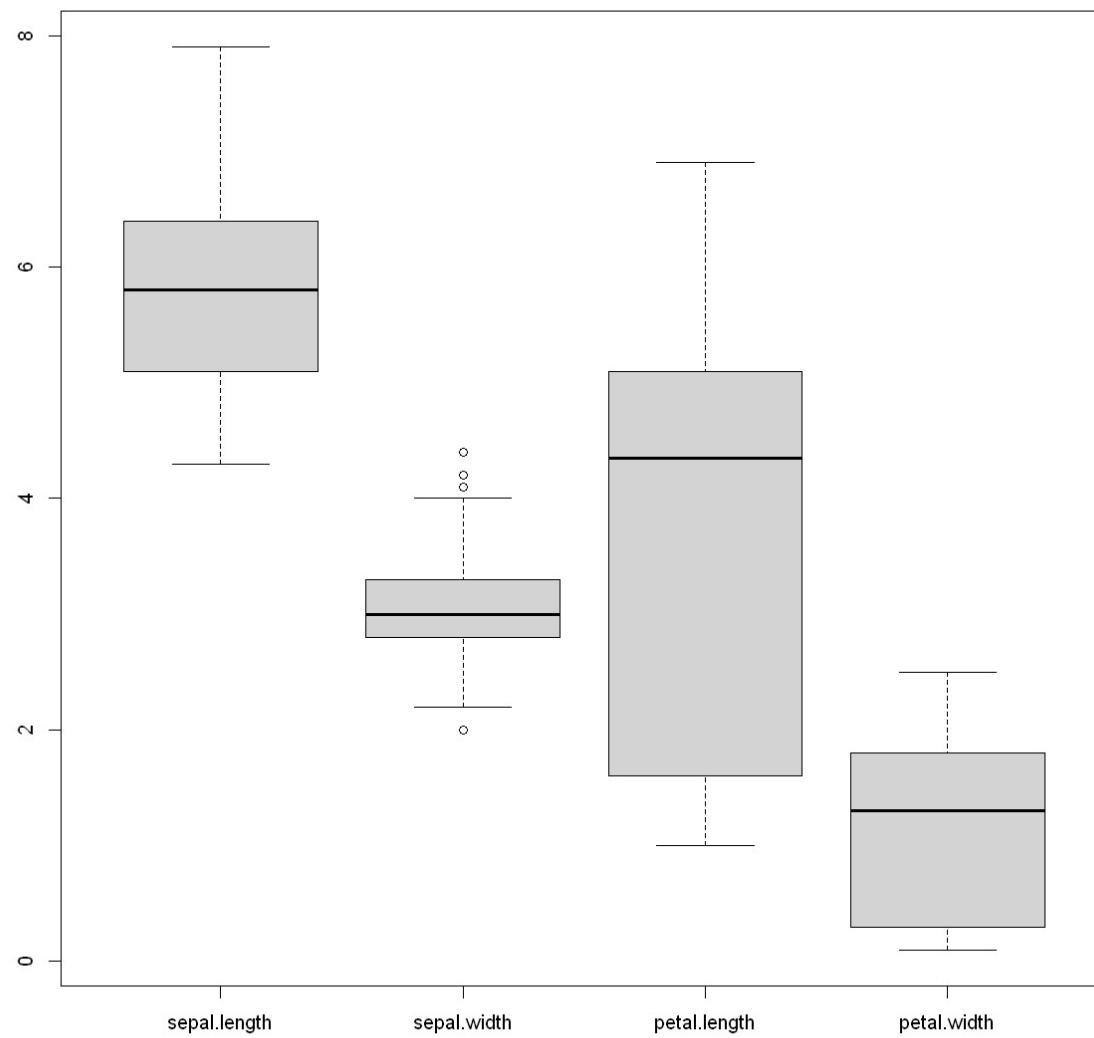
variety
Length:150
Class :character
Mode :character

A plotagem desses dados via um gráfico de caixas gera, para cada variável (ou atributo do conjunto de dados), uma representação dividida em quatro partes:

- a primeira parte (inferior), representada por uma linha pontilhada, diz respeito aos **primeiros 25% dos valores assumidos pela variável e é limitada pelo valor mínimo e pelo valor do 1º Quartil ou quartil inferior (Q1)**;
- a quarta parte (superior), também representada por uma linha pontilhada, diz respeito aos **últimos 25% dos valores assumidos pela variável e é limitada pelo valor do 3º Quartil ou quartil superior (Q3) e pelo valor máximo; no centro, representando os 50% valores centrais assumidos pela variável**,
- há uma caixa, dividida por uma linha contínua que representa a mediana dos valores assumidos pela variável, também chamada de 2º quartil (Q2).

A caixa, portanto, representa 50% de todos os valores observados, concentrados na tendência central dos valores, eliminando os 25% menores valores e 25% maiores valores ($75\% - 25\% = 50\%$). Valores muito dispareados que podem aparecer em alguma variável são considerados **outliers** e representados no gráfico por círculos

```
In [124]: boxplot(iris[1:4])
```

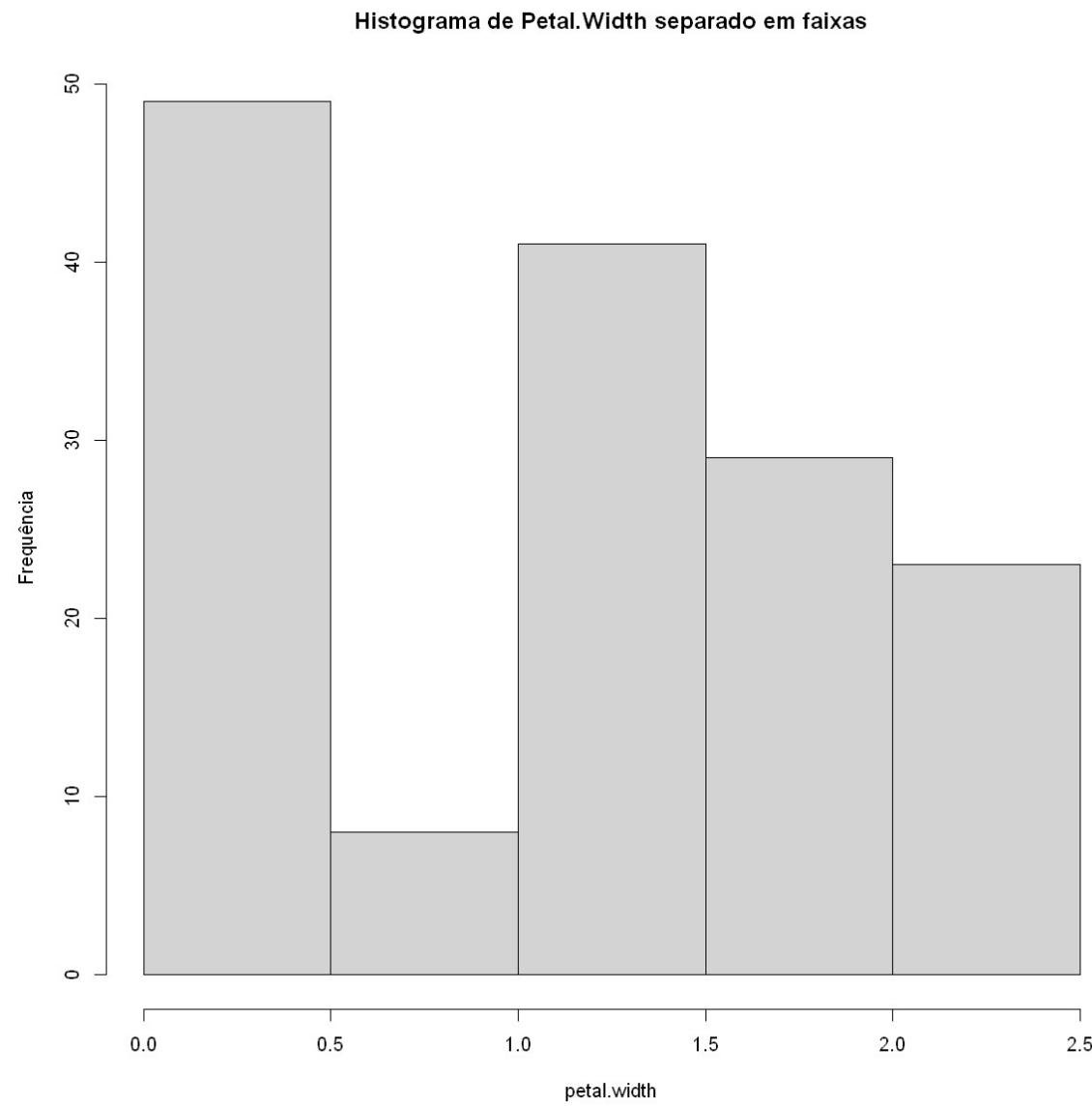


Histogramas

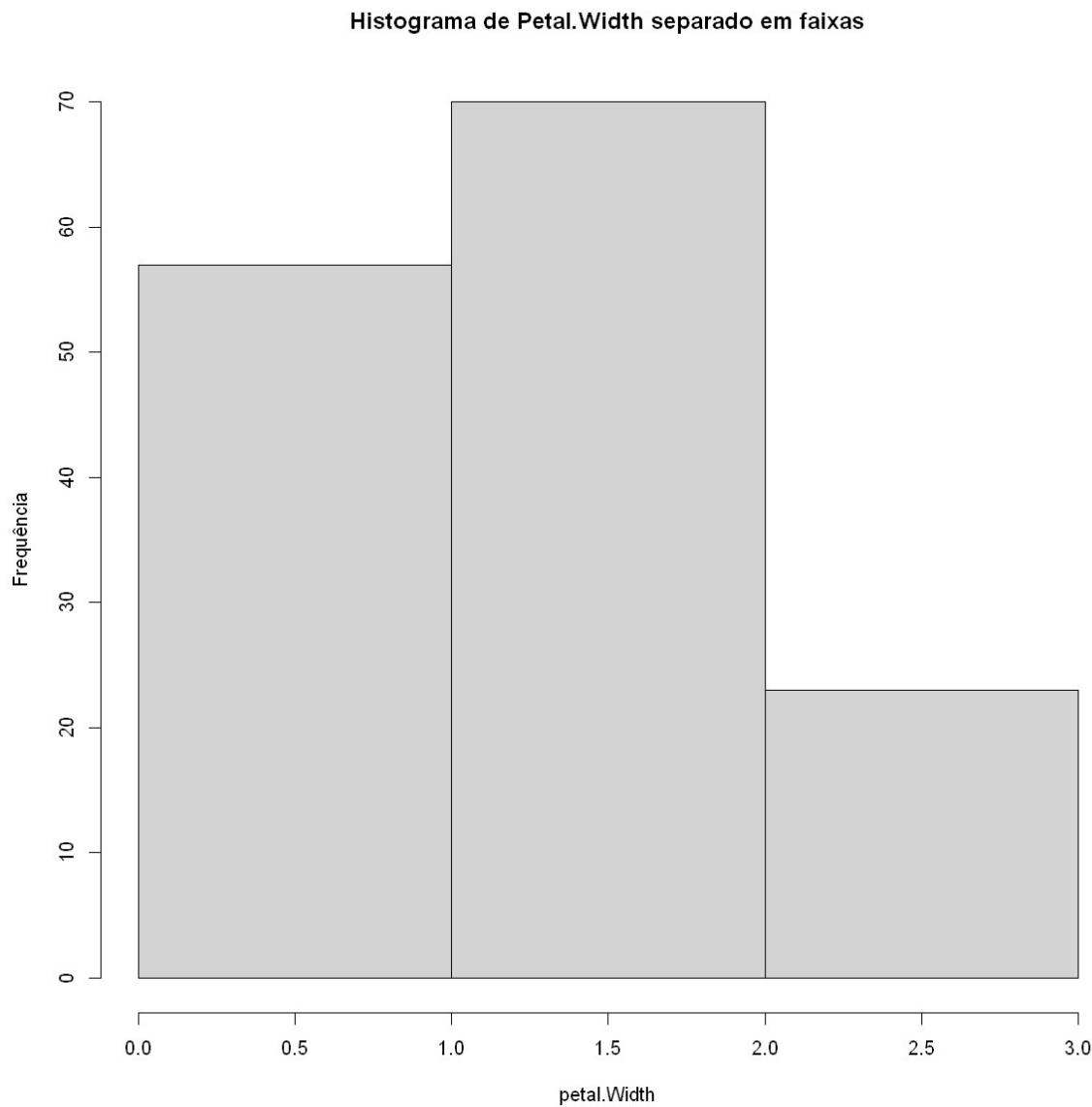
Um histograma , também conhecido como distribuição de frequências, pode ser apresentado em uma forma gráfica composta por colunas (ou barras), na qual as colunas dizem respeito à divisão em faixas (geralmente uniforme) dos valores de uma variável de um conjunto de dados. A função em *R* que gera um histograma é a `hist()` , cujos principais parâmetros estão definidos na Tabela abaixo.

| Parâmetros | Descrição |
|------------|---|
| X | Conjunto de valores a serem organizados no histograma, geralmente um vetor. |
| bin | Quantidade de faixas nas quais se deseja dividir os valores de x. |

```
In [129]: hist(t(iris['petal.width']),main='Histograma de Petal.Width separado em faixas',xla  
b='petal.width',  
5,ylab='Frequência')
```




```
In [127]: hist(t(iris['petal.width']),main='Histograma de Petal.Width separado em 3 faixas',x  
lab='petal.Width',  
3,ylab='Frequência')
```



Referencia para consulta sobre ANÁLISE GRÁFICA COM USO DO R do Prof Adilson dos Anjos Departamento de Estatística - UFPR - 2006 [link \(https://docs.ufpr.br/~aanjos/CE231/web/apostila.html#x1-120001.2.1\)](https://docs.ufpr.br/~aanjos/CE231/web/apostila.html#x1-120001.2.1)

In []: