- Tutorial
- Exercícios
- Apostila

# 5a. Criação e Edição de Gráficos no R


Nesse tutorial iremos apresentar os conceitos para a produção de gráficos no R, baseados nos pacotes da distribuição base do R: `grDevices` e `graphics`.

## Gráficos na Tela

Existem vários dispositivos gráficos no R que estão relacionados a dois grupos principais: os dispositivos de tela e os de arquivos. Nos dispositivos base do `grDevices` temos os dispositivos de tela `windows`, `X11` e `quartz` que produzem janelas gráficas, funções que dependem um pouco dos sistemas operacionais que estão sendo usados (i.e. Windows, Linux ou MacOS). Assim, para abrir um dispositivo de tela temos as funções:

```
X11()
windows()
quartz()
```

O `X11` ou `x11` é uma função mais geral e funciona na maioria dos sistemas operacionais. Os outros são mais específicos e deve retornar uma mensagem de erro quando usada em outros sistemas operacionais. Para informações sobre o `quartz` no macOSX veja a documentação oficial aqui.

**O sistema do pacote** graphics

Video

# Funções de Alto Nível

As funções gráficas de alto nível são aquelas que iniciam um dispositivo gráfico de tela e arranjam os elementos essenciais do gráfico no dispositivo. A principal função é a `plot`, mas já usamos outras no tutorial anterior, como o `hist`, `boxplot` e `barplot`.

### Criando Dados

Vamos criar um conjunto de dados fictícios para apresentar em gráficos e conhecer as principais funções e conceitos associados a elaboração deles pelo `graphics`. Para tornar o exemplo mais conectado a uma situação real vamos relacioná-los a um trabalho de ecologia de paisagem onde o objetivo é entender a influência do tamanho e conectividade dos fragmentos florestais com a riqueza de espécies de aves:

- `riqueza`: variável resposta, número de espécies de aves
- `capturas`: número de indivíduos capturados com o mesmo esforço amostral
- `area`: variável preditora relacionada à área do fragmento florestal em hectares
- `conectada`: variável preditora relacionada ao grau de conectividade do fragmento florestal

```
riqueza <- c(15, 18, 22, 24, 25, 30, 31, 34, 37, 39, 41, 45)
capturas <- c(33, 62, 75, 100, 150, 155, 167, 170, 171, 177, 178, 179)
area <- c(2, 4.5, 6, 10, 30, 34, 50, 56, 60, 77.5, 80, 85)
conectada <- factor(c("L", "M", "M", "L", "L", "H", "L", "H", "M", "M",
"H", "H"), levels = c("L", "M", "H"))
```

Podemos usar esses vetores diretamente para fazer os gráficos. Entretanto, não é o que

usualmente acontece. Quando fazemos a leitura de dados externos, os vetores das variáveis, normalmente, estão em um `dataframe`. Vamos criá-lo para tornar o exemplo ainda mais próximo da realidade:

```
frags <- data.frame(riq = riqueza, cap = capturas, ha = area, con =
conectada)
str(frags)
```

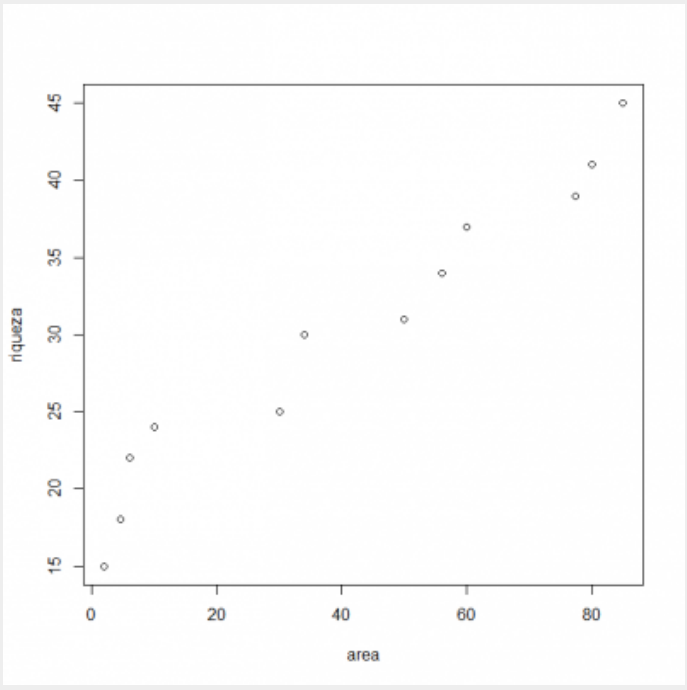
## Criando Gráficos

No pacote `graphics` há duas maneiras de se especificar as variáveis em gráficos, como vimos anteriormente também no tutorial de análise exploratória de dados:

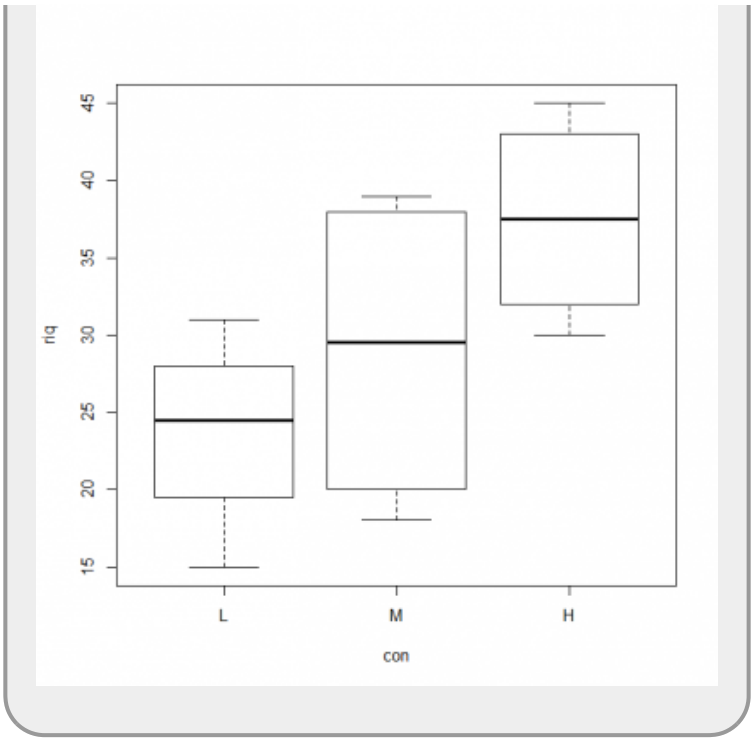
- Nome ou posição dos argumentos: `fungraph(x = a , y = b)`
- Fórmula estatística: `fungraph(b ~ a, data = dados)`

Abaixo podemos ver os gráficos associados a cada uma das variáveis preditoras com a saída padrão das funções `plot` e `boxplot`:

```
plot(x = area, y = riqueza)
```




```
boxplot(riq ~ con, data = frags)
```




Note que a primeira linha de código acima utilizou os objetos de vetores (`area` e `riqueza`), enquanto a segunda utiliza as variáveis que estavam no objeto `frags`, usando os nomes das colunas deste objeto. Além disso, a segunda linha de código utiliza a fórmula estatística, com o símbolo ~. Vamos usar esse último formato a partir de agora.

# Editando Gráficos

## Parâmetros Locais e Globais

A lógica dos gráficos no R é ajustá-los através de parâmetros que são estabelecidos antes ou durante a produção do gráfico. Os parâmetros globais devem ser modificados antes do gráfico ser iniciado pela função de alto nível. Os parâmetros locais são aqueles que podem ser modificados como argumentos dentro da função que produz o gráfico. Os parâmetros locais, muitas vezes, também podem ser modificados antes de iniciar o gráfico. Vamos ver como isso funciona! Primeiro vamos modificar localmente os parâmetros do tipo de símbolo grafado `pch` e seu tamanho `cex`:

```
plot(riq ~ ha, pch = 19, cex = 1.5, data = frags)
plot(riq ~ ha, data = frags)
```

Essa modificação é local e não fica registrada no dispositivo. Por outro lado, as modificações globais ficam armazenadas no dispositivo ativo e são modificadas antes da produção do gráfico, utilizando-se a função `par`:

```
par(pch = 19, cex = 1.5)
plot(riq ~ ha, data = frags)
plot(riq ~ ha, data = frags)
```

O `par` é uma das funções mais utilizadas, ao lado de `plot` para a produção de gráficos. Não tenho receio em afirmar que é a função que eu, pessoalmente (— *Alexandre Adalardo de Oliveira 2020/09/15 18:04*), mais consultei a documentação durante minha trajetória no R!

Vamos abrir a documentação e fazer a leitura para nos familiarizarmos com tudo que está disponível para modificação no dispositivo gráfico. Ao final, irá entender porque tive que consultar o `par` **"um par de vezes!"**

```
help(par)
```

Podemos acessar os valores dos parâmetros do dispositivo ativo, chamando a função `par` sem nenhum argumento:

```
par( )
par( )$pch
par( )$cex
```

Note que o `pch` e o `cex` permanecem com os valores que foram modificados. Neste caso, para voltar ao padrão é necessário retornar o parâmetro para o valor inicial ou fechar o dispositivo, e abrir um novo.

```
par(pch = 1, cex = 1)
plot(riq ~ ha, data = frags)
```

São muitos parâmetros e lembrar todos os que foram modificados e seus valores padrão é uma tarefa complicada. Por essa razão, existe um procedimento particular à função `par` que é atribuir os valores padrão a um objeto, no momento em que os parâmetros são modificados. É possível, então, usar este objeto para retornar os valores ao padrão:

```
oldpar <- par(pch = 19, cex = 1.5)
oldpar
plot(riq ~ ha, data = frags)
par(oldpar)
plot(riq ~ ha, data = frags)
```

Não são todos os parâmetros gráficos que podem ser modificados ou que se comportam da mesma maneira tanto localmente quanto globalmente. Por exemplo, o parâmetro `mfrow` é um parâmetro exclusivamente global que divide o dispositivo em painéis que podem ter diferentes gráficos. O `mfrow` recebe um vetor com dois valores inteiros, representando as divisões das linhas e colunas do dispositivo, respectivamente:

```
oldpar <- par(pch = 19, mfrow = c(1,2))
oldpar
plot(riq ~ ha, data = frags)
plot(riq ~ ha, data = frags, cex = 1.5)
par(oldpar)
```

Por outro lado, o parâmetro `cex` tem um comportamento diferente quando aplicado no `par` ou em `plot`:

```
plot(riq ~ ha, data = frags, cex = 2.5, pch = 16)
x11()
oldpar <- par(cex = 2.5)
plot(riq ~ ha, data = frags, pch = 16)
par(oldpar)
```

No código acima o `cex` no `par` utiliza o fator de incremento para todos os elementos do gráfico, enquanto que localmente o mesmo parâmetro no `plot` usa esse fator apenas para incrementar os símbolos associados às observações. Note também que abrimos uma nova janela com a função `x11`[1)] para compararmos os dois gráficos. Por padrão, o R sempre sobrescreve o gráfico na janela ativa e torna ativa a janela recém aberta.

## Parâmetros Vetorizados

Os parâmetros associados à representação dos dados no gráfico, em geral, são vetorizados. Isso significa que podem ser individualizados, tendo um valor para cada observação (linhas do `data.frame`). Vamos ver como isso acontece, utilizando o parâmetro `col` que define a coloração do símbolo associado a cada observação.

**CoRes**

O R tem vários métodos para atribuição de cores. Os mais comuns são valores inteiros e o nome, veja alguns exemplos:


Use o comando `colors()` para ver todos os nomes de cores.

Vamos usar então a representação de valores inteiros do quadro acima para definir cores para cada observação dos nossos dados:

```
plot(riq ~ ha, data = frags, cex = 1.5, pch = 19, col = 1:nrow(frags))
```

# Inserindo mais Informações em Gráficos

Uma outra lógica dos gráficos no R é que os elementos grafados não são apagados, mas podemos inserir novos elementos que serão sobrepostos aos que já existem. Para inserirmos elementos utilizamos as funções subordinadas às funções de alto nível que só operam se houver um dispositivo ativo e com um gráfico iniciado.

A seguir apresentamos alguns exemplos de funções para se inserir informações em gráficos.

## lines()

Função para inserir linhas retas ou curvas não-paramétricas utilizando alguma estimativa como `lowess`, `loess` e `gam`.

```
plot(cap ~ ha , data = frags)
lines(lowess(x = frags$ha, y = frags$cap))
```

## points()

Para inserir novos pontos no gráfico:

```
plot(riq ~ ha, data = frags, pch = 19, col = 1:nrow(frags))
points(riq ~ ha, data = frags, cex = 1.8)
```

## text()

Função utilizada para inserir caracteres dentro do gráfico. O texto pode ser letras, símbolos, palavras ou até mesmo uma frase. Lembre-se sempre que essas funções, em geral, são vetorizadas e podem inserir vários elementos de uma única vez, por exemplo, identificando cada observação:

```
text(x = 70, y = 31, labels = "<- olha esse dados!")
text(x = frags$ha, y = frags$riq + 1.1, labels = LETTERS[1: nrow(frags)],
cex = 0.7)
```

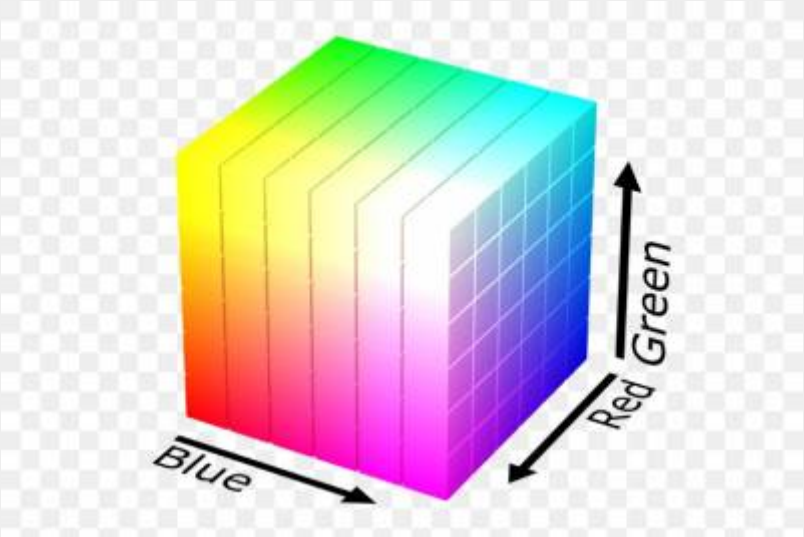
## arrows(), rect(), polygon(), segments() ...

São muitas funções acessórias que inserem novos elementos nos gráficos, não faz sentido passar por todas aqui, vamos usar mais uma para apresentar uma outra forma de usar cores no R:

```
rect(xleft = 25, ybottom = 22, xright = 41, ytop = 32, col = rgb(red =
1,green= 0, blue = 0, alpha = 0.1))
```

**coRes**

Um outro método para indicar cores no R é o RGB, o sistema de combinação de vermelho, verde e azul. Uma das vantagem desse sistema, além de possibilitar uma infinidade de cores e tonalidades, é que ele permite a inclusão da transparência da cor através do argumento `alpha` da função `rgb`.




## Ajustando o Gráfico

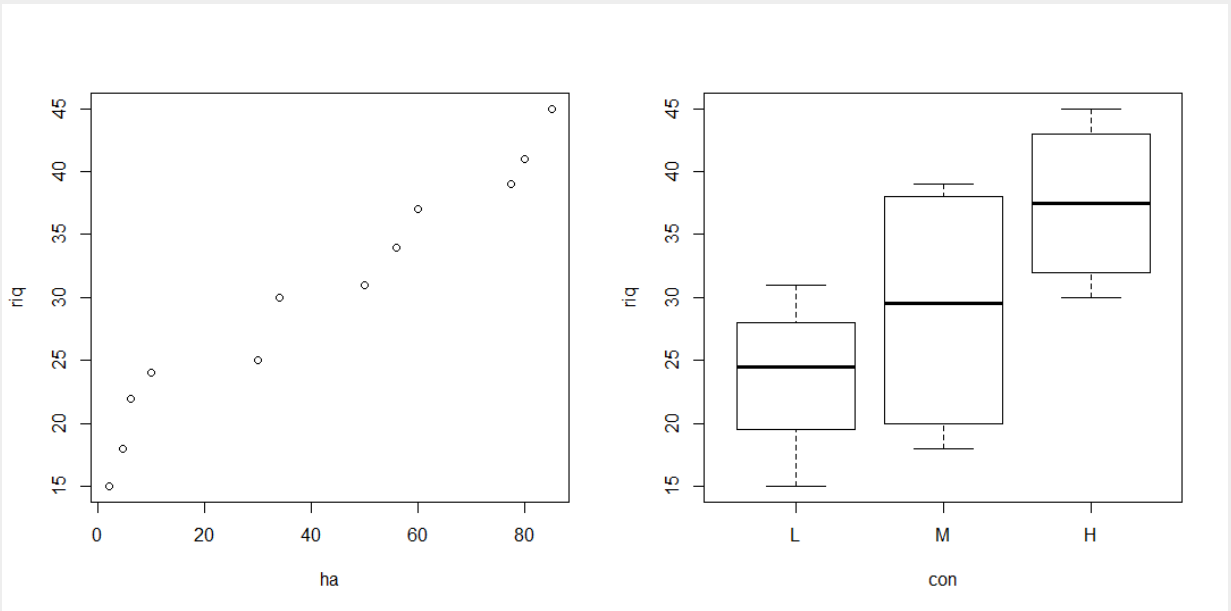
**Um exemplo**

Video

Agora que conhecemos os princípios da construção de gráficos utilizando o `graphics`, vamos ajustar

nosso gráfico base, primeiro fechando todos os dispositivos abertos e iniciando um novo com tamanho determinado e em seguida fazendo os gráficos base lado a lado:

```
graphics.off()
X11(width = 14, height = 7)
par(mfrow = c(1, 2))
plot(riq ~ ha, data = frags)
boxplot(riq ~ con, data = frags)
```




Vamos seguir ajustando os elementos dos gráficos. Primeiro, ajustando alguns parâmetros globais com o `par`:

| argumento | elemento | valor padrão | representação |
|---|---|---|---|
| `mar` | tamanho das margens | `c(5, 4, 4, 2)` | numero de linhas [2] |
| `las` | legenda dos eixos | `0` | paralela ao eixo |
| `mgp` | elementos dos eixos | `c(3, 1, 0)` | distância em linhas [3] |
| `family` | família de fonte | `""` | fonte padrão do dispositivo |

```
graphics.off()
X11(width = 14, height = 7)
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1), family = "serif", las = 1, mgp =
c(2.5, 0.8, 0), cex = 1.2 )
plot(riq ~ ha, data = frags)
boxplot(riq ~ con, data = frags)
```

Agora vamos ajustar o primeiro gráfico utilizando algumas dos parâmetros locais na própria função `plot`:

- `xlab` : título do eixo x
- `ylab` : título do eixo y
- `cex.lab`: aumento dos caracteres no título dos eixos

- `cex.axis`: aumento dos caracteres na escala dos eixos
- `bty`: tipo de caixa ao redor do gráfico
- `ylim`: limites da escala do eixo y

```
graphics.off()
X11(width = 14, height = 7)
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1), family = "serif",
                  las = 1, mgp = c(2.5, 0.8, 0), cex = 1.2 )
plot(riq ~ ha, data = frags, xlab = "Área (ha)", ylab = "Riqueza", cex =
1.5,
                  cex.lab = 1.5, cex.axis = 1.2, bty = "l", ylim = c(12,
45),
                  pch = c(15, 16, 17)[frags$con],
                  col = c("red", "cornflowerblue",
"aquamarine4")[frags$con])
boxplot(riq ~ con, data = frags)
```

Note a indexação que foi feita no `pch` e no `col`. Para fornecer o mesmo número de símbolo e nome de cor para as observação de um mesmo nível de fator, fizemos a indexação desses parâmetros gráficos pelo fator `conectada`. Para entender, primeiro vamos investigar o que foi produzido nas indexações:

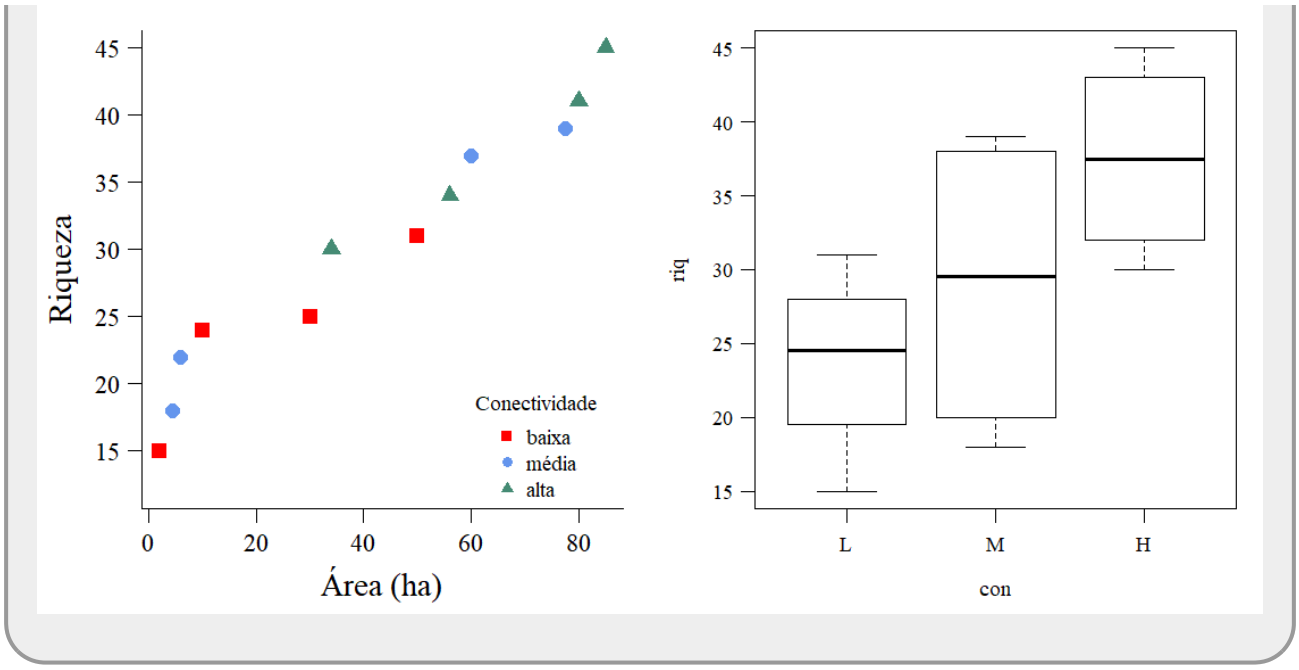
```
c(15, 16, 17)[frags$con]
c("red", "cornflowerblue", "aquamarine4")[frags$con]
```

Como `factor` é armazenado na forma de números inteiros relativos à ordem dos níveis, podemos muito facilmente substituir os valores dos níveis, mantendo a ordem que aparecem nos dados, simplesmente indexando os novos valores pelo fator.

Agora vamos inserir uma legenda para as cores e símbolos:

```
graphics.off()
X11(width = 14, height = 7)
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1), family = "serif",
    las = 1, mgp = c(2.5, 0.8, 0), cex = 1.2 )
plot(riq ~ ha, data = frags, xlab = "Área (ha)", ylab = "Riqueza", cex =
1.5,
     cex.lab = 1.5, cex.axis = 1.2, bty = "l", ylim = c(12, 45),
     pch = c(15, 16, 17)[frags$con],
     col = c("red", "cornflowerblue", "aquamarine4")[frags$con])
legend(x = 60, y = 20, legend = c("baixa", "média", "alta"), title =
"Conectividade",
       col = c("red", "cornflowerblue", "aquamarine4"), pch = c(15, 16, 17),
bty = "n")

boxplot(riq ~ con, data = frags)
```



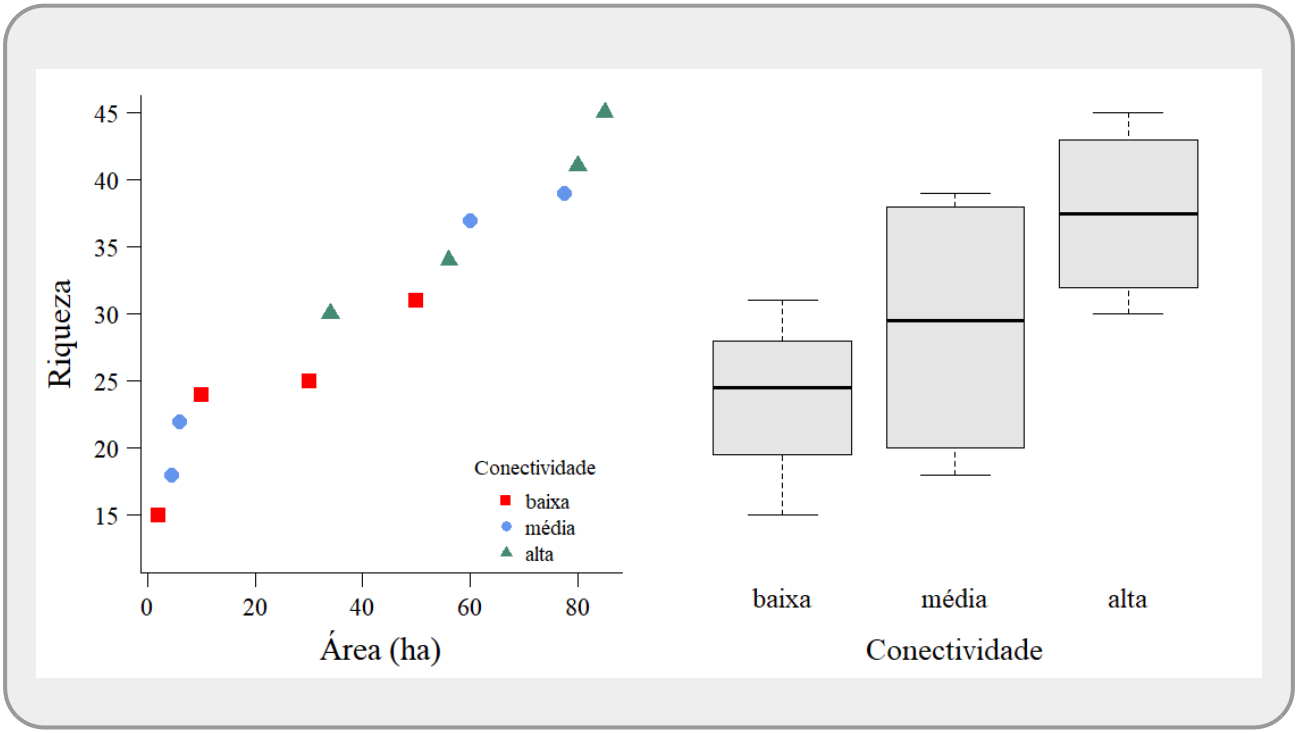

O procedimento normalmente é esse. Vamos ajustando os parâmetros e avaliando o resultado até conseguir o resultado almejado. Satisfeito com o primeiro painel, podemos passar para o segundo. Da mesma forma que o primeiro painel, os parâmetros gráficos globais que não podem ser modificados localmente, devem ser modificados antes de executar a função de alto nível.

```
graphics.off()
X11(width = 14, height = 7)
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1), family = "serif", las = 1, mgp =
c(2.5, 0.8, 0), cex = 1.2 )
plot(riq ~ ha, data = frags, xlab = "Área (ha)", ylab = "Riqueza", cex =
1.5, cex.lab = 1.5, cex.axis = 1.2, bty = "l", ylim = c(12, 45), pch = c(15,
16, 17)[frags$con], col = c("red", "cornflowerblue",
"aquamarine4")[frags$con])
legend(x = 60, y = 20, legend = c("baixa", "média", "alta"), title =
"Conectividade", col = c("red", "cornflowerblue", "aquamarine4"), pch =
c(15, 16, 17), bty = "n")
par(bty = "n", mar = c(4, 1, 1,1))
boxplot(riq ~ con, data = frags, ylim = c(12, 45), ann = FALSE, xaxt = "n",
yaxt = "n", col = "gray90")
```

No código do diagrama de caixas acima, para ter mais controle sobre título e legendas dos eixos, solicitamos na função que estes não fossem grafados. Agora podemos incluir esses elementos utilizando a função `mtext` que inclui caracteres nas bordas do gráfico:

```
graphics.off()
X11(width = 14, height = 7)
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1), family = "serif", las = 1, mgp =
c(2.5, 0.8, 0), cex = 1.2 )
plot(riq ~ ha, data = frags, xlab = "Área (ha)", ylab = "Riqueza", cex =
1.5, cex.lab = 1.5, cex.axis = 1.2, bty = "l", ylim = c(12, 45), pch = c(15,
16, 17)[frags$con], col = c("red", "cornflowerblue",
"aquamarine4")[frags$con])
legend(x = 60, y = 20, legend = c("baixa", "média", "alta"), title =
```

```
"Conectividade", col = c("red", "cornflowerblue", "aquamarine4"), pch =
c(15, 16, 17), bty = "n")
par(bty = "n", mar = c(4, 1, 1,1))
boxplot(riq ~ con, data = frags, ylim = c(12, 45), ann = FALSE, xaxt = "n",
yaxt = "n", col = "gray90")
mtext(text = c("baixa", "média", "alta"), side = 1, line = 0.5, at = c(1, 2,
3), cex = 1.5)
mtext("Conectividade", side = 1, line = 2.5, cex = 1.7)
```




Estando satisfeito com o resultado, podemos salvar o gráfico que está no dispositivo de tela ativo em nosso diretório, utilizando a função `savePlot`:

```
savePlot(filename = "graficoFrags.png", type = "png")
```

## Dispositivos de Arquivos

A função `savePlot` não permite a manipulação da qualidade e definição da imagem no arquivo que é construído. Para maior controle na produção do arquivo é preciso utilizar os dispositivos específicos para cada formato de arquivo. Os mais comuns são: `jpeg`, `pdf`, `png` e `tiff`. Para utilizá-los é preciso abrir o dispositivo, chamando a respectiva função com os parâmetros que serão utilizados na construção do arquivo. A partir desse momento podemos construir o gráfico da mesma forma que fazemos no dispositivo de tela. Para finalizar e criar o arquivo é necessário fechar o dispositivo com a função `dev.off`. Vamos salvar nosso gráfico com o dispositivo `jpeg` de alta qualidade e com 960 pixels de largura no arquivo `graficoFrags.jpeg`.

```
graphics.off()
jpeg(filename = "graficoFrags.jpg", width = 960, height = 480, units = "px",
pointsize = 12, quality = 100, bg = "white", res = NA)
par(mfrow = c(1, 2), mar = c(4, 4, 1, 1), family = "serif", las = 1, mgp =
```

```
c(2.5, 0.8, 0), cex = 1.2 )
plot(riq ~ ha, data = frags, xlab = "Área (ha)", ylab = "Riqueza", cex =
1.5, cex.lab = 1.5, cex.axis = 1.2, bty = "l", ylim = c(12, 45), pch = c(15,
16, 17)[frags$con], col = c("red", "cornflowerblue",
"aquamarine4")[frags$con])
legend(x = 60, y = 20, legend = c("baixa", "média", "alta"), title =
"Conectividade", col = c("red", "cornflowerblue", "aquamarine4"), pch =
c(15, 16, 17), bty = "n")
par(bty = "n", mar = c(4, 1, 1,1))
boxplot(riq ~ con, data = frags, ylim = c(12, 45), ann = FALSE, xaxt = "n",
yaxt = "n", col = "gray90")
mtext(text = c("baixa", "média", "alta"), side = 1, line = 0.5, at = c(1, 2,
3), cex = 1.5)
mtext("Conectividade", side = 1, line = 2.5, cex = 1.7)
dev.off()
```

**Gerenciando dispositivos**

Os dispositivos abertos são definidos por uma numeração sequencial e o seu tipo. Para gerenciar os dispositivos abertos utilizamos as funções da família `dev.`. Além do `dev.off`, as mais importantes são: `dev.list`, `dev.cur` e `dev.set`. O primeiro retorna o tipo e o número do dispositivo ativo, o segundo lista os dispositivos abertos enquanto o último define o dispositivo ativo. Caso necessite transitar entre dispositivos enquanto está editando é possível.

# Próximos passos

Para mais informações sobre a edição de gráficos siga para o capítulo da apostila 5a. Criação e Edição de Gráficos no R. Além disso, o tutorial 5b. Gráficos II: um procedimento apresenta um técnica poderosa para a construção de gráficos não usuais.

1) caso esteja usando um Mac, pode ter problemas para abrir a janela, nesse caso use `quartz()`

2) a ordem dos lados de um gráfico sempre obedece o sentido horário partindo do eixo x

3) entre o título, legenda e linha do eixo