- Tutorial
- Exercícios
- Apostila
- Apostila-Avançado

5a. Criação e Edição de Gráficos no R

Fazendo Gráficos no R

Aqui você irá aprender como fazer gráficos para publicação. Nesta aula, iremos passar apenas pelos gráficos mais simples como **gráficos de dispersão, de barras e box-plot**, pois estes serão os gráficos usados pela grande maioria dos alunos durante o curso de pós-graduação. Porém, lembre-se que no R é possível construir uma variedade incrível de gráficos e figuras. Para mais exemplos basta entrar no R Graph Gallery e ver as possibilidades.

Custo Benefício de Fazer Gráficos no R

Nesta apostila você aprenderá a editar os gráficos e adequá-los para dissertações, teses ou revistas científicas. Editar gráficos no R não é fácil, demora tempo (pode demorar horas para fazer apenas uma figura) e é muitas vezes frustrante, pois cada passo requer uma série de ajustes. Porém, o R permite mudar quase todos os parâmetros dentro de um gráfico, uma liberdade que (quase) nenhum outro pacote estatístico possui. E lembre-se, bons gráficos dizem mais que apenas o conjunto de dados a ser apresentado. Bons gráficos mostram vários resultados em um pequeno espaço de papel, são facilmente interpretáveis e podem aumentar suas chances de ter trabalhos aceitos em boas revistas científicas. Por isso, é muito importante investir bastante tempo em fazer figuras bonitas e bem explicativas.

Criando Gráficos

Fazer gráficos rapidamente no R é fácil. Basta dizer qual tipo de gráfico se deseja e quais são as variáveis.

Há duas maneiras de se especificar as variáveis

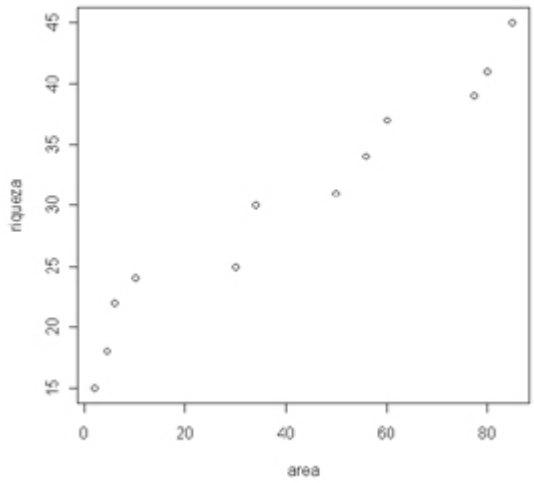
Cartesiana – plot(x,y)

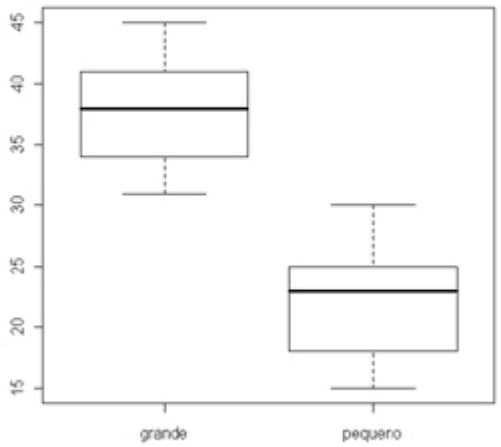
Formula – plot(y~x)

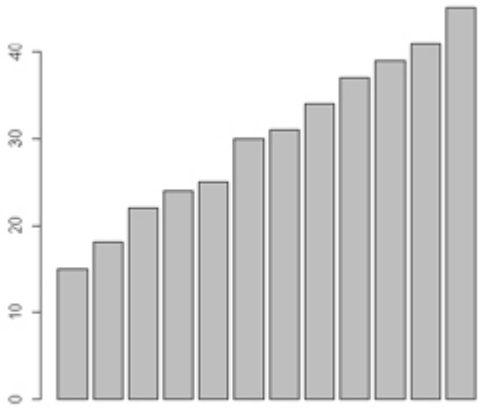
Ambas as formas são corretas, mas como a grande maioria das análises feitas são no formato y~x, em vez de x,y, acaba ficando mais fácil usar y~x.

```
riqueza <- c(15,18,22,24,25,30,31,34,37,39,41,45)
area <- c(2,4.5,6,10,30,34,50,56,60,77.5,80,85)
area.cate <- rep(c("pequeno", "grande"), each=6)

plot(riqueza~area)
plot(area,riqueza) # o mesmo que o anterior
boxplot(riqueza~area.cate)
barplot(riqueza)
```







As figuras padrão (`default`) que o R produz não são publicáveis, mas trazem toda a informação que foi usada para gerar o gráfico e podem perfeitamente ser usadas para uma interpretação inicial dos resultados. O `plot` ou scatterplot é um gráfico de dispersão, sendo que cada ponto no `plot` representa uma das réplicas (e.g. 12 réplicas, 12 pontos). Na sua forma mais simples, as legendas dos eixos vêm com o nome das variáveis usadas para criar o plot.

Quando as variáveis são categóricas, o gráfico padrão que o R produz é o boxplot ou “box and whiskers plot” (chamado em português de desenho esquemático, desenho da caixa, ou desenho de caixa e bigode). No boxplot, a linha grossa do meio representa a mediana, a caixa representa o 1° e 3°quartil, e os “bigodes” podem representar ou os valores máximos e mínimos, ou 1.5 vezes o valor dos quartis (aproximadamente 2 desvios padrões); é desenhado o que for menor. Às vezes, alguns pontos são desenhados individualmente além dos bigodes, estes são os “outliers”, que podem ser

suprimidos com o argumento `outline=F`.

O barplot, ou gráfico de barras, mostra cada ponto da variável especificada como uma barra. Na sua forma mais simples, são apresentados apenas os valores brutos e não há informação alguma quanto à dispersão dos dados. No barplot nenhum dos eixos vem com legendas (aliás, o eixo x também não é desenhado).

Exercício 1 – Fazendo os Primeiros Gráficos

Construa "plot", boxplot e barplot usando as variáveis do conjunto de dados Conjunto de Dados: Dados de Biomassa de Árvores de Eucalyptus Saligna, para explorar relações entre:

```
dap e ht
ht e tronco
dap e classe
dap e talhao
dap
ht
```

Note: `barplot` só aceita uma variável

Editando Gráficos

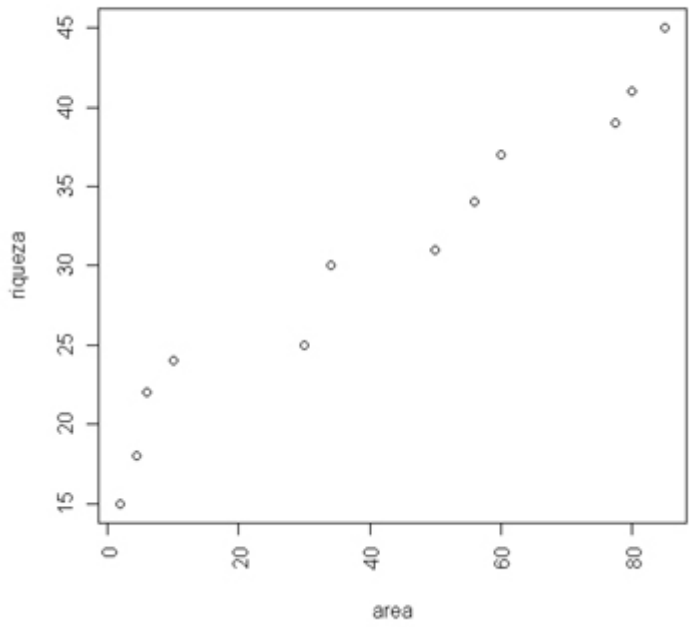
Aqui começa a parte mais complicada. Não porque é difícil mudar a forma como os gráficos são feitos, mas porque para chegar num resultado final adequado requer um processo iterativo. Em outras palavras, se o objetivo é mudar o tamanho da fonte, será necessário testar vários tamanhos até se atingir o "tamanho ideal" para incluir no manuscrito e/ou tese.

Existem duas maneiras de se mudar parâmetros no gráfico; uma é por dentro do gráfico, ou seja, dentro da função `plot, boxplot`, ou `barplot`, e a outra é pela função `par()`. Alguns argumentos só podem ser chamados **exclusivamente** por uma destas maneiras. Por exemplo `ylab` e `xlab` modificam o nome (label) dos eixos e só podem ser chamadas por dentro do gráfico, já outras funções só podem ser chamadas pelo `par()`, como por exemplo, `mar` que controla o tamanho das margens do gráfico e `mfrow` que controla quantos gráficos serão mostrados no mesmo dispositivo.

Para que as alterações controladas pelo `par()` possam surtir efeito, elas sempre devem vir antes do gráfico. Se um novo dispositivo gráfico não for aberto, todas as funções já controladas pelo `par()` continuarão valendo, mesmo que o gráfico mude.

Em geral, a informação que vem por último é a informação que o R vai tomar como verdadeira. Por exemplo, `las` controla a direção das legendas dos eixos (`las= 1`, legendas escritas sempre na horizontal, `las=3`, legendas sempre na vertical), sejam os números da escala ou o nome do eixo. Se o seguinte comando é dado:

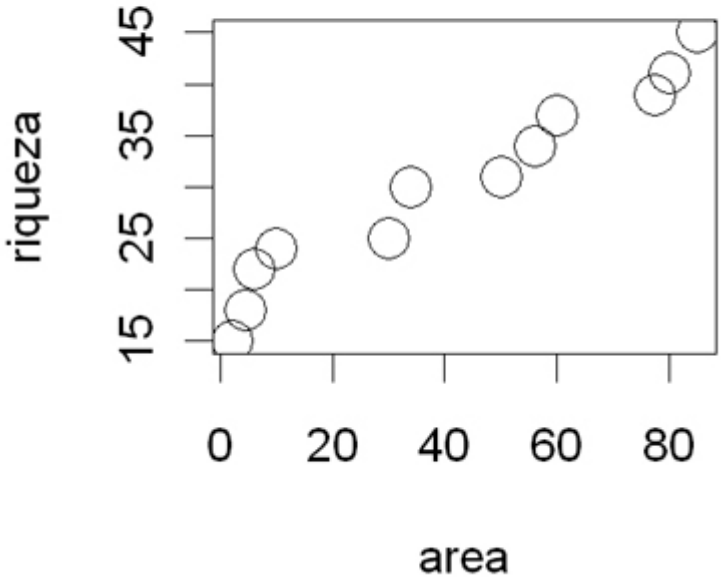
```
par(las=1)
plot(riqueza~area, las=3)
```



O resultado final será um gráfico com legendas na vertical. Isso a princípio pode parecer sem sentido, porém imagine um caso em que há vários gráficos no mesmo dispositivo gráfico e em todos os casos se deseja ter legendas horizontais, com exceção de um gráfico apenas em que um dos eixos será desenhado verticalmente. São em casos que nem este que se torna necessário poder dar informações "conflitantes" para o R.

Outro caso que é importante saber é a função `cex`. Em sua forma geral, ela se aplica ao tamanho de fonte das legendas, título, pontos, entre outros. Se o seguinte comando é dado:

```
par(cex=2)
plot(riqueza~area, cex=2)
```



O resultado final terá legendas com tamanho 2 (default=1) e pontos com tamanho 4. Isto ocorre pois `par(cex=2)` tem a função geral de aumentar todas as fontes e pontos, enquanto que no `plot(cex=2)` tem a função de aumentar só o pontos. E quando neste caso específico, em vez das informações entrarem em conflito, como no caso anterior, elas se multiplicam.

Exercício 2 – Aprendendo a Editar Gráficos

Entre no R e digite:

?plot

Agora, usando as variáveis:

```
riqueza <- c(15,18,22,24,25,30,31,34,37,39,41,45)
area <- c(2,4.5,6,10,30,34,50,56,60,77.5,80,85)
```

Mude:

O nome do eixo x para “Tamanho da Ilha (ha)”

O nome do eixo y para “Riqueza de Espécies”

O título do gráfico para “Aves das Ilhas Samoa”

Agora entre no:

?par

Usando o mesmo gráfico anterior, mude:

O tipo de ponto (numero de 0 a 25)

O tamanho dos pontos e legendas

A direção da escala do gráfico (para ficar tudo na horizontal)

O tipo de fonte das legendas (para ficar tudo como em Times New Roman – dica= “serif”)

Apesar das páginas de ajuda do R não serem muito amigáveis no começo, é preciso ter calma e aprender a procurar a informação desejada. A página do `par()` é uma das mais procuradas por todos que estão fazendo gráficos no R, e por isso é importante que se gaste um tempo para aprender qual tipo de informação ela fornece, onde está a informação, e como mudar os parâmetros do R.

par()

DICA

No começo, quando ainda não se conhece direito todas as funções do `par()` é aconselhável que se imprima a página de ajuda para que se possa visualizar todas os argumentos.

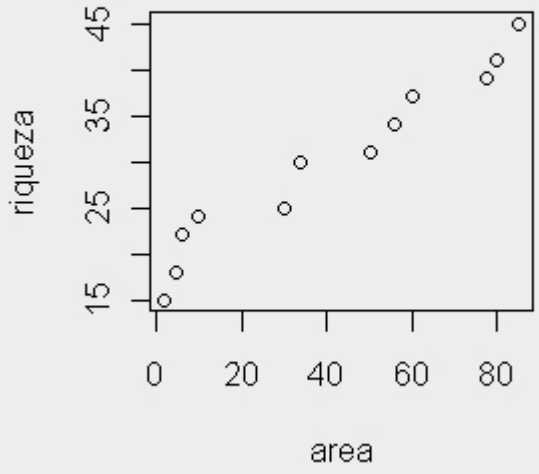
Existem dois argumentos do `par()` **muito importantes** e que são usadas quase 100% das vezes. Um, `par(mfrow=c())` controla “quantas figuras” serão desenhadas dentro de um mesmo dispositivo. O vetor contido dentro da função `mfrow=()` controla o numero de gráficos que serão desenhados no eixo x (1° número) e no eixo y (2° número).

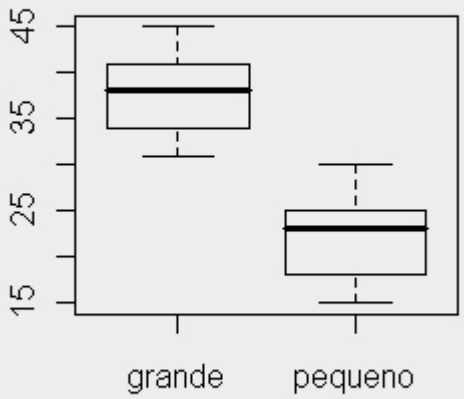
O outro, o `par(mar=c())` controla o “tamanho das margens” do gráfico e como a figura ficará disposta dentro do dispositivo. O vetor contido dentro da função `mar=()`, controla as posições das margens, sendo que o 1° numero controla a margem da parte de baixo do gráfico, o 2° controla a margem do lado esquerdo, o 3° numero controla a parte de cima e o 4° numero controla o tamanho da margem do lado direito do gráfico.

Dica: as figuras abaixo foram preenchidas de cinza para facilitar a visualização com o parâmetro par(bg=“gray93”)

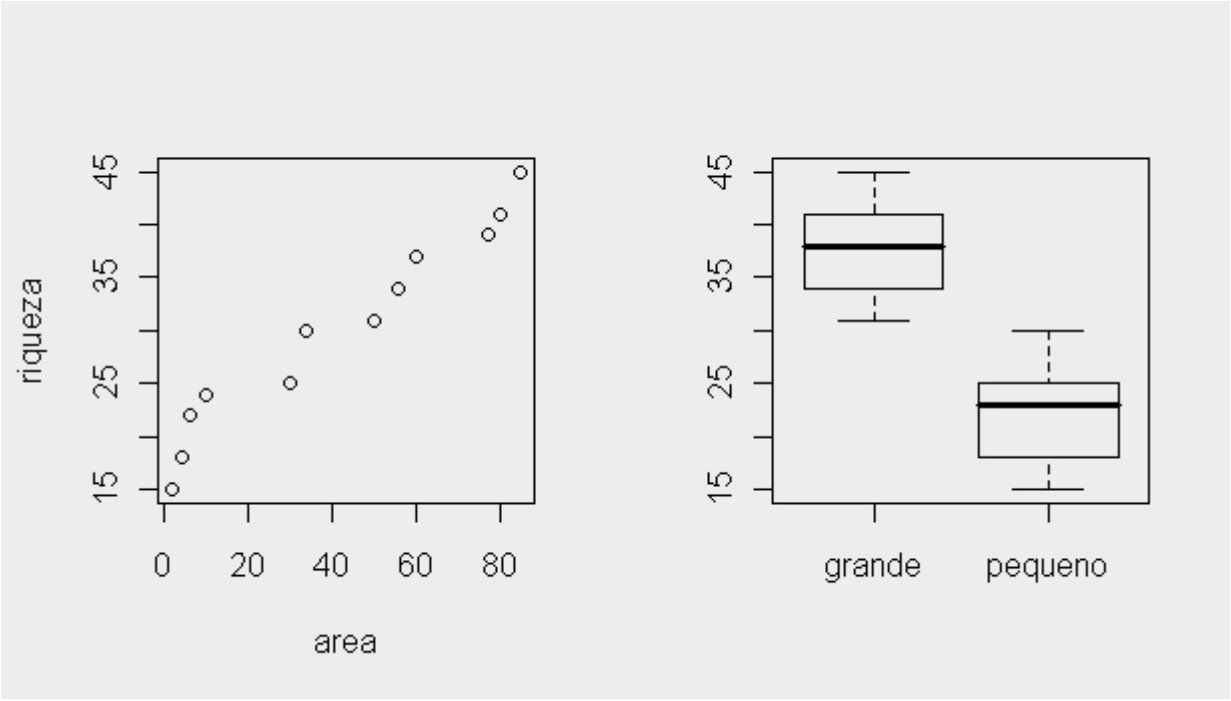
Exemplos de `par(mfrow=c())`

```
par(mfrow=c(2,1))
plot(riqueza~area)
boxplot(riqueza~area.cate)
```



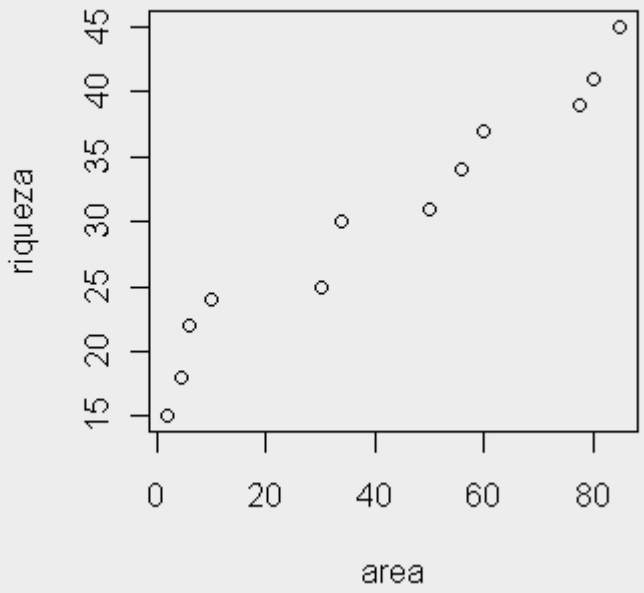


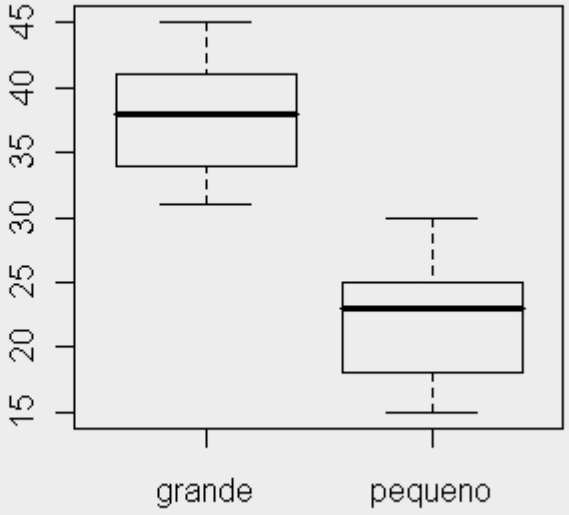

```
par(mfrow=c(1,2))
plot(riqueza~area)
boxplot(riqueza~area.cate)
```



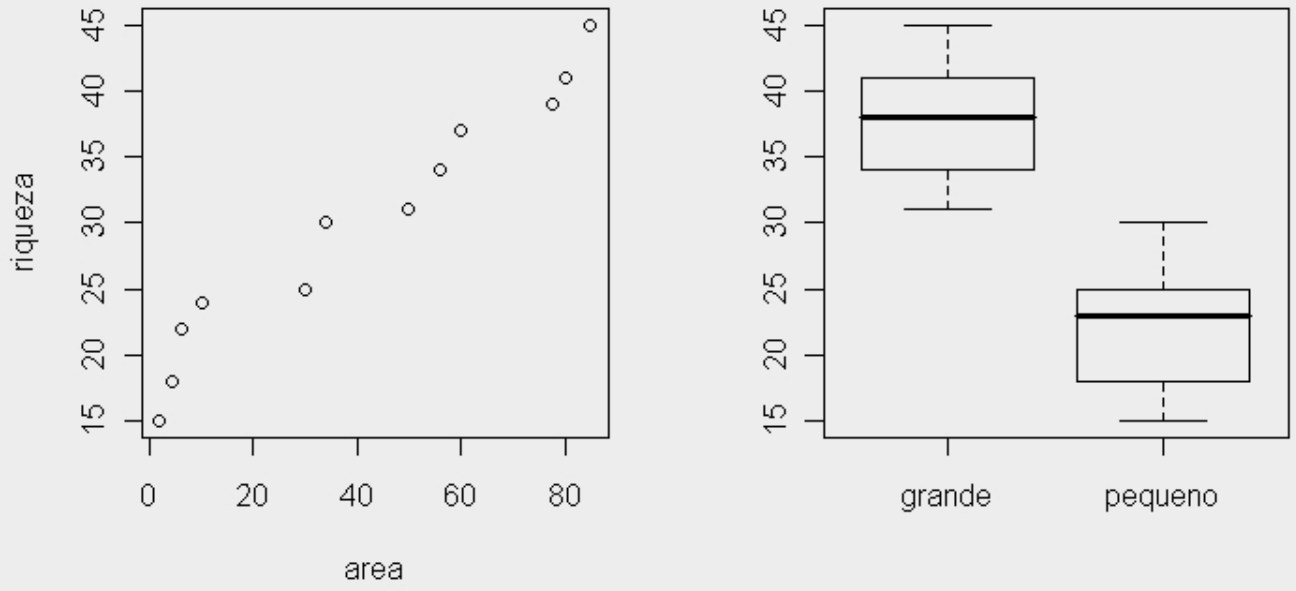
Exemplos de `par(mar=c())`

```
par(mfrow=c(2,1))
par(mar=c(4,14,2,6))
plot(riqueza~area)
boxplot(riqueza~area.cate)
```

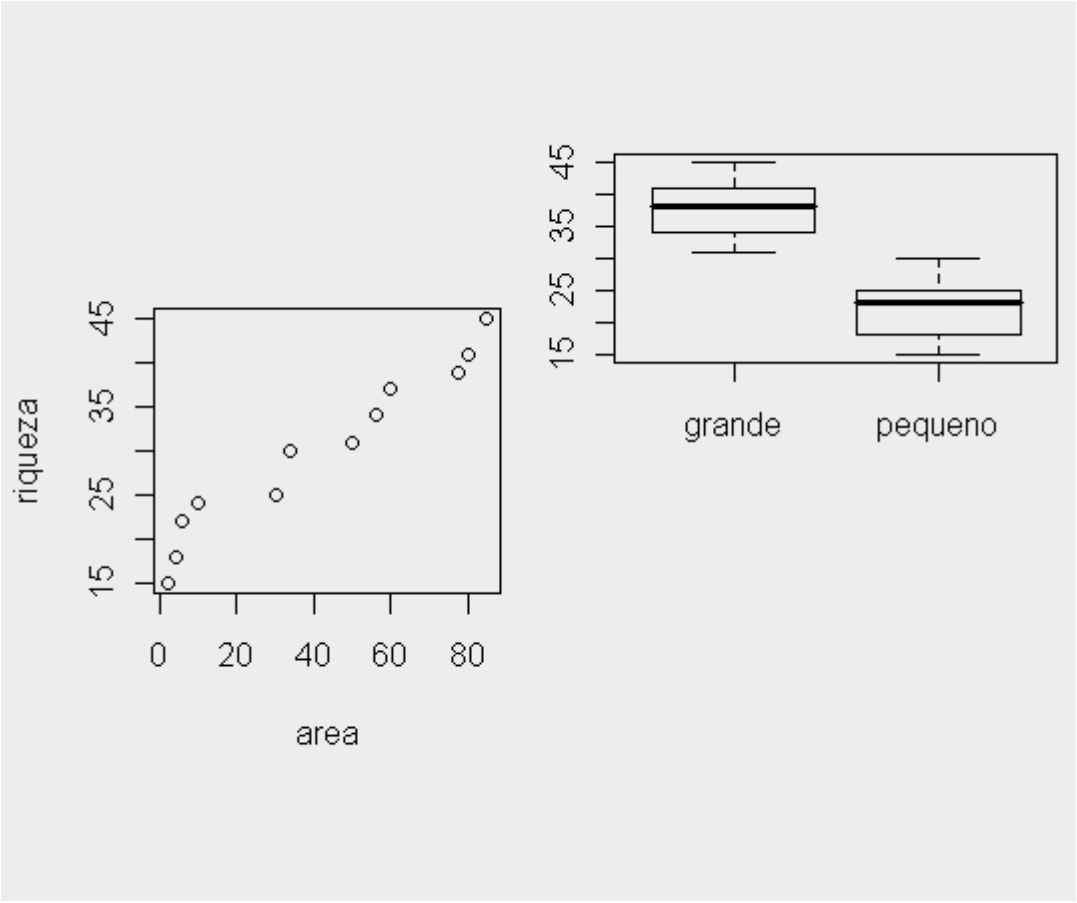





```
par(mfrow=c(1,2))
par(mar=c(14,4,8,2))
plot(riqueza~area)
boxplot(riqueza~area.cate)
```




```
par(mfrow=c(1,2))
par(mar=c(8,4,8,1))
plot(riqueza~area)
par(mar=c(14,2,4,0.5))
boxplot(riqueza~area.cate)
```



Diferenças Entre Tipos De Gráfico

Infelizmente, a forma como se muda argumentos do `plot(), boxplot()` e `barplot()` não é sempre a mesma, ou seja, comandos que funcionam perfeitamente para o `plot()` podem não produzir efeito algum no `boxplot()`, e vice-versa. Esta característica, de fato, atrapalha um pouco, mas assim que se acostuma fica mais fácil. Há duas dicas para resolver este problema: (i) tente sempre jogar os argumentos para o `par()` pois às vezes eles podem não funcionar se chamadas por dentro do `plot(), boxplot()`, etc, mas irão funcionar pelo `par()`; (ii) descubra o nome em inglês do parâmetro que se quer mudar (`label, tick, legend`) e jogue no Google “legend boxplot”. Com certeza, alguém já teve este mesmo problema, e entrando dentro da lista do R (as diversas que existem) ou em aulas disponibilizadas na internet, com certeza se acha uma solução.

Exercício 3 – Mudando diferentes Gráficos

Com as variáveis:

```
riqueza <- c(15,18,22,24,25,30,31,34,37,39,41,45)
area <- c(2,4.5,6,10,30,34,50,56,60,77.5,80,85)
area.cate <- rep(c(“pequeno”, “grande”), each=6)
```

Crie:

```
plot(riqueza~area)
```

E agora:

```
 plot(riqueza~area, bty="l", tcl=0.3)
```

Perecebeu o que mudou?

Agora tente:

```
boxplot(riqueza~area.cate, bty=”l”, tcl=0.3)
```

O que aconteceu?

E agora?

```
par(bty=”l”)
par(tcl=0.3)
boxplot(riqueza~area.cate)
```

Viu só?

Inserindo mais Informações em Gráficos

Existem diversas informações que se pode incluir em um gráfico. Pode-se colocar uma letra para mostrar que este é o painel “a” e ao lado é o painel ”b”; pode-se colocar asteriscos para mostrar quais relações são significativas; pode-se desenhar flechas, outros pontos, uma infinidade de coisas. Tudo isto pode ser feito, mas requer funções comandos separados daqueles já passados pelo par() e plot(), boxplot() ou barplot(). Dentre as várias funções existentes para se inserir informações em gráficos, existem sete que são bastante úteis. Use:

Exercício 4

Usando as variáveis:

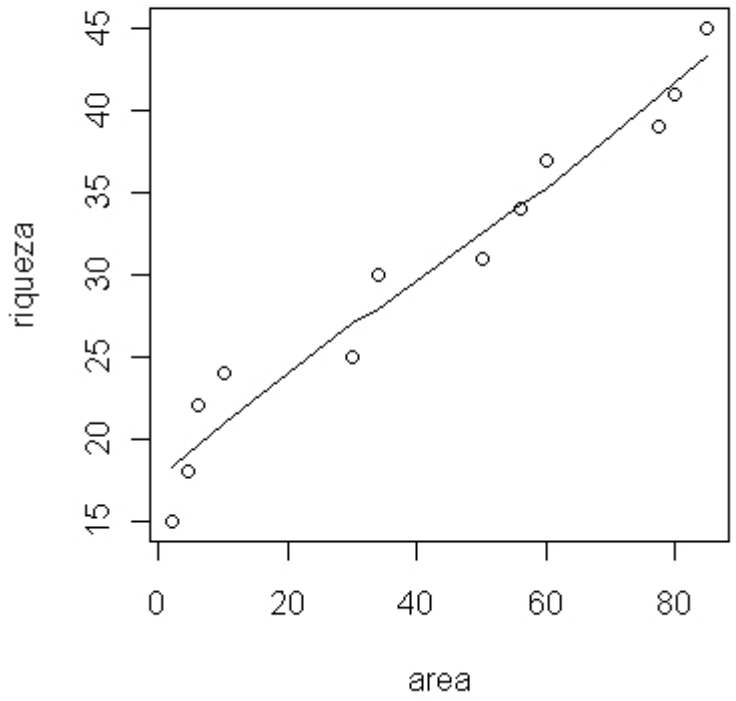
```
riqueza <-
c(15,18,22,24,25,30,31,34,37,39,41,45)
area <-
c(2,4.5,6,10,30,34,50,56,60,77.5,80,85)
abundancia <- rev(riqueza)
```

Crie gráficos inserindo os parâmetros abaixo.

lines()

Para inserir linhas retas ou curvas não-paramétricas (como `lowess`, `loess`, `gam`, etc).

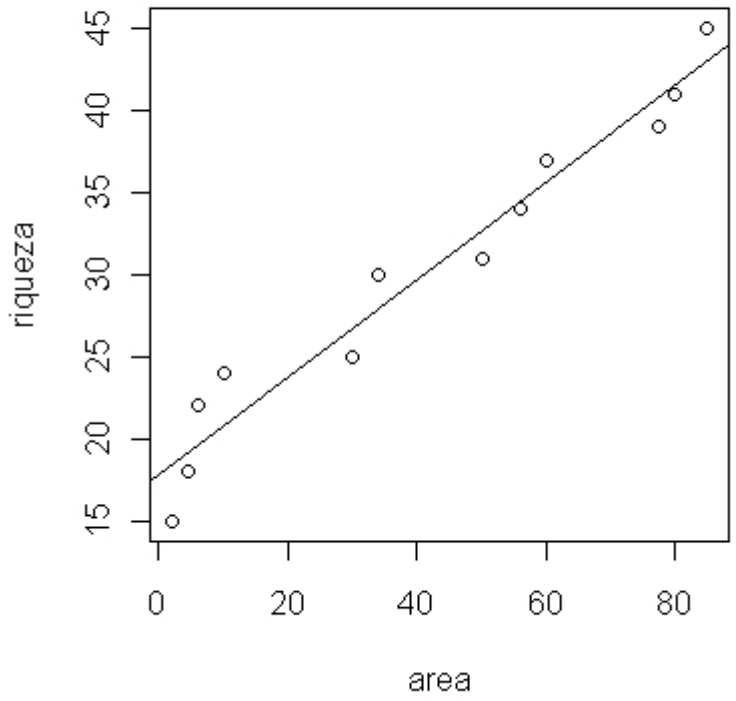
```
plot(riqueza~area)
lines(lowess(area, riqueza))
```



abline()

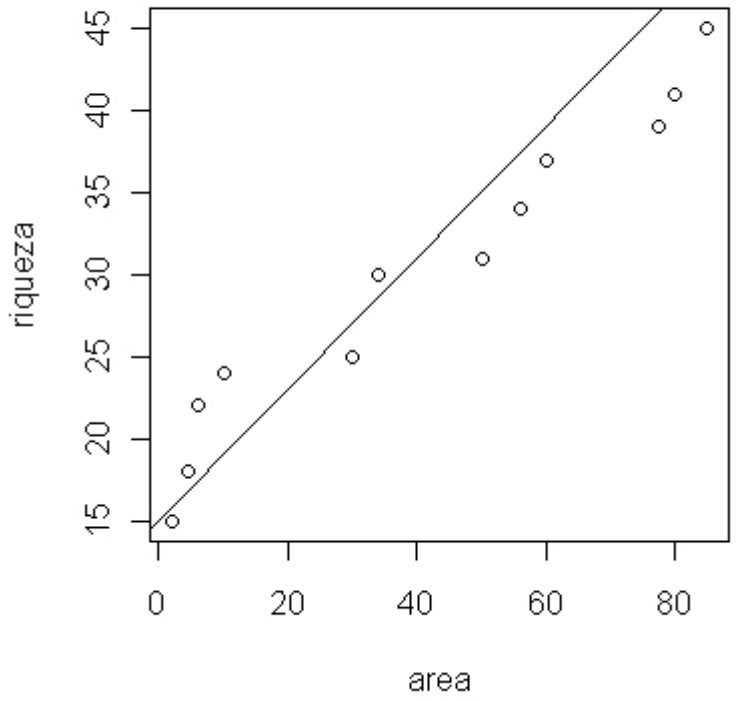
Para inserir linhas de tendência criadas a partir de um modelo linear. Para isso é primeiro necessário criar o modelo, para depois criar a linha.

```
model <-  lm(riqueza~area)
plot(riqueza~area)
abline(model)
```



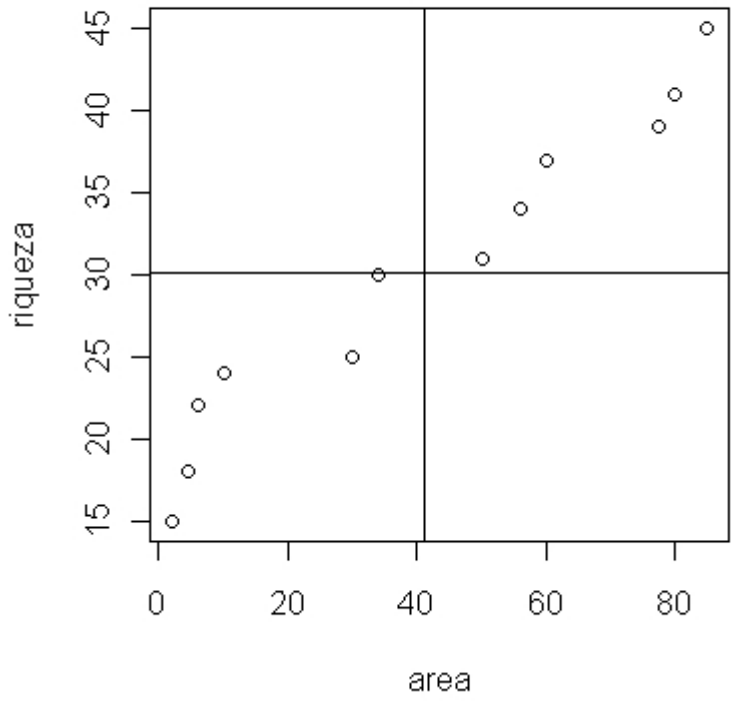
Com a função `abline` você pode também inserir uma linha reta com intercepto e inclinação definidos por você, com os dois primeiros argumentos:

```
plot(riqueza~area)
abline(15,0.4)
```



A função `abline` também serve para acrescentar linhas verticais e horizontais, com os argumentos v e h. No código abaixo traçamos estas linhas passando pelas médias das duas variáveis do diagrama de dispersão:

```
plot(riqueza~area)
abline(v=mean(area))
abline(h=mean(riqueza))
```

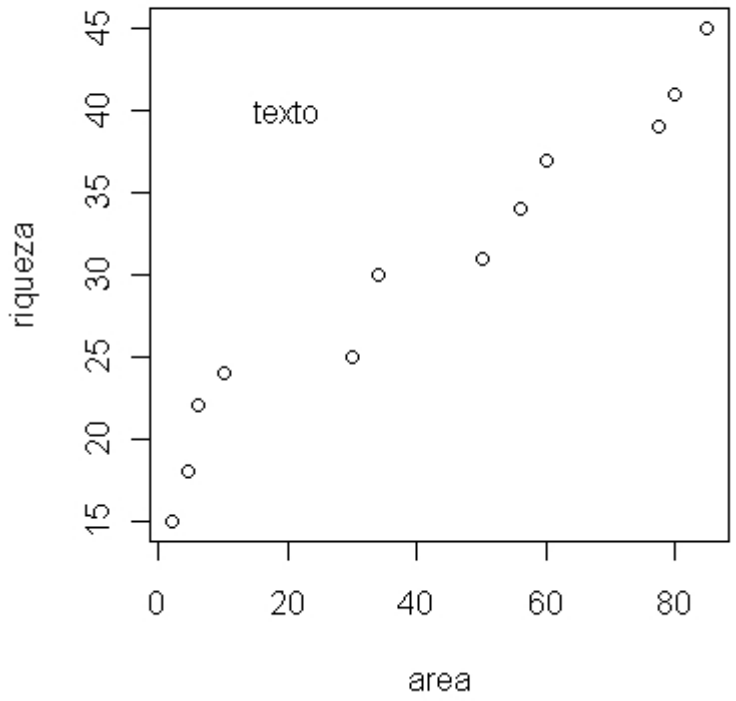


Você sabia? A reta da regressão linear simples sempre passa pelo ponto que é a interseção destas duas linhas.

text()

Para inserir texto dentro do gráfico. O texto pode ser uma letra, um símbolo (muito usado para mostrar diferenciar classes no gráfico), uma palavra ou até mesmo uma frase.

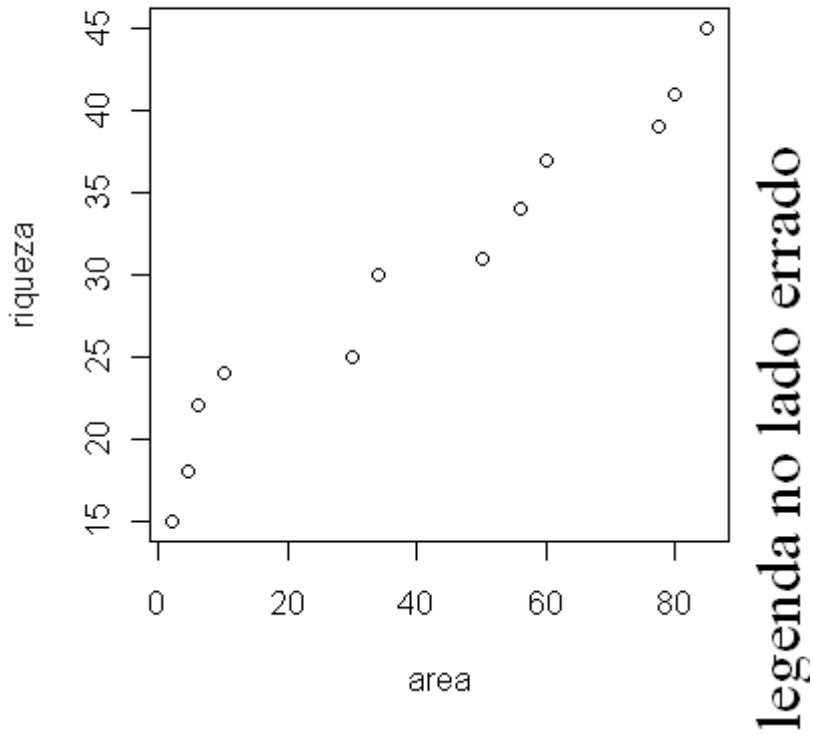
```
plot(riqueza~area)
text(20,40,"texto")
```



mtext()

Este comando acrescenta texto nas margens do gráfico ou da janela gráfica. Seu uso mais freuquente é inserir legendas dos eixos. Apesar de ser possível controlar as legendas por dentro das funções `plot`, `boxplot` e `barplot`, o numero de parâmetros que se pode mudar é limitado. Quando se deseja um controle mais fino dos parâmetros, como posição, alinhamento, cor, tamanho da fonte, etc, é necessário usar `mtext()`.

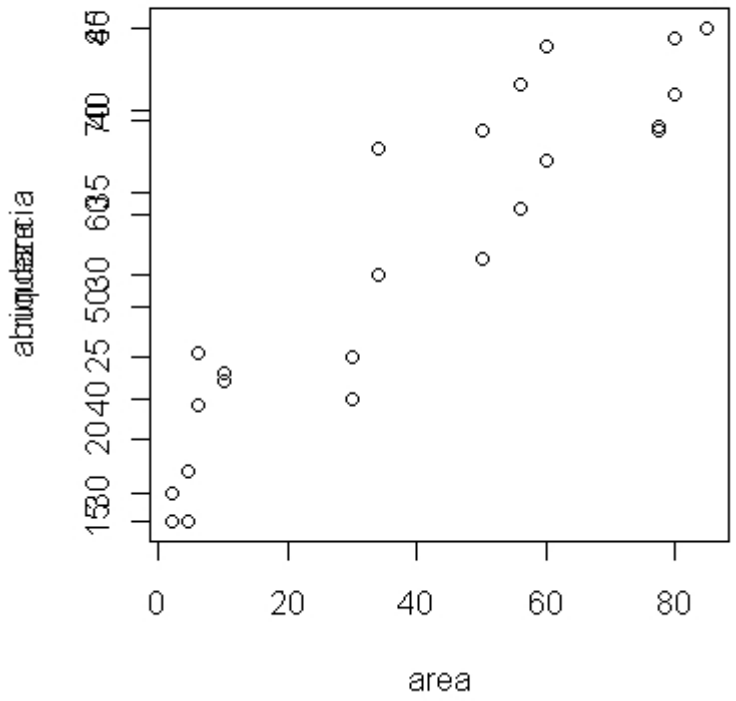
```
plot(riqueza~area)
mtext("legenda no lado errado", side=4, line=0.9, at=20,cex=2,
family="serif")
```



par(new=TRUE)

Para sobrepor um novo gráfico a um gráfico já existente. Em vez de criar gráficos lado-a-lado, como em `par(mfrow=c())`, este argumento irá desenhar o novo gráfico sobre o gráfico anterior.

```
plot(riqueza~area)
par(new=TRUE)
plot(abundancia~area)
```

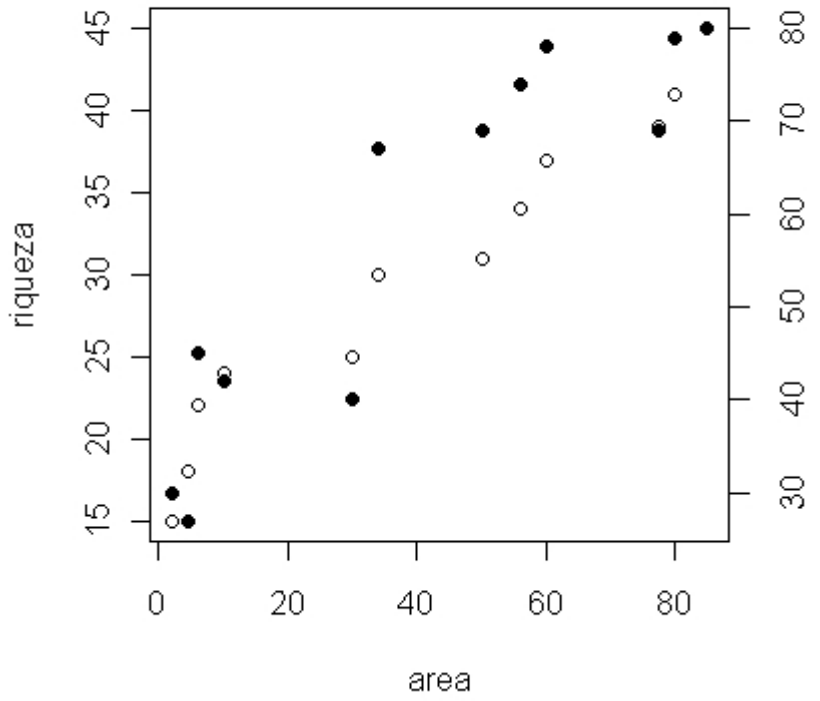


Mas reparem que aqui será necessário alguns ajustes para suprimir eixos e legendas. Em muitos casos quando se está inserindo informações será necessário suprimir parâmetros.

axis()

Para se inserir um eixo novo. Esta função é bastante usada nos casos em que se deseja ter dois gráfico dentro de uma mesma figura (ver `par(new=TRUE)`), ou então se deseja controlar muitos dos parâmetros dos eixos (como em `mtext()`).

```
plot(riqueza~area)
par(new=TRUE)
plot(abundancia~area, axes=FALSE, ann=FALSE, pch=16)
axis(4)
```

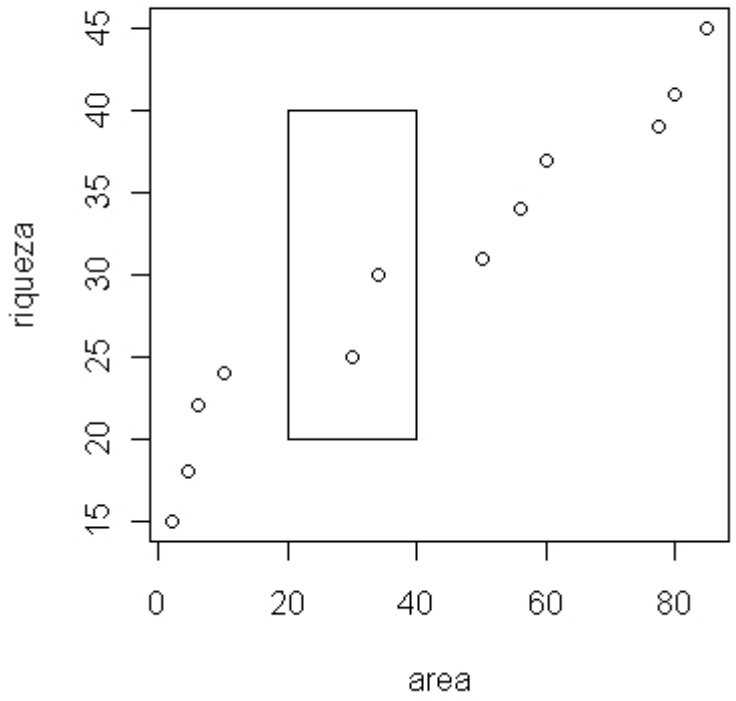


Aqui no caso será necessário usar `axes=F` para suprimir a criação dos eixos do gráfico inicial de abundância e `ann=F` para suprimir a legenda de abundância do lado direito. Para para diferenciar os pontos entre os dois plots usar `pch=16`, ou qualquer outro número. Para inserir a legenda de abundância do lado direito será necessário usar `mtext()`, mas daí será necessário mudar outros parâmetros como distância da margem.

arrows(), rect(), polygon()

Para inserir flechas ou **barras de erros** use `arrows()`. Já para inserir retângulos, polígonos e outros formatos use `rect()` e `polygon()`.

```
plot(riqueza~area)
rect(20,20,40,40)
```



Salvando Gráficos

Após ter feitos todos os gráficos desejados, é possível salvá-los em vários formatos, como jpeg, png, postscript, pdf. Consulte a ajuda do pacote grDevices para a lista completa e mais informações.

Após chegar ao gráfico final, ajustando todos os parâmetros desejados, você pode usar a função do R para criar o arquivo no formato desejado. Há funções para cada formato de arquivo, todas elas com o primeiro argumento `filename`, que especifica o nome do arquivo a salvar. Para criar um arquivo *jpg*, por exemplo, há a função `jpeg`:

```
  jpeg(filename = "Algumnome.jpg")
```

Feito isso, o R agora irá enviar todos os resultados de comandos gráficos para este arquivo, que é fechado com a função `dev.off()`.

Exemplo

```
jpeg(filename = "Rplotaula.jpg", width = 480, height = 480,
     units = "px", pointsize = 12, quality = 100,
     bg = "white",  res = NA, restoreConsole = TRUE)

par(mfrow=c(1,2))
par(mar=c(14,4,8,2))
plot(riqueza~area)
boxplot(riqueza~area.cate)
```

```
dev.off()
```

Quatro Fatos Importantes sobre Arquivos de Figuras (e uma dica)

- Seu arquivo de figura só terá os parâmetros desejados se você executar todo o código após abrir o arquivo da figura, incluindo todos os comandos `par()`.
- Seu arquivo de figura só será salvo quando você executar o comando `dev.off()`. Até que isso aconteça, **todos os resultados de comandos gráficos continuarão a ser enviados para este arquivo**.
- Por isso, se você criar dois gráficos, o segundo substituirá o primeiro.
- Ao executar o comando `dev.off()`, o arquivo será gravado no diretório de trabalho. Se você quiser gravá-lo em outro lugar, terá que especificar o caminho completo no comando de criação do arquivo, no argumento `filename`.

DICA

Para gravar uma sequência de gráficos sem precisar dar vários comandos de abertura de arquivos, use no argumento do nome do arquivo a notação `nome%[número]d.extensão`, onde [número] normalmente são os números 01, 02 ou 03:

```
png("meugrafico%02d.png")
```

Com essa notação os gráficos gerados serão gravados com uma numeração sequencial, que tem `[numero]` algarismos. Por exemplo `[número] = 01` indica numeração sequencial de um algarismo, e você pode gravar até nove gráficos (*meugrafico1.png* a *meugrafico9.png*). Se `[número] = 02` a numeração sequencial tem dois algarismos, portanto você pode gravar até 99 figuras (*meugrafico01.png* a *meugrafico99.png*).

Se você gera mais gráficos do que este valor máximo, os excedentes sobrepõem os primeiros, na sequência. Para evitar isso, normalmente usamos `[número] = 03`, que permite gerar até 999 arquivos, o que é mais do que suficiente na maioria dos casos.

O padrão das funções de arquivos gráficos do tipo bitmap no R é gerar arquivos com o nome *Rplotxxx.extensão*, com numeração sequencial com 3 algarismos, ou seja:

```
bmp(filename = "Rplot%03d.bmp")
jpeg(filename = "Rplot%03d.jpg")
png(filename = "Rplot%03d.png")
tiff(filename = "Rplot%03d.bmp")
```

Para arquivos do tipo *postscript* e *pdf* o padrão é um pouco diferente, consulte a ajuda.

Dispositivos Gráficos

Para operar bem com arquivos gráficos no R, é preciso entender o conceito de **dispositivo gráfico** (*graphical device*).

Um dispositivo (*device*) é qualquer unidade de entrada (**I**=*input*) ou saída (**O**=*output*) em um

computador. O teclado é um dispositivo de entrada (**I**), o monitor de vídeo de saída (**O**) e o disco rígido de ambos (**I/O**).

Em termos bem gerais, o R usa o padrão de sistemas da família UNIX e abre arquivos especiais (*device files*) para controlar cada dispositivo de saída gráfica [1)]. Na prática podemos podemos pensar que o R cria uma "rota" para cada dispositivo, e que você pode ter várias destas "rotas" abertas ao mesmo tempo, **mas apenas uma ativa por vez**.

Abrindo e trocando de dispositivos

Quando executamos um comando com resultado gráfico, o padrão do R é enviar seu resultado para um dispositivo de tela. Se não há nenhum aberto, este dispositivo de tela será aberto. Por exemplo, se iniciamos o R e executamos um comando `plot`:

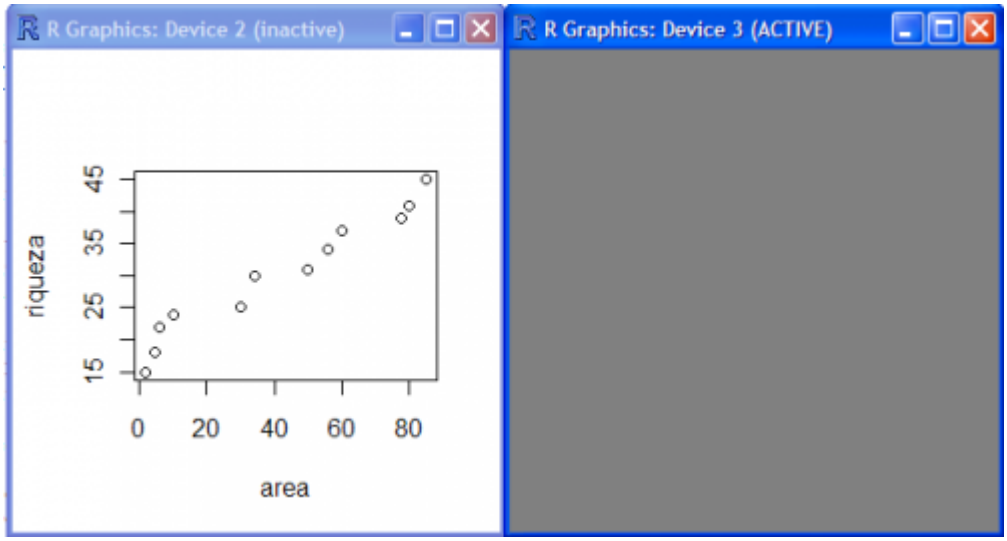
```
plot(riqueza~area)
```

uma janela se abrirá para exibir a figura nele. Este é um **dispositivo gráfico de tela**, normalmente do tipo X11 em sistemas UNIX, e do tipo `windows` em sistemas Microsoft Windows.

É possível também abrir mais dispositivos de tela, com o comando:

```
> x11()
```

que funciona nos dois tipos de sistemas operacionais. Ao executar este comando, uma nova janela gráfica em branco se abrirá, na margem superior da qual você verá a indicação de que agora este é o dispositivo ativo:



O dispositivo ativo é que receberá o resultado de todos os comandos gráficos. Assim,você pode criar um novo gráfico neste dispositivo, ou alterar seus parâmetros, sem afetar o gráfico que está na outra janela. Quando você abre um novo dispositivo, os parâmetros são os mantidos por padrão no R, que você obtém com o comando `par()`. Para mudá-los, é preciso usar o comando `par`, como explicado nas seções anteriores.

Para trocar de dispositivo ativo, use a função `dev.set(which=)`, cujo o argumento `which` é o número do dispositivo que você quer tornar ativo. Por exemplo, para voltar à janela do dispositivo 2

execute:

```
> dev.set(which=2)
```

Qual o Dispositivo Ativo?

Enquanto temos apenas dispositivos de tela abertos, é fácil descobrir qual é o ativo, pois esta informação é exibida na janela de cada um. Mas quando abrimos um ou mais dispositivos de arquivo, como:

```
> png("figura%02d.png")
> pdf("figura%02d.pdf")
```

é fácil se perder. Para que isso não aconteça há as funções `dev.list`, para listar todos os dispositivos abertos, e `dev.cur`, que retorna o dispositivo ativo:

```
> dev.list()
           windows            windows png:figura%02d.png                pdf
                 2                  3                   4                  5
> dev.cur()
pdf
  5
> dev.set(3)
windows
      3
> dev.cur()
windows
      3
```

Você fecha um dispositivo com o comando `dev.off(which=n)`, em que *n* é o número do dispositivo. Se este argumento é omitido, o valor padrão é o dispositivo ativo:

```
> dev.list()
           windows            windows png:figura%02d.png                pdf
                 2                  3                   4                  5
> dev.cur()
pdf
  5
> dev.off()
windows
      2
> dev.list()
           windows            windows png:figura%02d.png
                 2                  3                   4
> dev.cur()
windows
      2
```

Quando o dispositivo ativo é fechado, o seguinte na lista de dispositivos torna-se o ativo. O último

comando acima ilustra isto.

> Exercício 5
>
> Crie diferentes gráficos em diferentes dispositivos. Por fim salve-os em jpeg.

Para saber mais como salvar gráficos em jpeg use a função ‘’?jpeg’’.

1)

veja este link para uma explicação mais precisa