

# Gráfico com R

Vanderlei Júlio Debastiani ([vanderleidebastiani@yahoo.com.br](mailto:vanderleidebastiani@yahoo.com.br))  
05 Janeiro 2019

## Índice de conteúdo

<b>1</b>	<b>Introdução</b>	<b>2</b>
<b>2</b>	<b>Funções básicas</b>	<b>2</b>
2.1	Função <i>plot()</i>	2
2.2	Função <i>par()</i>	5
2.3	Alterando eixos	6
2.4	Desconstruindo o gráfico	9
2.5	Adicionando pontos	11
2.6	Adicionando contornos	12
2.7	Adicionando eixos	13
2.8	Adicionando linhas	14
2.9	Adicionando legendas	15
2.10	Adicionando linhas simples	16
2.11	Adicionando setas	17
2.12	Adicionando textos	18
2.13	Adicionando segmentos	20
2.14	Adicionando grades	21
2.15	Adicionando títulos	22
2.16	Adicionando polígonos	24
2.17	Função <i>locator()</i>	26
2.18	Expressões	26
2.19	Cores	28
2.20	Fontes	31
2.21	Reciclagem	31
<b>3</b>	<b>Princípios do design eficaz de um gráfico</b>	<b>36</b>
<b>4</b>	<b>Gráficos personalizados</b>	<b>37</b>
4.1	Gráfico de barras	37
4.2	Gráfico de barras empilhadas	39
4.3	Gráfico de pontos com linha de regressão	42
4.4	Gráfico de pontos com ajustes não lineares	45
4.5	Diagrama de dispersão	49
4.6	Boxplot	55
4.7	Beanplot	59
4.8	Gráfico de interação	62
4.9	Gráfico de pontos com barras de erro	65
4.10	Gráfico de linhas com duas escalas	68
4.11	Histogramas	75
4.12	Pares gráficos	79
<b>5</b>	<b>Painéis gráficos</b>	<b>81</b>
5.1	Margens	81
5.2	Configurar painéis	83
5.3	Subgráficos	85
5.4	Escala de cores	86

<b>6 Exportar gráficos</b>	<b>90</b>
6.1 Exportar como imagem . . . . .	91
6.2 Exportar como imagem vetorial . . . . .	92
6.3 Função recordPlot() . . . . .	93
<b>7 Guia de ajuda rápida</b>	<b>96</b>
<b>8 Conclusão</b>	<b>102</b>
<b>9 Mais informações</b>	<b>102</b>
<b>10 Referências</b>	<b>102</b>

## 1 Introdução

R é uma linguagem e ambiente de programação estatística e gráfica. Considerando apenas a parte gráfica existem muitos recursos, sendo possível fazer desde gráficos simples de pontos ou barras até figuras complexas como mapas e filogenias. Para fazer um gráfico muitos parâmetros devem ser especificados sendo que a tarefa de produzir um bom gráfico não é das tarefas mais fáceis. Os problemas são principalmente pela quantidade de parâmetros que podem ser especificados e que a documentação que não é apresentada de maneira visual.

A proposta deste texto é mostrar as principais opções para construção de gráficos utilizando as funções gráficas básicas. O objetivo não é explorar todos os parâmetros e nem esgotar as possibilidades do R, apenas mostrar as principais funções e argumentos. O conjunto básico de funções gráficas já é suficiente para produzir bons gráficos, embora essa não seja a única opção disponível no programa. Pacotes como *ggplot2* estão se tornando populares por aparentemente apresentar soluções mais simples e eficientes. Entretanto as opções básicas são bastante flexíveis para adicionar qualquer elemento gráfico sem a necessidade de formatação prévia específica.

Este texto é dividido em cinco partes. A primeira parte apresenta as funções básicas para adicionar os elementos gráficos básicos em qualquer gráfico. A segunda apresenta resumidamente alguns princípios de design eficaz de gráficos. A terceira parte aborda a construção de gráficos personalizados onde são abordados os tipos de gráficos mais comuns utilizados nas publicações. A quarta parte apresenta opções para a elaboração de painéis gráficos. A quinta parte aborda opções de exportação. Ao final também é apresentado um guia de ajuda rápida mostrando visualmente as principais argumentos das funções gráficas. Os códigos apresentados ao longo do texto podem ser executados passo-a-passo para entender o funcionamento de cada argumento sendo que os dados de exemplos provêm de conjuntos de dados inclusos em alguns pacotes do R. Para carregar esses dados basta usar a função *data* com o nome do conjunto de dados (ex, *data("anscombe")*). Além disso, antes de executar os códigos dos gráficos os dados foram organizados em *dataframes* para padronizar a indexação das variáveis.

## 2 Funções básicas

Para construir um gráfico usa-se função genérica *plot*. A função *plot* é uma função que produz gráficos de maneira automática dependendo do tipo de informação e classe do objeto passado para ela. Quando apropriado, os eixos, nomes dos eixos e títulos são gerados automaticamente (a menos se especificado o contrário). Ainda existem várias funções auxiliares para adicionar elementos ao gráfico.

### 2.1 Função *plot()*

#### 2.1.0.1 Dados

Dados *anscombe* é um conjunto de dados didáticos para uso em regressões lineares. O conjunto é formado por 4 pares de variáveis independente e dependente totalizando 11 observações.

```

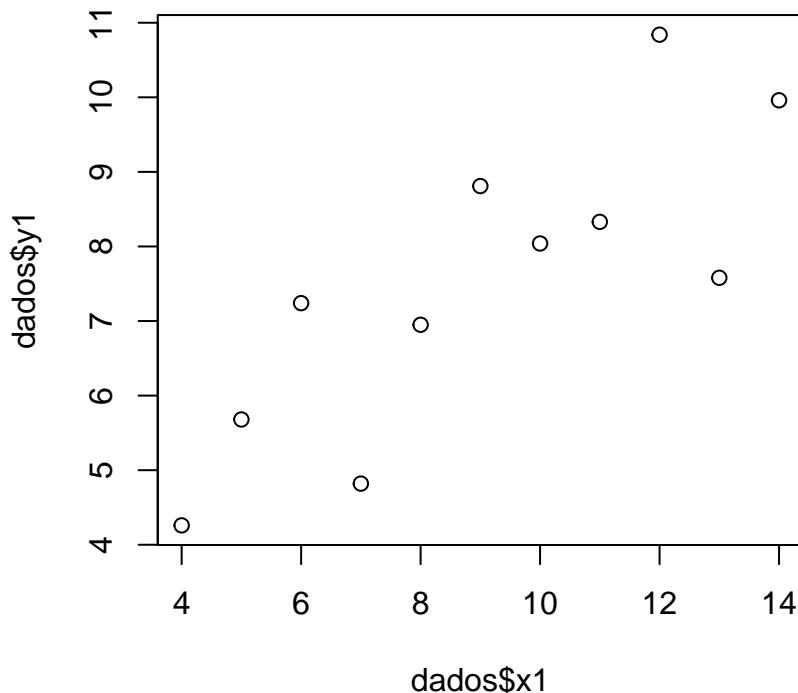
data(anscombe)
dados <- anscombe
str(dados)
'data.frame': 11 obs. of 8 variables:
$ x1: num 10 8 13 9 11 14 6 4 12 7 ...
$ x2: num 10 8 13 9 11 14 6 4 12 7 ...
$ x3: num 10 8 13 9 11 14 6 4 12 7 ...
$ x4: num 8 8 8 8 8 8 19 8 8 ...
$ y1: num 8.04 6.95 7.58 8.81 8.33 ...
$ y2: num 9.14 8.14 8.74 8.77 9.26 8.1 6.13 3.1 9.13 7.26 ...
$ y3: num 7.46 6.77 12.74 7.11 7.81 ...
$ y4: num 6.58 5.76 7.71 8.84 8.47 7.04 5.25 12.5 5.56 7.91 ...

```

### 2.1.0.2 Gráficos de pontos

Se  $x$  e  $y$  são dois vetores do tipo *numeric* o gráfico resultante é de pontos, sendo mostrado  $x$  as coordenadas dos pontos no eixo horizontal e  $y$  as coordenadas do eixo vertical.

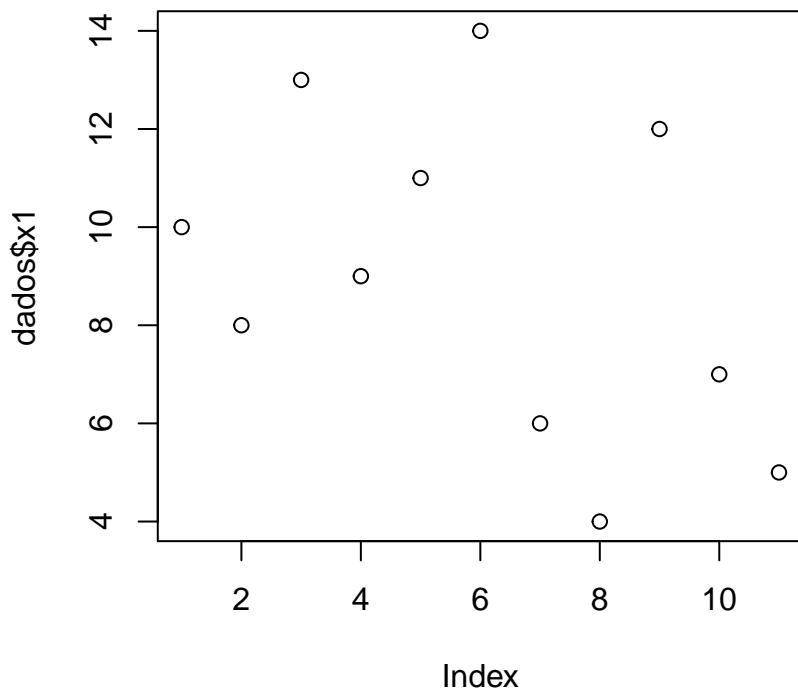
```
plot(dados$x1, dados$y1) # plot(y1 ~ x1, data=dados) produz o mesmo resultado.
```



### 2.1.0.3 Gráficos de séries

Se apenas um vetor do tipo *numeric* é fornecido a função gera uma série com os dados, seguindo a ordem que os valores estão no vetor.

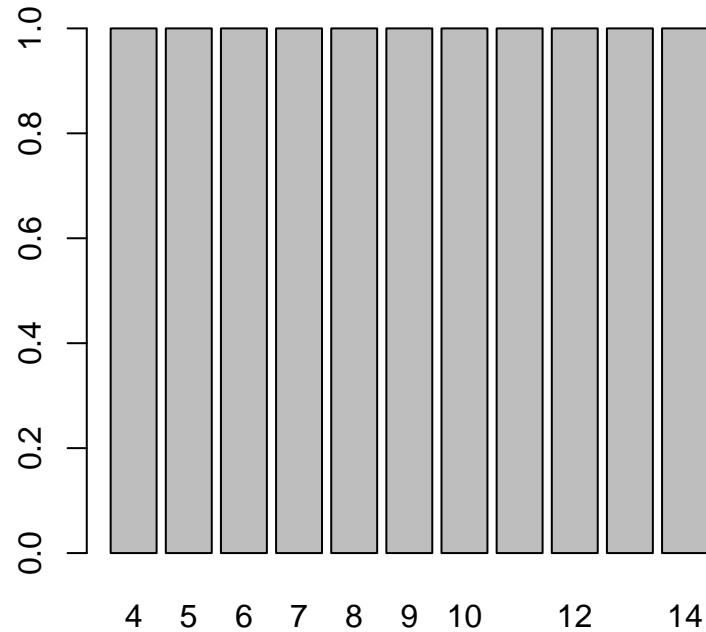
```
plot(dados$x1)
```



#### 2.1.0.4 Gráficos de barra

Se apenas um vetor do tipo *factor* é fornecido a função gera um gráfico de barra.

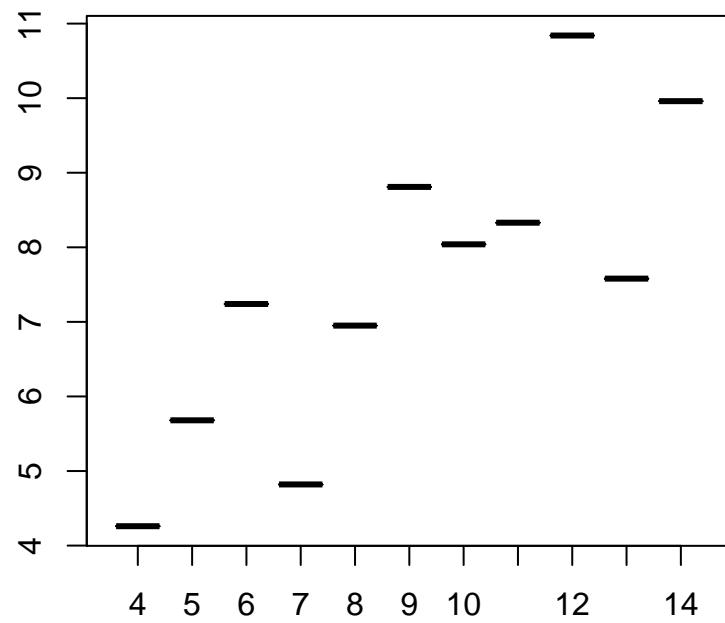
```
plot(as.factor(dados$x1))
```



#### 2.1.0.5 Boxplot

Se são fornecidos um vetor do tipo *factor* e outro do tipo *numeric* a função plot gera um *boxplot*.

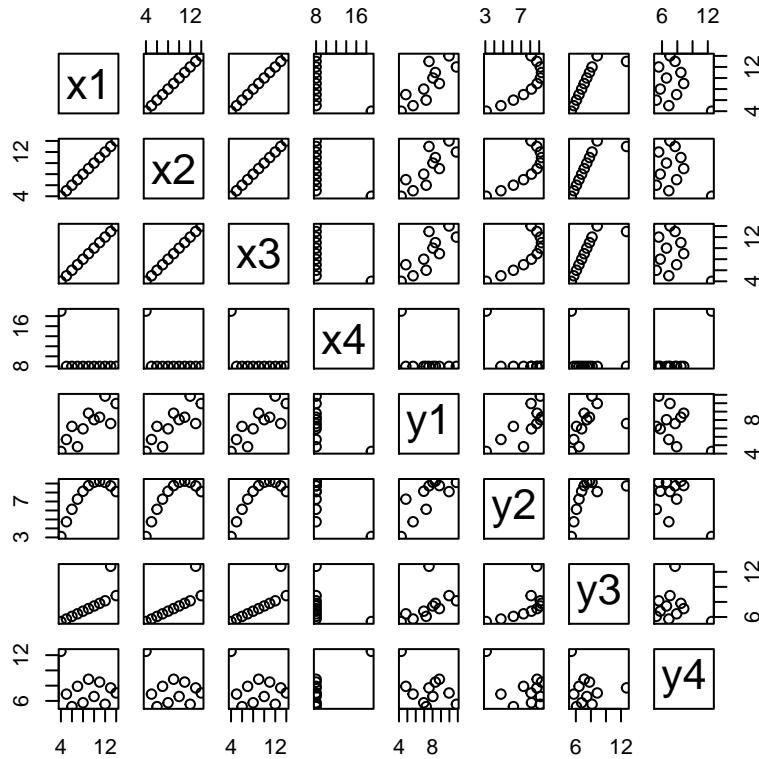
```
plot(as.factor(dados$x1), dados$y1)
```



#### 2.1.0.6 Painéis gráficos

Se um objeto do tipo *dataframe* é fornecido a função gera gráficos com os pares de todas as variáveis.

```
plot(dados)
```



## 2.2 Função *par()*

Os parâmetros gráficos gerais são predefinidos pela função *par*, e podem ser acessados pela função de ajuda *?par*. O R possui vários dispositivos que são usados para gerar os gráficos. Normalmente usa-se um

dispositivo para visualizar os gráficos e outro para exportar para o gráfico para um arquivo de imagem ou vetorial. Informações sobre cada dispositivo podem ser obtidas na ajuda `?Devices`, sendo que cada dispositivo possui uma lista de parâmetros própria.

```
ls(par()) # Parâmetros gráficos do dispositivo ativo
[1] "adj"      "ann"      "ask"      "bg"       "bty"
[6] "cex"      "cex.axis"  "cex.lab"   "cex.main"  "cex.sub"
[11] "cin"      "col"      "col.axis"  "col.lab"   "col.main"
[16] "col.sub"  "cra"      "crt"      "csi"      "cxy"
[21] "din"      "err"      "family"   "fg"       "fig"
[26] "fin"      "font"     "font.axis" "font.lab"  "font.main"
[31] "font.sub" "lab"      "las"      "lend"    "lheight"
[36] "ljoin"    "lmitre"   "lty"      "lwd"     "mai"
[41] "mar"      "mex"      "mfcol"    "mfg"     "mfrow"
[46] "mgp"      "mhk"      "new"     "oma"     "omd"
[51] "omi"      "page"    "pch"      "pin"     "plt"
[56] "ps"       "pty"      "smo"     "srt"     "tck"
[61] "tcl"      "usr"      "xaxp"    "xaxs"    "xaxt"
[66] "xlog"    "xpd"      "yaxp"    "yaxs"    "yaxt"
[71] "ylbias"  "ylog"
```

Cada parâmetro define uma opção de configuração do gráfico. Por exemplo, o parâmetro `pch` define o tipo de símbolo que é usado em um gráfico de pontos. Os parâmetros definem uma infinidade de opções como cor e tamanho do texto, tamanho das margens e tipos de eixos. Pode-se alterar esses parâmetros de duas formas.

A primeira maneira de alterar um parâmetro gráficos é especificando o nome do parâmetro e o novo valor que receberá diretamente da função `plot`. Desta maneira o gráfico que será construído mostrará a nova opção, conforme especificado. Uma segunda maneira de alterar é especificar na função `par`, antes de começar a construir o gráfico. Desta maneira os parâmetros serão alterados de maneira permanente para todos os gráficos feitos pelo dispositivo, até o dispositivo ser reiniciado. De uma maneira geral os parâmetros mais gerais da figura como tamanho das margens só podem ser alterados pela função `par`.

Pode-se salvar as alterações da função `par` em um objeto, isso permite que os parâmetros alterados possam ser restaurados facilmente a qualquer momento.

```
op <- par(las = 1) # Alterar parâmetro las para todos os gráficos
par(op) # Restaurar parâmetros originais
```

## 2.3 Alterando eixos

Uma das opções mais utilizada na elaboração de gráficos se refere as alterações dos aspectos relacionados aos eixos. Em quase todos os gráficos altera-se os nomes dos eixos, títulos e os limites dos eixos.

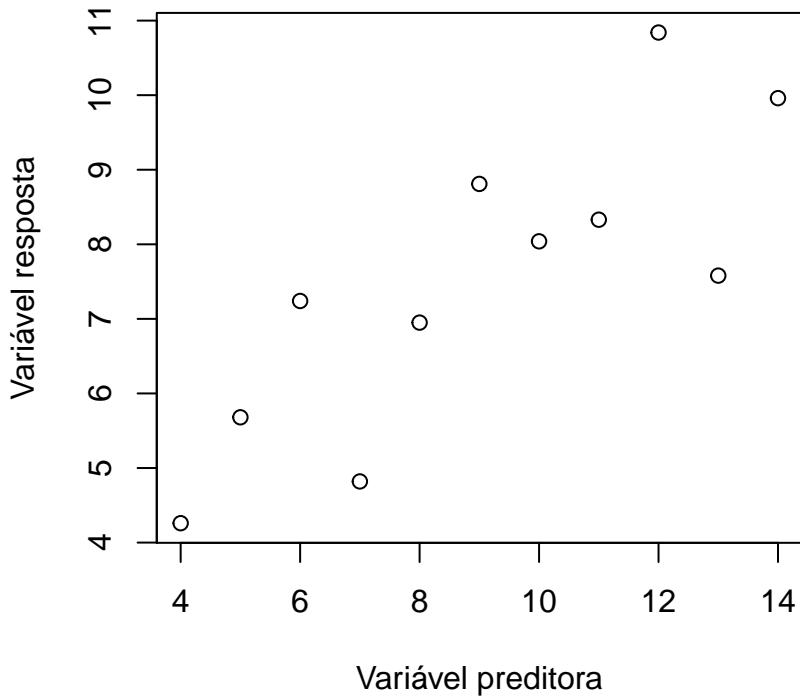
Argumentos:

**xlab** e **ylab** - Alterar nomes dos eixos *x* e *y* respectivamente. Podem ser usados junto com `font.lab` para alterar o tipo de fonte, `col.lab` para alterar cor e `cex.lab` para alterar o tamanho da fonte.

**main** - Alterar nomes do título do gráfico. Pode ser usado junto com `font.main`, `col.main` e `cex.main`, para alterar fonte, cor e tamanho do título respectivamente.

```
plot(dados$x1, dados$y1,
      xlab = "Variável preditora", ylab = "Variável resposta",
      main = "Anscombe")
```

## Anscombe



**las** - Alterar orientação do números nos eixos. Valores aceitos 0 (paralelo ao eixo), 1 (horizontal), 2 (perpendicular) e 3 (vertical).

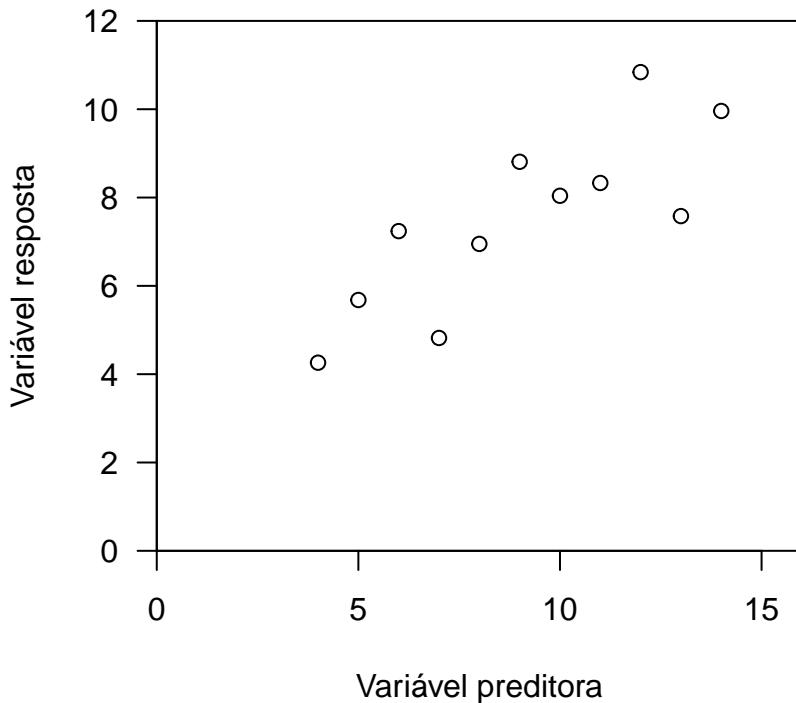
**xlim** e **ylim** - Alterar limites dos eixos  $x$  e  $y$  respectivamente. Argumento do tipo vetor com dois valores, sendo o primeiro valor o mínimo e o segundo o valor máximo do eixo.

**xaxs** e **yaxs** - Alterar o estilo de cálculo dos limites dos eixos  $x$  e  $y$  respecivamente. Valores aceitos "r" (o limite de cada eixo estende por quatro porcento em cada lado do eixo) e "i" (o limite fica exatamente no estabelecidos pelos argumentos **xlim** e **ylim** ou pelos dados).

**cex.axis** - Altera o tamanho da fonte para os valores dos eixos. Argumento numérico com padrão = 1. Pode ser usado junto com **font.axis** para alterar o tipo de fonte e **col.axis** para alterar a cor.

```
plot(dados$x1, dados$y1,
      xlab = "Variável preditora", ylab = "Variável resposta",
      main = "Anscombe",
      las = 1,
      xlim = c(0, 16), ylim = c(0, 12),
      xaxs = "i", yaxs = "i",
      cex.axis = 1)
```

## Anscombe



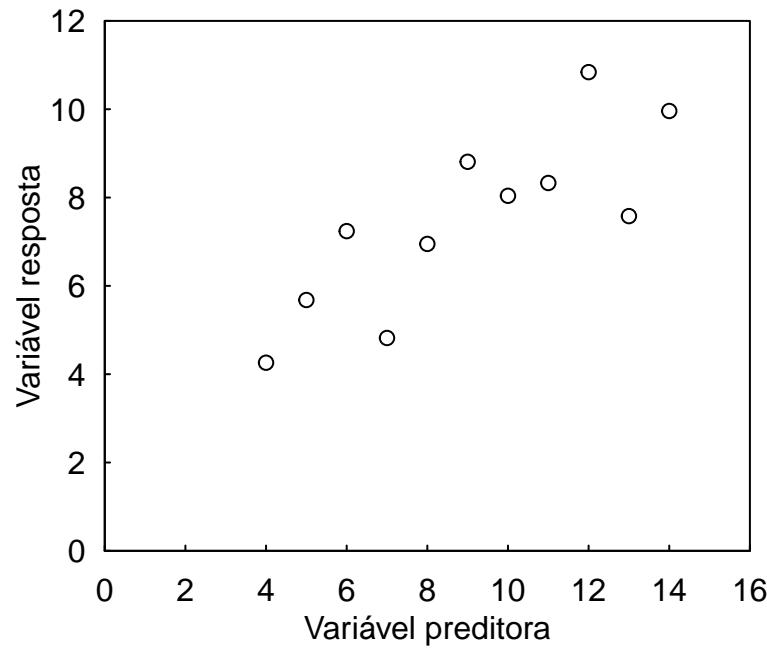
**xaxp** e **yaxp** - Alterar a posição e quantidade de separadores dos eixos  $x$  e  $y$  respectivamente. Argumento do tipo vetor com  $c(x1, x2, n)$ , onde  $x1$  e  $x2$ , são o mínimo e máximo e representam o alcance do separadores e  $n$  a quantidade de espaços internos.

**tck** - Alterar o comprimento dos separadores dos eixos. Argumento numérico, valores expressos em fração do tamanho do eixo. Se positivos os marcadores ficam na parte interna do gráfico, se negativo ficam na parte externa e se zero separadores não são mostrados.

**mgp** - Alterar a linha da margem onde são mostrados os textos do eixos, valores dos eixos e próprio eixo. Argumento do tipo vetor com padrão  $c(3, 1, 0)$ . O primeiro valor altera texto do eixo, o segundo altera os valores dos eixos e o terceiro linha no próprio eixo. Valores próximos de 0 aproximam os textos ou valores ao eixo. Pode ser usado valores negativos.

```
plot(dados$x1, dados$y1,
      xlab = "Variável preditora", ylab = "Variável resposta",
      main = "Anscombe",
      las = 1,
      xlim = c(0, 16), ylim = c(0, 12),
      xaxs = "i", yaxs = "i",
      cex.axis = 1,
      xaxp = c(0, 16, 8), yaxp = c(0, 12, 6),
      tck = 0.01,
      mgp = c(1.5, 0.5, 0))
```

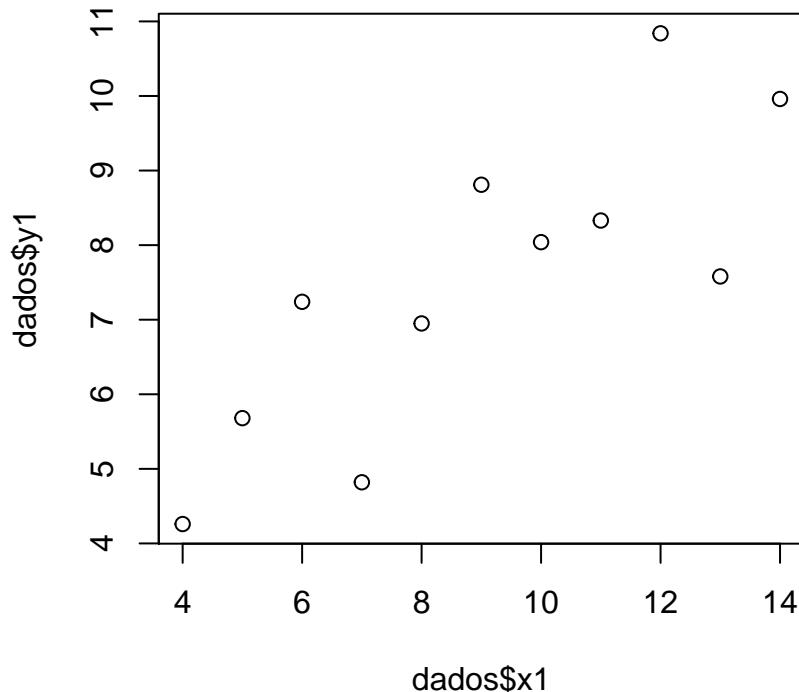
## Anscombe



### 2.4 Desconstruindo o gráfico

Assim como elementos podem ser adicionados ao gráfico, eles também podem ser removidos.

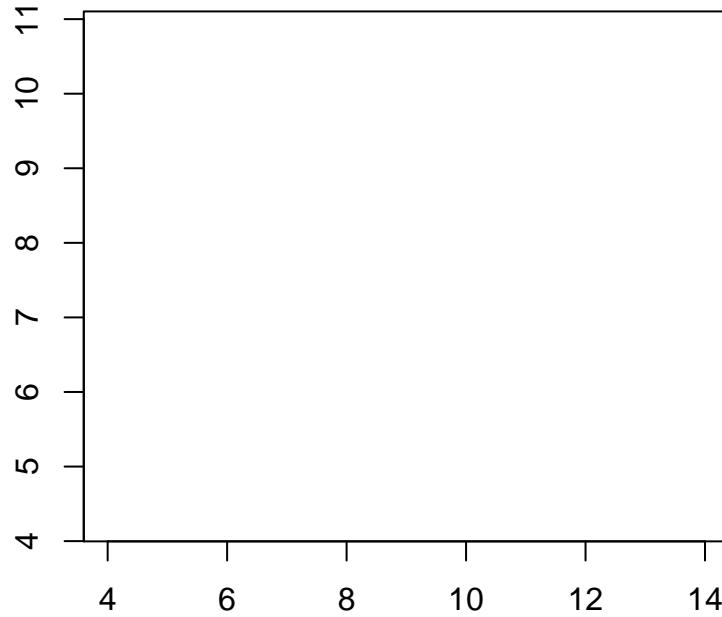
```
plot(dados$x1, dados$y1)
```



Argumentos:

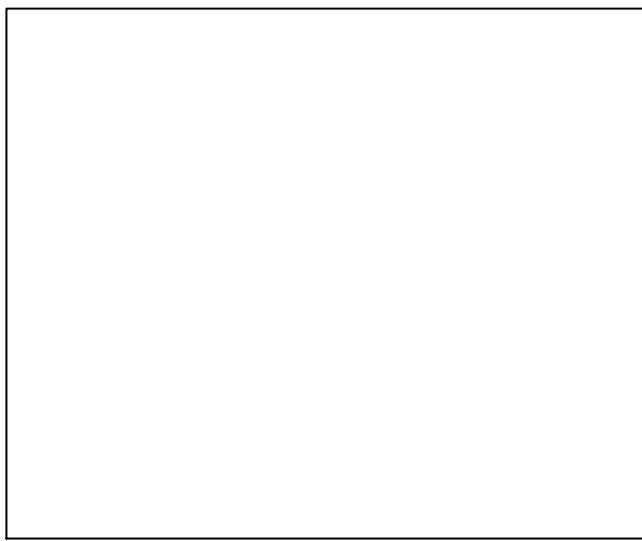
**type** - Alterar o tipo de gráfico que será construído. Vários valores são aceitos, os principais são “p” (gráfico de pontos), “l” (gráfico de linhas), “b” (gráficos com pontos e linhas) e “n” (para não mostrar nada referente aos dados).

```
plot(dados$x1, dados$y1,  
      type = "n",  
      xlab = "", ylab = "")
```



**xaxt** e **yaxt** - Especificar se o eixo  $x$  e  $y$ , respectivamente, deve ser mostrado. Quando o valor for “n” o eixo não é mostrado.

```
plot(dados$x1, dados$y1, type = "n",  
      xlab = "", ylab = "",  
      xaxt = "n", yaxt = "n")
```



**bty** - Alterar o tipo de caixa (box) usado envolta do gráfico. Valores aceito “o”, “l”, “7”, “c”, “u” e “[” que resultam em contorno igual a forma do símbolo ou “n” para não mostrar o contorno.

```
plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")
```

A figura acima ficou totalmente branca, sem elementos visíveis. As margens e limites dos eixos são os mesmos do gráfico original, mas os elementos foram ocultos.

## 2.5 Adicionando pontos

A função auxiliar *points* adiciona pontos ou linhas ao gráfico.

Argumentos:

**x** e **y** - Valor único ou vetores com as coordenadas dos pontos para os eixos *x* e *y* respectivamente.

**pch** - Alterar o tipo de símbolo dos pontos. Valores aceitos entre 0 e 25. Se for um único valor todos os símbolos serão idênticos, se for um vetor cada ponto receberá um símbolo conforme especificado.

**cex** - Alterar o tamanho do símbolos. Argumento com valor numérico com padrão = 1. Se for um único valor todos os símbolos serão do mesmo tamanho, se for um vetor cada ponto será do tamanho especificado.

```
plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")
```

```
points(x = dados$x1, y = dados$y1,
```

```
  pch = 23,  
  cex = 1.3)
```



## 2.6 Adicionando contornos

A função auxiliar *box* adiciona linhas de contorno ao gráfico.

Argumentos:

**bty** - Alterar o tipo de caixa (box) usado envolta do gráfico. Valores aceito “o”, “l”, “7”, “c”, “u”e “[” que resultam em contorno igual a forma do símbolo ou “n” que não mostra contorno.

```
plot(dados$x1, dados$y1, type = "n",  
      xlab = "", ylab = "",  
      xaxt = "n", yaxt = "n",  
      bty = "n")  
  
points(x = dados$x1, y = dados$y1,  
       pch = 23,  
       cex = 1.3)  
  
box(bty = "1")
```



## 2.7 Adicionando eixos

Função auxiliar *axis* adiciona eixos ao gráfico.

Argumentos:

**side** - Especificar a posição do eixo. Argumento numérico com os valores de 1 (eixo inferior), 2 (eixo à esquerda), 3 (eixo superior) e 4 (eixo à direita).

**las** - Alterar orientação do rótulos do eixo. Valores aceitos 0 (paralelo ao eixo), 1 (horizontal), 2 (perpendicular) e 3 (vertical).

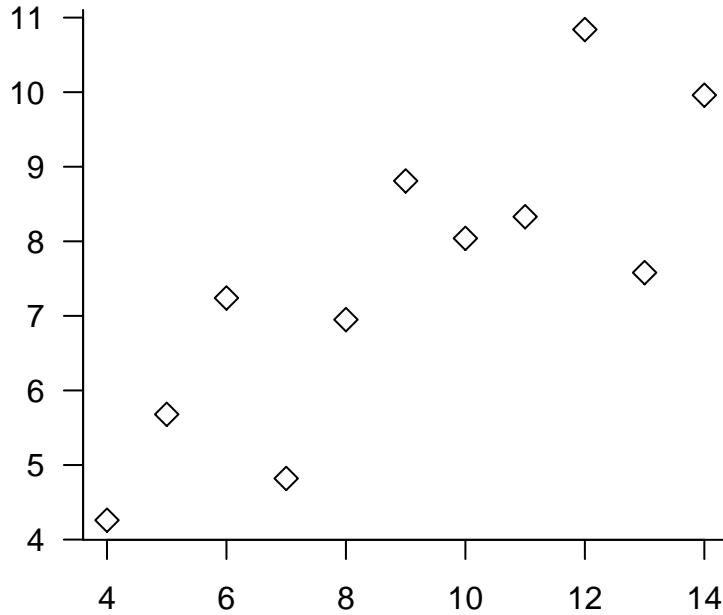
```
plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")

points(x = dados$x1, y = dados$y1,
       pch = 23,
       cex = 1.3)

box(bty = "l")

axis(side = 1)

axis(side = 2,
     las = 1)
```



## 2.8 Adicionando linhas

Função auxiliar *lines* adiciona linhas a um gráfico. Pode ser usada para adicionar linhas de modelos lineares.

Argumentos:

**lty** - Alterar o tipo de linha. Valores aceitos 0 (sem linha), 1 (linha sólida), 2 (linha tracejada), 3 (linha pontilhada), 4 (pontilhada e tracejada), 5 (linha com traço longo) e 6 (linha com traço duplo). Alternativamente pode ser usado os valores “blank”, “solid”, “dashed”, “dotted”, “dotdash”, “longdash”e “twodash”, que substitui os números de 1 a 6 respectivamente.

**lwd** - Alterar a espessura da linha. Argumento numérico com padrão = 1.

```

mod <- lm(dados$y1 ~ dados$x1)
mod # Modelo linear

Call:
lm(formula = dados$y1 ~ dados$x1)

Coefficients:
(Intercept)      dados$x1
               3.0001      0.5001

ypredito <- predict(mod)
ypredito # Valores preditos pelos modelo linear
     1       2       3       4       5       6       7
8.001000 7.000818 9.501273 7.500909 8.501091 10.001364 6.000636
     8       9      10      11
5.000455 9.001182 6.500727 5.500545

plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")

points(x = dados$x1, y = dados$y1,

```

```

    pch = 23,
    cex = 1.3)

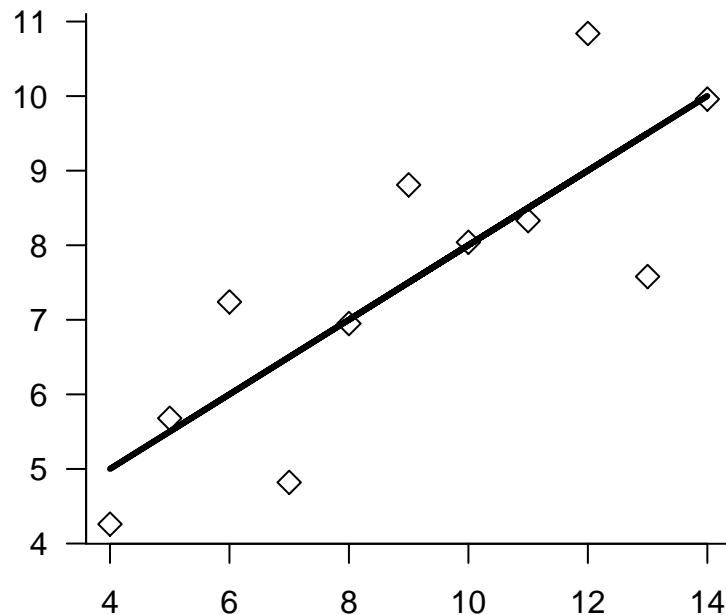
box(bty = "1")

axis(side = 1)

axis(side = 2,
     las = 1)

lines(x = dados$x1, y = ypredito,
      lty = 1,
      lwd = 3)

```



## 2.9 Adicionando legendas

A função *legend* adiciona legenda ao gráfico.

Argumentos:

**x** e **y** - Coordenadas para a posição da legenda. Alternativamente pode ser substituído pelos valores “bottomright”, “bottom”, “bottomleft”, “left”, “topleft”, “top”, “topright”, “right” e “center” para usar posições predefinidas.

**legend** - Texto que será mostrado na legenda. Pode ser uma única expressão ou uma vetor, sendo que cada um representará um item na legenda.

**lty, lwd, pch, angle e density** - Especificar o tipo de símbolo usado na legenda. Os argumentos são os mesmo usado na construção de linhas (*lty* e *lwd*), pontos (*pch*) e barras (*angle* e *density*).

**bty** - Alterar o tipo de caixa (box) usado envolta da legenda. Valores aceito “o” (com contorno) e “n” (sem contorno).

```

plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")

points(x = dados$x1, y = dados$y1,

```

```

    pch = 23,
    cex = 1.3)

box(bty = "1")

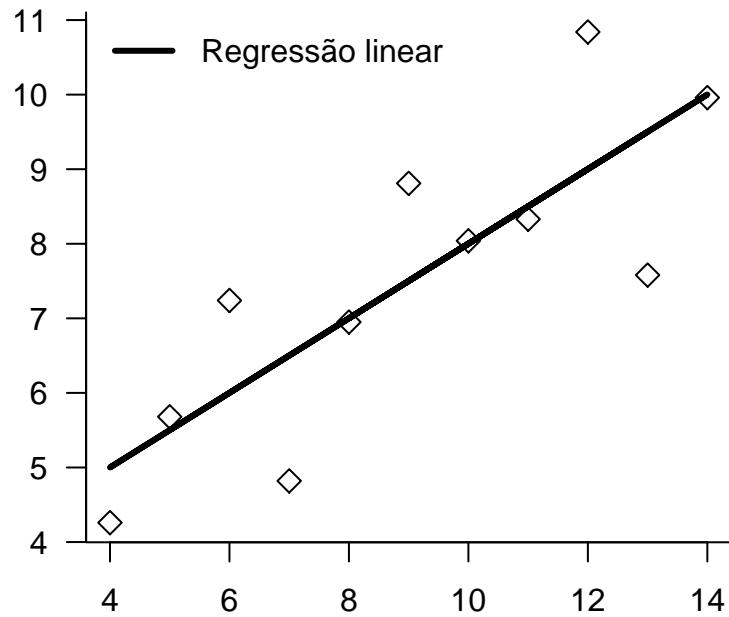
axis(side = 1)

axis(side = 2,
     las = 1)

lines(x = dados$x1, y = ypredito,
      lty = 1,
      lwd = 3)

legend(x = "topleft",
       legend = "Regressão linear",
       lty = 1,
       lwd = 3,
       bty = "n")

```



## 2.10 Adicionando linhas simples

Função auxiliar *abline* adiciona linhas simples ao gráfico.

Argumentos:

**h** e **v** - Especificar valor para linha horizontal e vertical respectivamente.

```

plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")

points(x = dados$x1, y = dados$y1,
       pch = 23,

```

```

cex = 1.3)

box(bty = "1")

axis(side = 1)

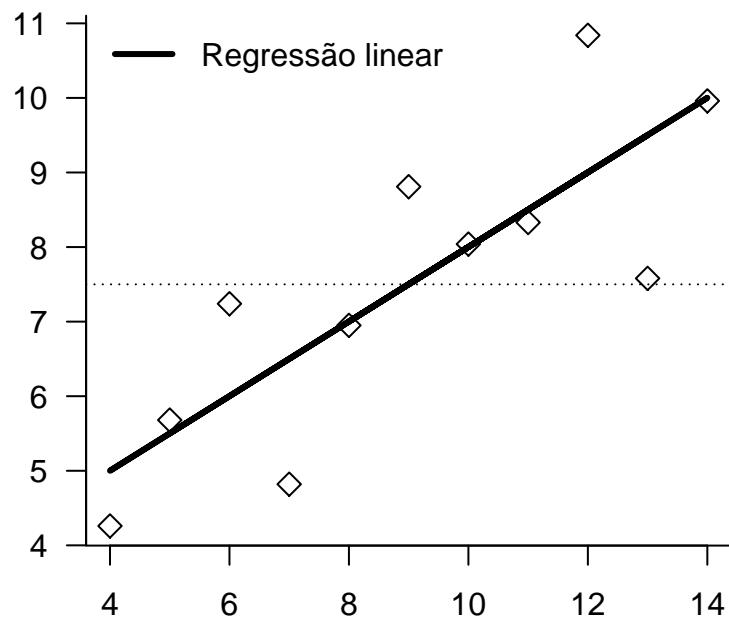
axis(side = 2,
     las = 1)

lines(x = dados$x1, y = ypredito,
      lty = 1,
      lwd = 3)

legend(x = "topleft",
       legend = "Regressão linear",
       lty = 1,
       lwd = 3,
       bty = "n")

abline(h = 7.5,
       lty = 3)

```



## 2.11 Adicionando setas

Função auxiliar *arrows* adiciona setas ao gráfico.

Argumentos:

**x0, y0, x1 e y1** - Coordenadas para o início e fim da seta. Sendo que  $x0$  e  $y0$  são para a origem da seta e  $x1$  e  $y1$  para o destino da seta. Pode ser um único valor ou um vetor para adicionar várias setas.

**length** - Alterar o comprimento da ponta da seta. Argumento numérico expresso em polegadas com padrão = 0.25.

```

plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",

```

```

xaxt = "n", yaxt = "n",
pty = "n")

points(x = dados$x1, y = dados$y1,
       pch = 23,
       cex = 1.3)

box(bty = "1")

axis(side = 1)

axis(side = 2,
     las = 1)

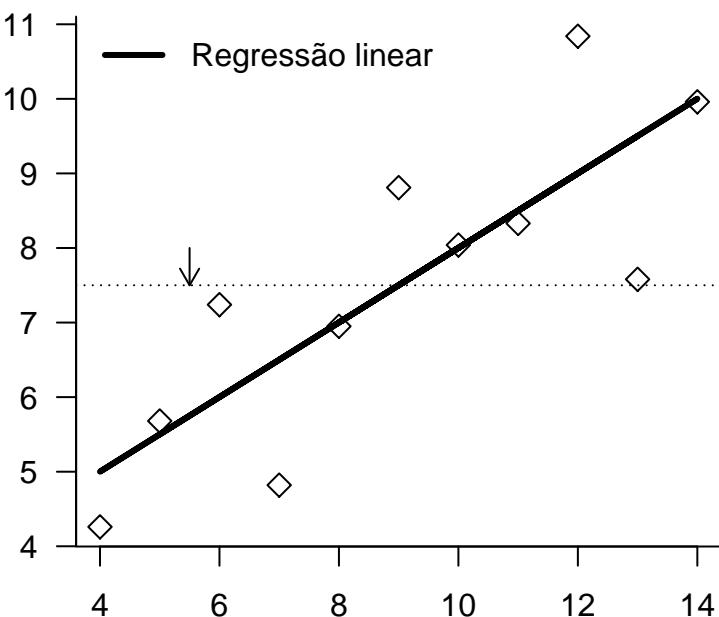
lines(x = dados$x1, y = ypredito,
      lty = 1,
      lwd = 3)

legend(x = "topleft",
       legend = "Regressão linear",
       lty = 1,
       lwd = 3,
       bty = "n")

abline(h = 7.5,
       lty = 3)

arrows(x0 = 5.5, y0 = 8,
       x1 = 5.5, y1 = 7.5,
       length = 0.1)

```



## 2.12 Adicionando textos

Função auxiliar *text* adiciona texto ao gráfico.

Argumentos:

**x** e **y** - Coordenadas para o texto. Pode ser um vetor com uma série de valores.

**labels** - Texto ou expressão para adicionar a coordenada. Pode ser um vetor com o texto de seré adicionado em cada coordenada fornecida.

```
plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")
```

```
points(x = dados$x1, y = dados$y1,
       pch = 23,
       cex = 1.3)
```

```
box(bty = "l")
```

```
axis(side = 1)
```

```
axis(side = 2,
     las = 1)
```

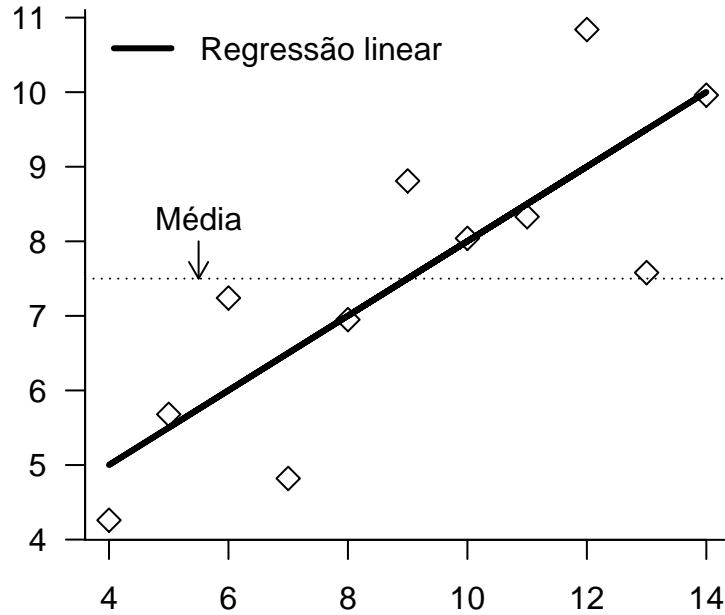
```
lines(x = dados$x1, y = ypredito,
      lty = 1,
      lwd = 3)
```

```
legend(x = "topleft",
       legend = "Regressão linear",
       lty = 1,
       lwd = 3,
       bty = "n")
```

```
abline(h = 7.5,
       lty = 3)
```

```
arrows(x0 = 5.5, y0 = 8,
       x1 = 5.5, y1 = 7.5,
       length = 0.1)
```

```
text(x = 5.5, y = 8.3,
     labels = "Média")
```



## 2.13 Adicionando segmentos

Função auxiliar *segments* adiciona pequenos segmentos ao gráfico.

Argumentos:

**x0, y0, x1 e y1** - Coordenadas para o início e fim do segmento. Sendo que *x0* e *y0* são para a origem e *x1* e *y1* para o fim do segmento. Pode ser um único valor ou um vetor para adicionar várias segmentos.

```
plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")

points(x = dados$x1, y = dados$y1,
       pch = 23,
       cex = 1.3)

box(bty = "l")

axis(side = 1)

axis(side = 2,
     las = 1)

lines(x = dados$x1, y = ypredito,
      lty = 1,
      lwd = 3)

legend(x = "topleft",
       legend = "Regressão linear",
       lty = 1,
       lwd = 3,
       bty = "n")

abline(h = 7.5,
```

```

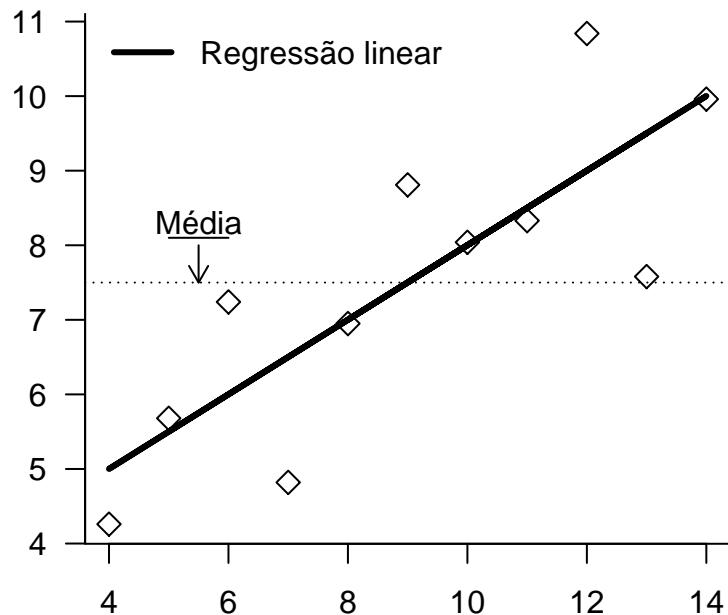
lty = 3)

arrows(x0 = 5.5, y0 = 8,
       x1 = 5.5, y1 = 7.5,
       length = 0.1)

text(x = 5.5, y = 8.3,
     labels = "Média")

segments(x0 = 5, y0 = 8.1,
         x1 = 6, y1 = 8.1)

```



## 2.14 Adicionando grades

Função auxiliar *grid* adiciona uma grade de linhas ao gráfico.

Argumentos:

**nx** e **ny** - Especificar o número de células do grid para as direções de *x* e *y* respectivamente. Se for *NULL* as linhas são mostradas nos marcadores de cada eixo.

```

plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")

points(x = dados$x1, y = dados$y1,
       pch = 23,
       cex = 1.3)

box(bty = "1")

axis(side = 1)

axis(side = 2,

```

```

las = 1)

lines(x = dados$x1, y = ypredito,
      lty = 1,
      lwd = 3)

legend(x = "topleft",
       legend = "Regressão linear",
       lty = 1,
       lwd = 3,
       bty = "n")

abline(h = 7.5,
       lty = 3)

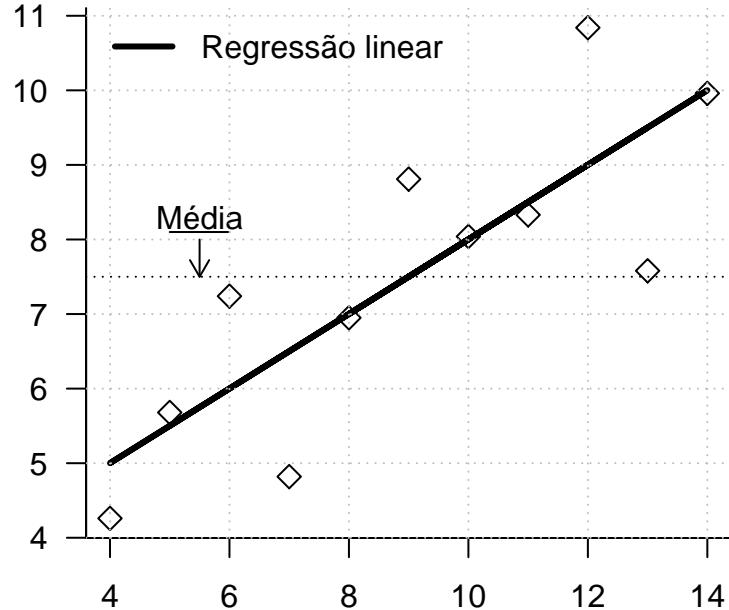
arrows(x0 = 5.5, y0 = 8,
        x1 = 5.5, y1 = 7.5,
        length = 0.1)

text(x = 5.5, y = 8.3,
     labels = "Média")

segments(x0 = 5, y0 = 8.1,
         x1 = 6, y1 = 8.1)

grid(nx = NULL, ny = NULL,
      lty = 3,
      lwd = 1,
      col = "gray")

```



## 2.15 Adicionando títulos

A função auxiliar *title* adiciona títulos ao gráfico e aos eixos.  
Argumentos:

**main, xlab e ylab** - Texto para o título do gráfico e para os rótulos dos eixos  $x$  e  $y$ , respectivamente.  
**font** - Especificar tipo de fonte. O tipo de fonte depende do dispositivo usado. De uma maneira básica 1 (fonte normal), 2 (fonte em negrito), 3 (fonte em itálico) e 4 (fonte em negrito e itálico).

```
plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")

points(x = dados$x1, y = dados$y1,
       pch = 23,
       cex = 1.3)

box(bty = "1")

axis(side = 1)

axis(side = 2,
     las = 1)

lines(x = dados$x1, y = ypredito,
      lty = 1,
      lwd = 3)

legend(x = "topleft",
       legend = "Regressão linear",
       lty = 1,
       lwd = 3,
       bty = "n")

abline(h = 7.5,
       lty = 3)

arrows(x0 = 5.5, y0 = 8,
       x1 = 5.5, y1 = 7.5,
       length = 0.1)

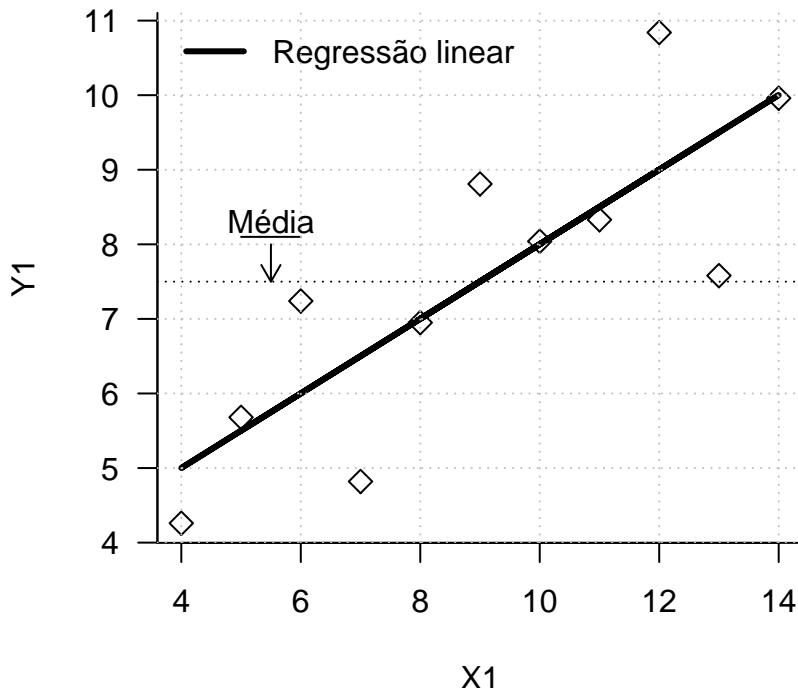
text(x = 5.5, y = 8.3,
     labels = "Média")

segments(x0 = 5, y0 = 8.1,
         x1 = 6, y1 = 8.1)

grid(nx = NULL, ny = NULL,
      lty = 3,
      lwd = 1,
      col = "gray")

title(main = "Anscombe",
      xlab = "X1", ylab = "Y1",
      font.main = 3)
```

## Anscombe



### 2.16 Adicionando polígonos

A função auxiliar *polygon* adiciona polígonos ao gráfico, desde figuras simples até contornos complexos.  
Argumentos:

**x, y** - Vetor com as coordenadas para cada vértice do polígono, sendo coordenadas para os eixos *x* e *y*, respectivamente.

**col** - Especificar cor para o preenchimento. No exemplo foi aletarado do transparência da cor, ver mais na seção cores abaixo.

**density** - Especificar densidade de linhas de sombreamento, em linhas por polegada. Se for *NULL* a figura será sólida.

**border** - Especificar cor para a linha da borda. Se *NA* a borda não é mostrada.

```
plot(dados$x1, dados$y1, type = "n",
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      bty = "n")

points(x = dados$x1, y = dados$y1,
       pch = 23,
       cex = 1.3)

box(bty = "1")

axis(side = 1)

axis(side = 2,
     las = 1)

lines(x = dados$x1, y = ypreditivo,
```

```

lty = 1,
lwd = 3)

legend(x = "topleft",
       legend = "Regressão linear",
       lty = 1,
       lwd = 3,
       bty = "n")

abline(h = 7.5,
       lty = 3)

arrows(x0 = 5.5, y0 = 8,
       x1 = 5.5, y1 = 7.5,
       length = 0.1)

text(x = 5.5, y = 8.3,
     labels = "Média")

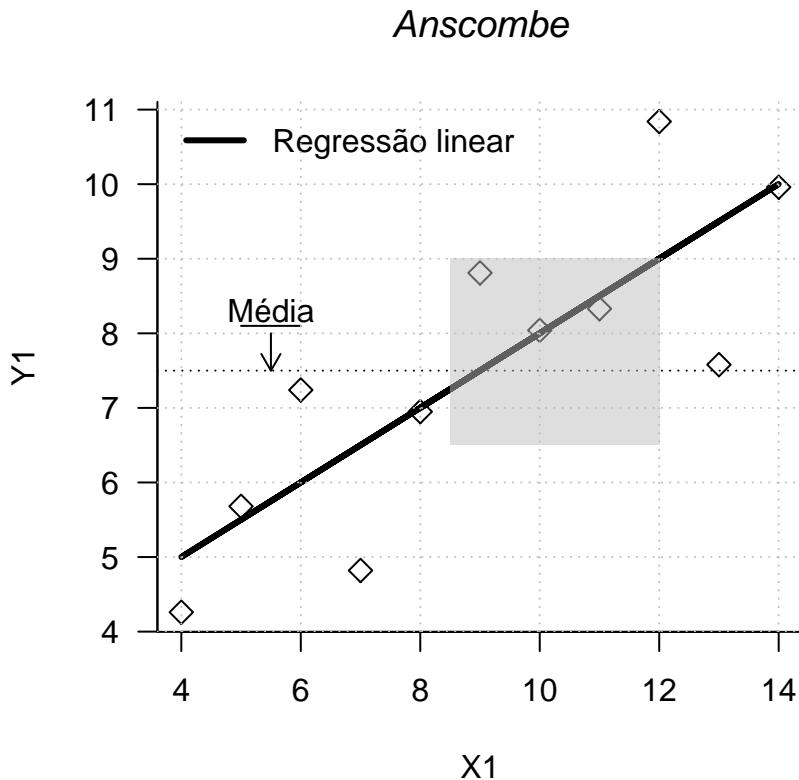
segments(x0 = 5, y0 = 8.1,
         x1 = 6, y1 = 8.1)

grid(nx = NULL, ny = NULL,
      lty = 3,
      lwd = 1,
      col = "gray")

title(main = "Anscombe",
      xlab = "X1", ylab = "Y1",
      font.main = 3)

polygon(x = c(8.5, 12, 12, 8.5), y = c(6.5, 6.5, 9, 9),
        col = adjustcolor("grey", alpha.f = 0.5),
        density = NULL,
        border = NA)

```



## 2.17 Função locator()

Muitas funções usadas para adicionar itens aos gráficos precisam da localização exata dentro do gráfico. Uma opção para obter a localização é usar a função *locator*. Executando a função *locator()* permitirá clicar na área do gráfico e obter as coordenadas dos pontos. A função fecha automaticamente quando clicado no botão direito do mouse ou especificando o argumento *n* com a quantidade de pontos que serão obtidos.

```
locator(n = 1, labels = "Cliquei aqui")
```

## 2.18 Expressões

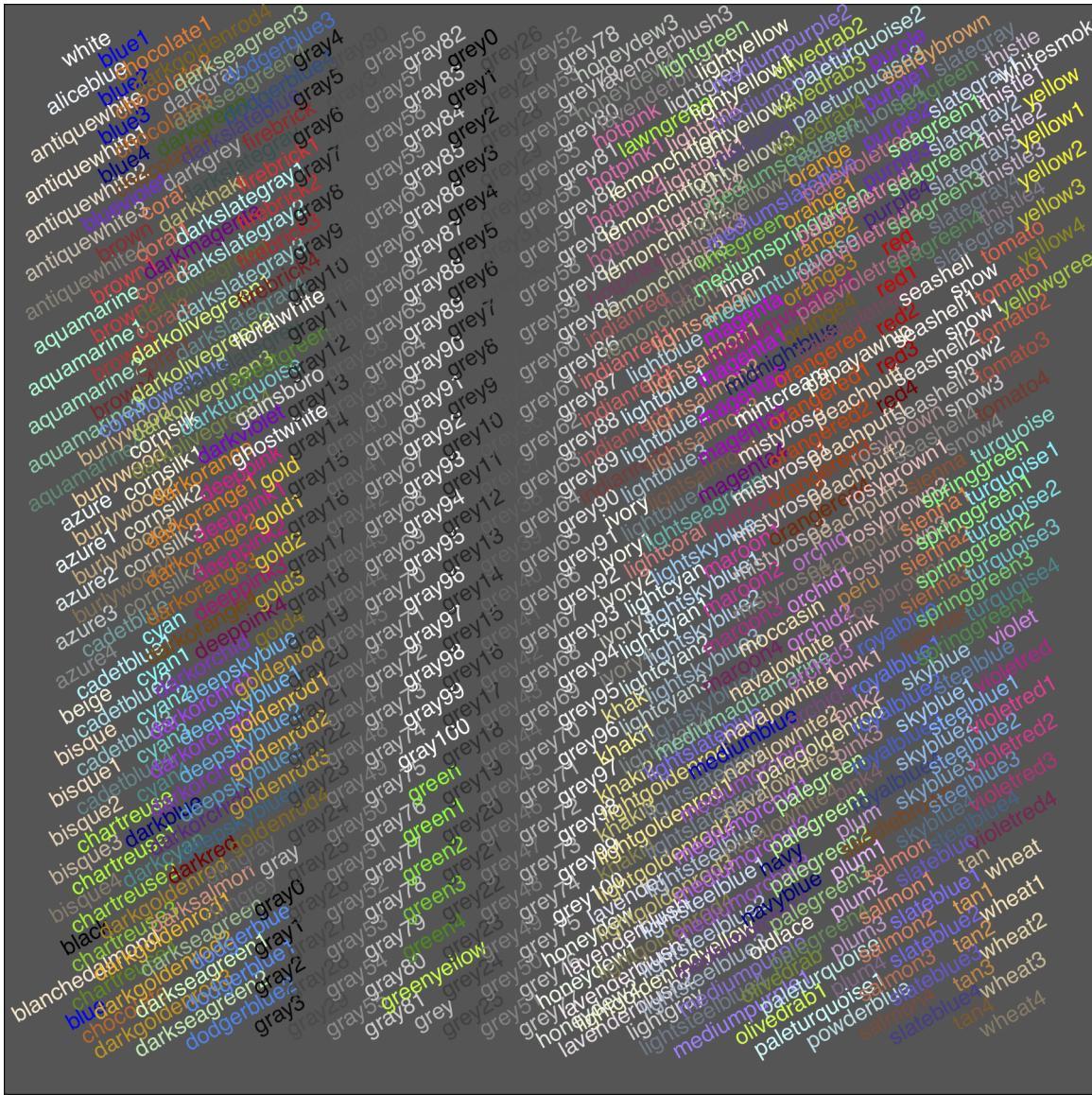
Elementos com formatação especial, notações matemáticas e letras gregas podem ser mostrados na forma de texto, títulos e eixos usando a função *expression* (ver ?*expression*). Mais informações podem ser encontradas na função *plotmath* (ver ?*plotmath*) e em *demonstração(plotmath)*. As figuras abaixo mostram alguns exemplos de expressões que podem ser usadas, as mesmas foram geradas a partir do exemplos de *demonstração(plotmath)*.

$x + y$	$X + Y$	$x \%subset% y$	$X \subset y$
$x - y$	$X - Y$	$x \%subseteq% y$	$X \subseteq y$
$x * y$	$Xy$	$x \%supset% y$	$X \supset y$
$x/y$	$X/y$	$x \%supseteqq% y$	$X \supseteq y$
$x \%+-% y$	$X \pm y$	$x \%notsubset% y$	$X \not\subset y$
$x\%/\%y$	$X \div y$	$x \%in% y$	$X \in y$
$x \%*\% y$	$X \times y$	$x \%notin% y$	$X \notin y$
$x \%.\% y$	$X \cdot y$	$\text{hat}(x)$	$\hat{X}$
$x[i]$	$X_i$	$\text{tilde}(x)$	$\tilde{X}$
$x^2$	$X^2$	$\text{ring}(x)$	$\overset{\circ}{X}$
$\text{paste}(x, y, z)$	$Xyz$	$\text{bar}(xy)$	$\overline{xy}$
$\text{sqrt}(x)$	$\sqrt{X}$	$\text{widehat}(xy)$	$\widehat{xy}$
$\text{sqrt}(x, y)$	$\sqrt[y]{X}$	$\text{widetilde}(xy)$	$\widetilde{xy}$
$\text{list}(x, y, z)$	$X, y, Z$	$x \%<->% y$	$X \leftrightarrow y$
$x == y$	$X = Y$	$x \%->% y$	$X \rightarrow y$
$x != y$	$X \neq Y$	$x \%<% y$	$X \leftarrow y$
$x < y$	$X < Y$	$x \%up% y$	$X \uparrow y$
$x <= y$	$X \leq Y$	$x \%down% y$	$X \downarrow y$
$x \%~~% y$	$X \approx Y$	$x \%<=>% y$	$X \Leftrightarrow y$
$x \%=~~% y$	$X \cong Y$	$x \%=>% y$	$X \Rightarrow y$
$x \%==% y$	$X \equiv Y$	$x \%<=% y$	$X \Leftarrow y$
$x \%prop% y$	$X \propto Y$	$x \%dblup% y$	$X \uparrow\downarrow y$
$x \%~% y$	$X \sim Y$	$x \%dbldown% y$	$X \downarrow\uparrow y$
$\text{plain}(x)$	$X$	$\text{Alpha} - \text{Omega}$	$\Lambda - \Omega$
$\text{italic}(x)$	$X$	$\text{alpha} - \text{omega}$	$\alpha - \omega$
$\text{bold}(x)$	$\mathbf{X}$	$\text{phi1} + \text{sigma1}$	$\varphi + \varsigma$
$\text{bolditalic}(x)$	$\mathbf{X}$	$\text{Upsilon1}$	$\Upsilon$
$\text{underline}(x)$	$\underline{X}$	$\text{infinity}$	$\infty$
$\text{list}(x[1], \dots, x[n])$	$X_1, \dots, X_n$	$32 * \text{degree}$	$32^\circ$
$x[1] + \dots + x[n]$	$X_1 + \dots + X_n$	$60 * \text{minute}$	$60'$
$\text{list}(x[1], \text{cdots}, x[n])$	$X_1, \dots, X_n$	$30 * \text{second}$	$30''$
$x[1] + \text{ldots} + x[n]$	$X_1 + \dots + X_n$	$x \sim \sim y$	$X \sim y$

<code>frac(x, y)</code>	$\frac{x}{y}$	<code>min(g(x), x &gt;= 0)</code>	$\min_{x \geq 0} g(x)$
<code>over(x, y)</code>	$\frac{x}{y}$	<code>inf(S)</code>	$\inf S$
<code>atop(x, y)</code>	$\frac{x}{y}$	<code>sup(S)</code>	$\sup S$
<code>sum(x[i], i = 1, n)</code>	$\sum_1^n x_i$	$(x + y) * z$	$(x + y)z$
<code>prod(plain(P)(X == x), x)</code>	$\prod_X P(X = x)$	$x^y + z$	$x^y + z$
<code>integral(f(x) * dx, a, b)</code>	$\int_a^b f(x)dx$	$x^{y+z}$	$x^{(y+z)}$
<code>union(A[i], i == 1, n)</code>	$\bigcup_{i=1}^n A_i$	$x^{\{y+z\}}$	$x^{y+z}$
<code>intersect(A[i], i == 1, n)</code>	$\bigcap_{i=1}^n A_i$	<code>bgroup(" ", atop(x, y), ")")</code>	$\binom{x}{y}$
<code>lim(f(x), x %&gt;% 0)</code>	$\lim_{x \rightarrow 0} f(x)$	<code>group("!", x, "!"")</code>	$ x $

## 2.19 Cores

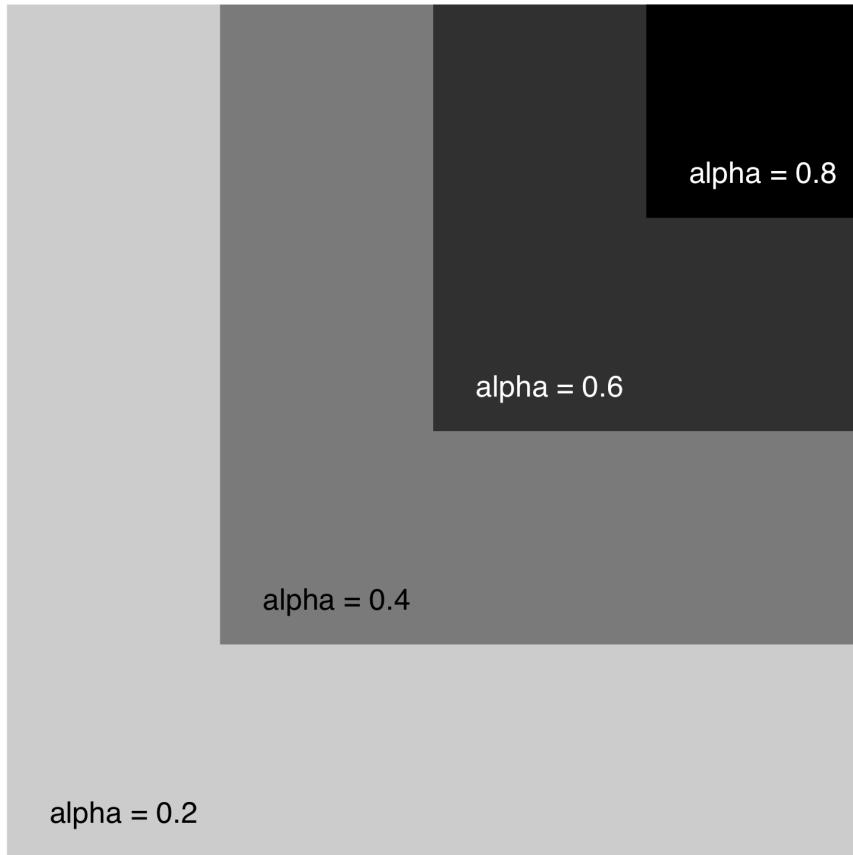
As cores de cada um dos itens podem ser alteradas pelo argumento `col`. As cores podem ser alteradas pelo nome da cor (por exemplo `col = "black"`) ou por outras codificações. Cores básicas para a maioria dos gráficos são 'black', 'white' e 'grey', mas mais informações sobre podem ser buscadas na ajuda de `?colors` e `demo(colors)`. Ao fazer gráficos coloridos certifique que as cores sejam apropriadas para daltônicos. As figuras abaixo mostram alguns exemplos com nomes das cores, as mesmas foram geradas a partir do exemplos de `demo(colors)`.



Existem algumas paletas de cores prontas no R. A paleta básica pode ser encontrada com a função `palette()`. Essa paleta é composta por apenas 8 cores (“black”, “red”, “green3”, “blue”, “cyan”, “magenta”, “yellow” e “gray”), mas essa pode ser alterada permitindo adicional, remover ou personalizar novas cores na paleta básica. Os pacotes `fields` e `RColorBrewer` oferecem algumas funções interessantes para criar paletas. Essas paletas podem ser expandidas usando a função `colorRampPalette`. A figura a seguir mostra algumas opções de comandos que podem ser utilizados para gerar paletas de cores.

	Cores	Adequado para daltônicos
palette('default')		F
cm.colors(12)		T
topo.colors(12)		F
terrain.colors(12)		F
heat.colors(12)		T
tim.colors(12)		F
rainbow(12)		F
two.colors(12, start = 'red', end = 'blue', middle = 'white')		T
colorRampPalette(brewer.pal(8, 'Greys'))(12)		T
brewer.pal(9, 'YlOrRd')		T
brewer.pal(9, 'YlOrBr')		T
brewer.pal(9, 'YlGnBu')		T
brewer.pal(9, 'YlGn')		T
brewer.pal(9, 'Reds')		T
brewer.pal(9, 'RdPu')		T
brewer.pal(9, 'Purples')		T
brewer.pal(9, 'PuRd')		T
brewer.pal(9, 'PuBuGn')		T
brewer.pal(9, 'PuBu')		T
brewer.pal(9, 'OrRd')		T
brewer.pal(9, 'Oranges')		T
brewer.pal(9, 'Greys')		T
brewer.pal(9, 'Greens')		T
brewer.pal(9, 'GnBu')		T
brewer.pal(9, 'BuPu')		T
brewer.pal(9, 'BuGn')		T
brewer.pal(9, 'Blues')		T
brewer.pal(12, 'Set3')		F
brewer.pal(8, 'Set2')		T
brewer.pal(9, 'Set1')		F
brewer.pal(8, 'Pastel2')		F
brewer.pal(9, 'Pastel1')		F
brewer.pal(12, 'Paired')		T
brewer.pal(8, 'Dark2')		T
brewer.pal(8, 'Accent')		F
brewer.pal(11, 'Spectral')		F
brewer.pal(11, 'RdYIGn')		F
brewer.pal(11, 'RdYlBu')		T
brewer.pal(11, 'RdGy')		F
brewer.pal(11, 'RdBu')		T
brewer.pal(11, 'PuOr')		T
brewer.pal(11, 'PRGn')		T
brewer.pal(11, 'PiYG')		T
brewer.pal(11, 'BrBG')		T

A opacidade das cores podem ser ajustadas usando a função *adjustcolor*, principalmente alterando o argumento *alpha*. Na figura a seguir foi gerada apenas com a cor “black” com diferentes valores dos parâmetros *alpha*. Valores de *alpha* próximos a 0 deixam as cores mais opacas (mais transparências) enquanto que valores próximos a 1 deixam as cores mais sólidas. Gráficos feitos com essas opções permitem sobrepor informações em uma mesma figura.



## 2.20 Fontes

Cada dispositivo gráfico do R possui um conjunto próprio de fontes. A lista de fontes pode ser acessada pelas funções: `quartzFonts()` para OS X, `X11Fonts()` para Linux, `windowsFonts()` para Windows, `pdfFonts()` para pdf e `postscriptFonts()` para postscript. As fontes podem ser alteradas pelo argumentos `family` e `font`.

## 2.21 Reciclagem

O R trabalha com o conceito de reciclagem também nos parâmetros gráficos. Quando um vetor de comprimento  $n$  é esperado (sequência de valores), mas é forcecido um único valor ou vetor de comprimento menor que  $n$  esses valores são reciclados até os valores fornecidos preencherem o vetor de tamanho  $n$ . No exemplo abaixo os dados são três variáveis e 10 observações. As variáveis numéricas  $x$  e  $y$  são duas sequências de 1 a 10 representando as respectivas coordenadas dos pontos no gráfico e a variável `fator` categoriza as observações em duas classes.

```

dados <- data.frame(x = 1:10, y = 1:10, fator = rep(c("A", "B"), each = 5))
str(dados)
'data.frame': 10 obs. of 3 variables:
 $ x : int 1 2 3 4 5 6 7 8 9 10
 $ y : int 1 2 3 4 5 6 7 8 9 10
 $ fator: Factor w/ 2 levels "A","B": 1 1 1 1 1 2 2 2 2 2

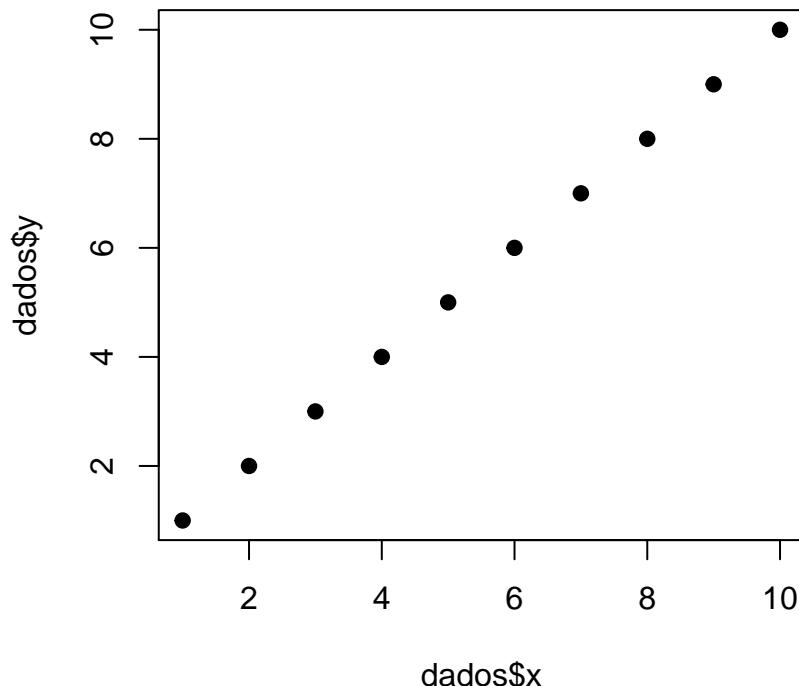
```

Ao fazer um gráfico especificando apenas o tipo de ponto e cor dos pontos essas especificações são usadas para todos os pontos do gráfico.

```

plot(dados$x, dados$y,
      pch = 19,
      col = "black")

```

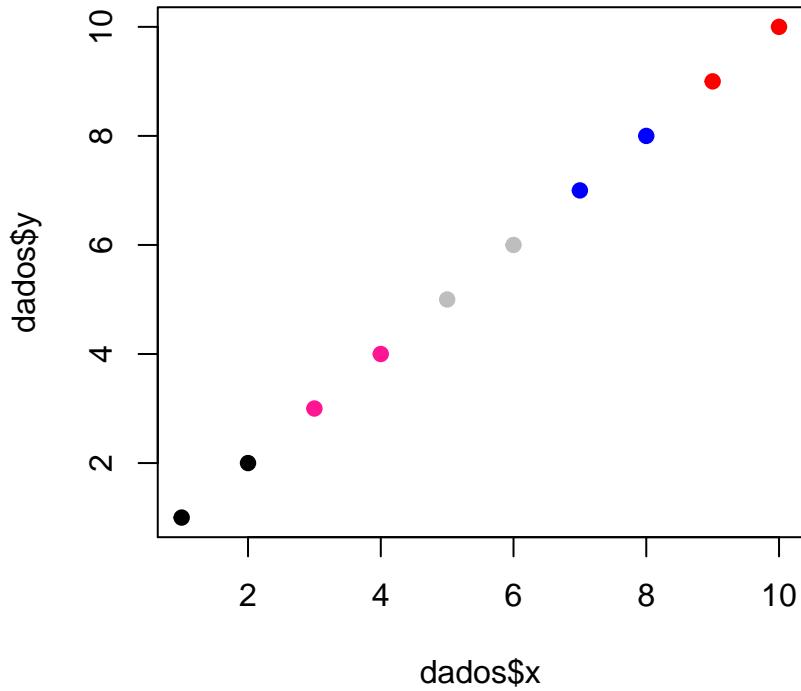


Para especificar cores para cada ponto uma opção seria fornecer 10 cores, uma para cada ponto.

```

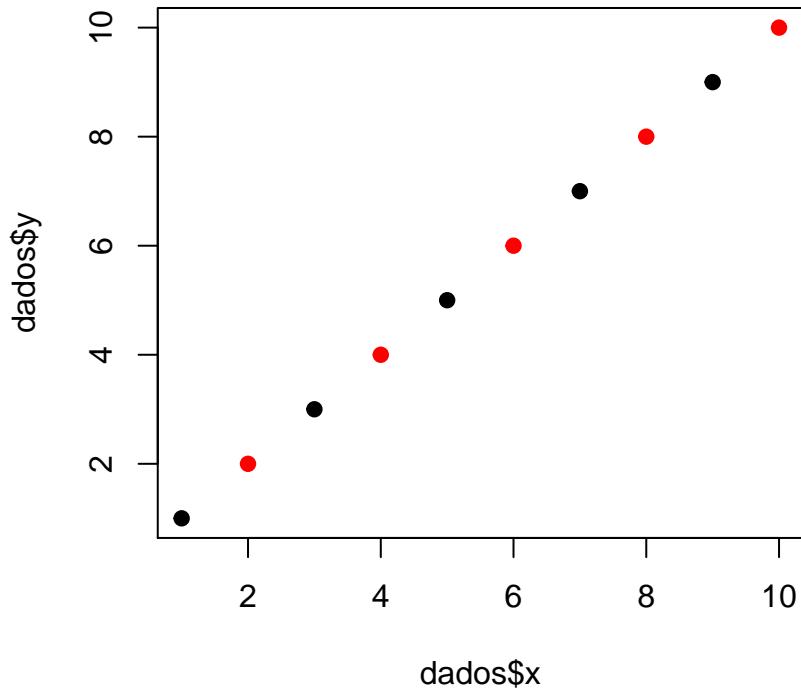
plot(dados$x, dados$y,
      pch = 19,
      col = c("black", "black",
             "deeppink", "deeppink",
             "grey", "grey",
             "blue", "blue",
             "red", "red"))

```



Se apenas duas cores são fornecidas, no exemplo “black” e “red”, a cor será repetida para os demais pontos. Nesse exemplo o R esperava 10 nomes de cores, mas só foram fornecidas duas. Como resultado as cores foram reclinadas até comprimento esperado.

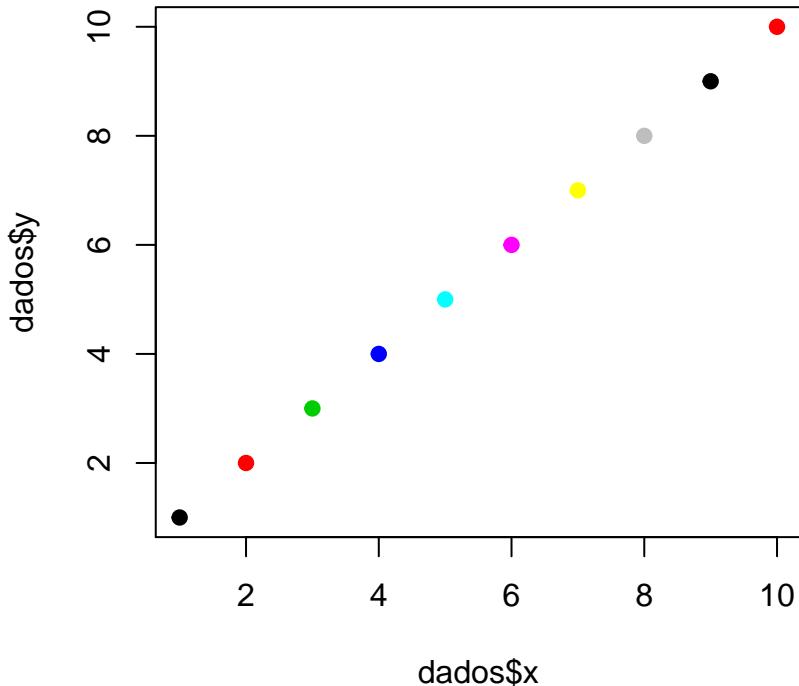
```
plot(dados$x, dados$y,
      pch = 19,
      col = c("black", "red"))
```



Os parâmetros gráficos podem ser especificados usando outras variáveis. No exemplo abaixo as cores foram especificadas usando como base a variável  $y$ . As cores resultantes foram obtidas usando a paleta básica

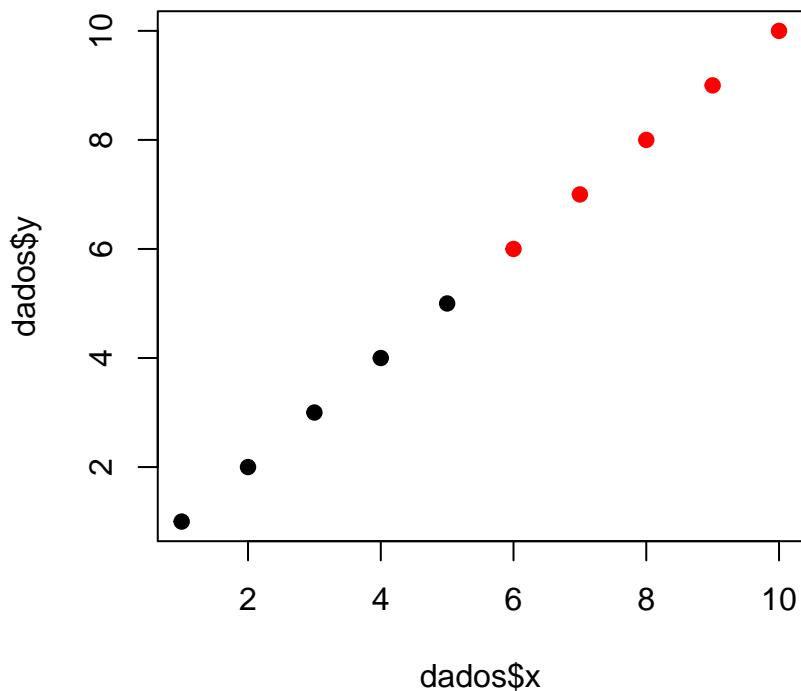
de cores. Por padrão a paleta básica é composta por apenas 8 cores (ver seção Cores acima), no exemplo as cores preta e vermelha (as primeiras da paleta) foram repetida nos últimos dois pontos.

```
plot(dados$x, dados$y,  
      pch = 19,  
      col = dados$y)
```



Ainda uma variável categoria poderia ser usada para definir as cores. No exemplo a variável *fator* foi usada para atribuir as cores, sendo os pontos da categoria *A* foram mostrados como pretos e da categoria *B* como vermelhos (novamente as primeiras cores da paleta básica).

```
plot(dados$x, dados$y,  
      pch = 19,  
      col = dados$fator)
```



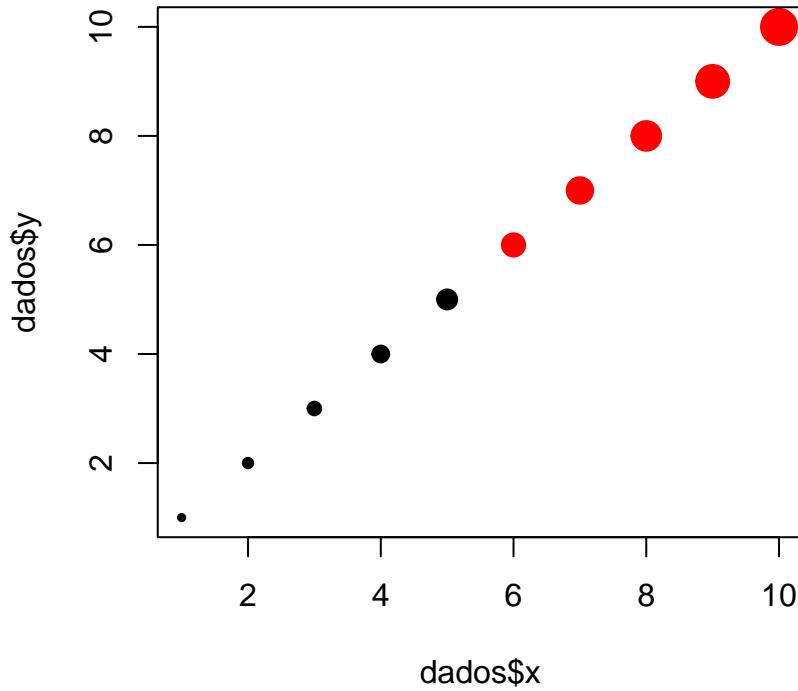
Os argumentos gráficos (*pch*, *col*, *lwd*, *cex*, entre outros) podem ser especificados com ajuda da funções básicas como *rep*, *seq*, *factor* e *ifelse*. A função *rescale* do pacote *scales* permite rescalar uma variável para uma novo intervalo. Essa função pode ser útil para atribuir tamanhos dos pontos em função de uma variável.

Argumentos:

**to** - Um vetor número como o mínimo e máximo do novo intervalo para a variável.

```
require(scales)
Loading required package: scales

plot(dados$x, dados$y,
      pch = 19,
      col = dados$fator,
      cex = scales::rescale(dados$x,
                            to = c(0.5, 2.5)))
```



### 3 Princípios do design eficaz de um gráfico

A comunicação visual efetiva de informações facilita a compreensão e tomada de decisões, por essa razão é importante levar em consideração alguns pontos para melhorar o design nos gráficos. Primeiro passo é identificar a finalidade do gráfico e as informações essenciais para apoiar os objetivos do gráfico. Além disso, é preciso considerar o público-alvo e adaptar as possíveis restrições de espaço e formatação. É importante ressaltar que criar um gráfico é um processo iterativo envolvendo etapas de produção, revisão e refinamento.

Os seguintes princípios foram extraídos de Vandemeulebroecke *et al.* 2018.

- **Proximidade** - agrupe elementos relacionados.
- **Alinhamento** - elementos no mesmo plano vertical ou horizontal são percebidos como tendo propriedades semelhantes.
- **Simplicidade** - corte qualquer coisa supérflua, inclua apenas elementos que agreguem valor, limite a 2-3 cores ou fontes.
- **Espaço vazio** - use espaço em branco para minimizar a distração e fornecer clareza.
- **Legibilidade** - fontes como *sans* e *serif* são mais fáceis de ler, use cores para dar ênfase em vez de um novo tipo de letra.
- **Cor** - selecione cores que apresentem contraste suficiente para tornar o gráfico legível. Escolha esquemas de cores monocromáticas para evitar confrontos. Use cores escuras e cores acentuadas para enfatizar informações importantes. Considere usar cores adaptadas para daltónicos.
- **Hierarquia visual** - use cores, fontes, tamanho da imagem, tipo de letra, alinhamento e posicionamento para criar uma ordem de visualização.
- **Pontos Focais** - tente organizar uma área principal no gráfico para atrair imediatamente os olhos, enfatizar o conceito mais importante em torno de um ponto focal. Use cores contrastantes para chamar a atenção.
- **Repetição** - elementos repetitivos podem ser visualmente atraentes, formas repetidas, rótulos, cores.
- **Familiaridade** - usando estilos familiares, ícones, estrutura de navegação faz com que os espectadores se sintam confiantes.
- **Consistência** - seja consistente com tamanhos de títulos, opções de fontes, esquema de cores e espaçamento. Use imagens com estilos semelhantes.

## 4 Gráficos personalizados

Nesta seção o objetivo é mostrar alguns tipos de gráficos simples que podem ser construídos a partir do R. Nem todos os elementos gráficos são aplicados em todos os gráficos, sendo que são mostradas apenas as opções mais comuns usadas em cada tipo de gráfico.

### 4.1 Gráfico de barras

#### 4.1.0.1 Dados

Dados disponíveis no pacote *ade4* com o nome de *tortues*. Os dados são 48 observações de tartarugas descritas por quatro variáveis (gênero, comprimento, largura e altura). Para cada gênero pode-se calcular média e desvio padrão das observações.

```
require(ade4)
Loading required package: ade4
data(tortues)
str(tortues)
'data.frame': 48 obs. of 4 variables:
 $ long: num 93 94 96 101 102 103 104 106 107 112 ...
 $ larg: num 74 78 80 84 85 81 83 83 82 89 ...
 $ haut: num 37 35 35 39 38 37 39 39 38 40 ...
 $ sexe: Factor w/ 2 levels "M","F": 1 1 1 1 1 1 1 1 1 1 ...

dados <- rbind(tapply(tortues[, 1], tortues[, 4], mean),
               tapply(tortues[, 2], tortues[, 4], mean),
               tapply(tortues[, 3], tortues[, 4], mean))
rownames(dados) <- names(tortues)[1:3]
dados # Média por gênero
      M          F
long 113.37500 136.0000
larg  88.29167 102.7083
haut 113.37500 136.0000

dados_sd <- rbind(tapply(tortues[, 1], tortues[, 4], sd),
                   tapply(tortues[, 2], tortues[, 4], sd),
                   tapply(tortues[, 3], tortues[, 4], sd))
rownames(dados_sd) <- names(tortues)[1:3]
dados_sd # Desvio padrão por gênero
      M          F
long 11.779911 21.245971
larg  7.074013 13.113150
haut  3.352481  8.464654
```

#### 4.1.0.2 Plot

A função *barplot* é específica para desenhar gráficos de barras horizontais e verticais. Neste caso dos dados são uma pequena tabela com duas colunas e três linhas. Pode-se construir um gráfico mostrando seis barras com as medidas, agrupadas pelo gênero (pelos colunas). Neste gráfico as cores são definidas de maneira automática, mostradas em tons de cinza por padrão.

Argumentos:

**width** - Especificar largura das barras, com padrão de 1.

**beside** - Argumento lógico para especificar se colunas devem ser mostradas lado a lado.

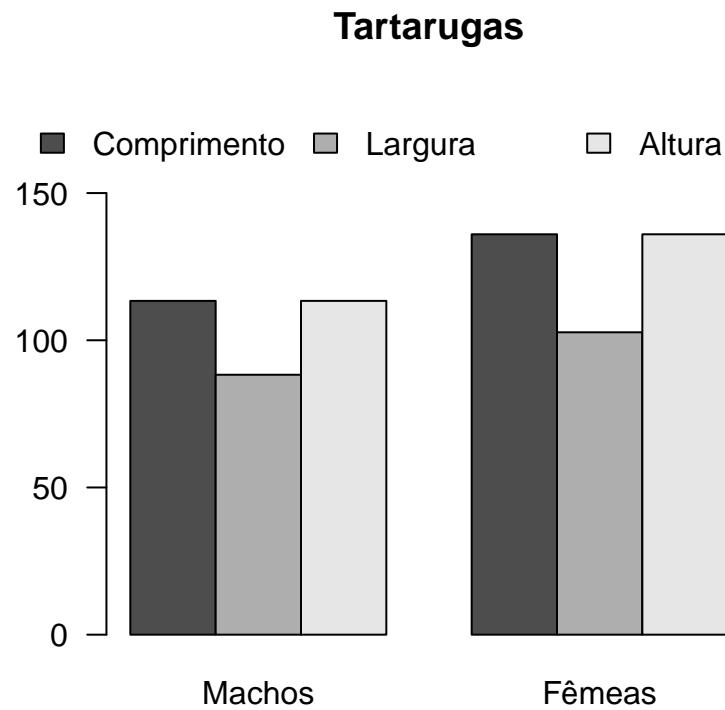
**legend** - Especificar o texto de cada nível no gráfico.

**args.legend** - Especificar argumentos que serão usados na construção da legenda.

**x** - Parte de args.legend usado para especificar posição da legenda.

- bty** - Parte de args.legend usado para especificar tipo de borda da legenda.
- ncol** - Parte de args.legend usado para especificar número de colunas para os argumentos da legenda.
- names** - Especificar nomes usados em cada grupo de barras.

```
barplot(dados,
        beside = TRUE,
        width = 0.7,
        ylim = c(0, 180),
        las = 1,
        legend = c("Comprimento", "Largura", "Altura"),
        args.legend = list(x = "top",
                            bty = "n",
                            ncol = 3),
        names = c("Machos", "Fêmeas"),
        main = "Tartarugas")
```



Para adicionar os linhas com o desvio padrão pode-se usar a função *arrows* (para setas). Uma alternativa é usar a função *plotCI* do pacote *plotrix*. Essa função permite adicionar barras de erros tanto na horizontal quanto na vertical. Um pequeno truque no caso do gráfico de barras e salvar os comandos do primeiro gráfico em um objeto qualquer (Neste caso objeto *xx*). O objeto terá a posição no eixo *x* de cada barra. A posição no eixo *y* são os próprios dados usados para construir o gráfico. A altura de cada barra de erro foi calculada pelo desvio padrão de cada grupo.

Argumentos:

**x** e **y** - Coordenadas para o início da cada linha de erro.

**uiw** e **liw** - Comprimento da linha de erro para a parte superior e inferior, respectivamente.

**add** - Argumento lógico para especificar se linhas de erro devem ser adicionadas ao gráfico existente ou não. Padrão é FALSE.

**pch** - Tipo de símbolo usado no centro da barra de erro. Se for NA não é mostrado nenhum símbolo.

```
require(plotrix)
Loading required package: plotrix
```

```

Attaching package: 'plotrix'
The following object is masked from 'package:scales':
    rescale

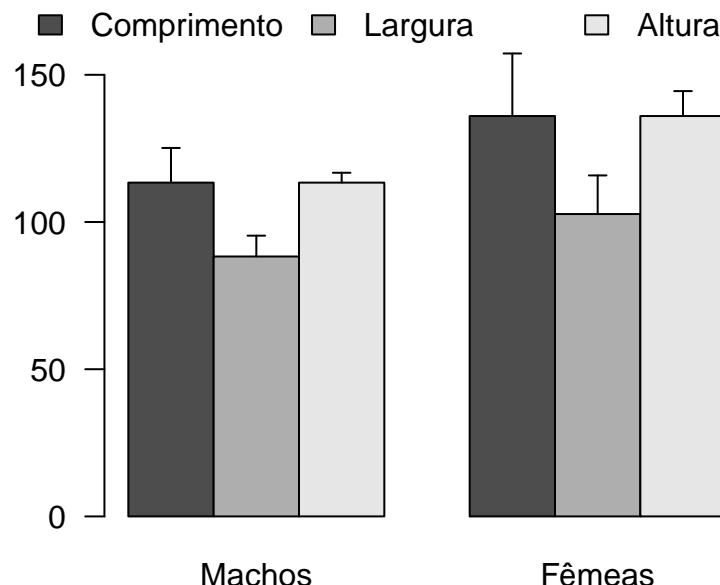
xx <- barplot(dados, beside = TRUE,
               width = 0.7,
               ylim = c(0, 180),
               las = 1,
               legend = c("Comprimento", "Largura", "Altura"),
               args.legend = list(x = "top",
                                  bty = "n",
                                  ncol = 3),
               names = c("Machos", "Fêmeas"),
               main = "Tartarugas",
               xaxs = "r")

xx
[,1] [,2]
[1,] 1.05 3.85
[2,] 1.75 4.55
[3,] 2.45 5.25

plotCI(x = xx, y = dados,
        uiw = dados_sd,
        liw = 0,
        add = TRUE,
        pch = NA)

```

**Tartarugas**



## 4.2 Gráfico de barras empilhadas

### 4.2.0.1 Dados

Dados sobre a população brasileira na década de 2010. Os valores são em proporções para 4 faixas etárias arbitrárias. Extraído do United Nations, Department of Economic and Social Affairs.

```
dados <- data.frame(Mulheres = c(0.328, 0.461, 0.159, 0.052),
                     Homens = c(0.351, 0.464, 0.146, 0.039))
rownames(dados) <- c("0-19", "20-49", "50-69", "70+")
dados <- as.matrix(dados)
dados
  Mulheres Homens
0-19      0.328  0.351
20-49     0.461  0.464
50-69     0.159  0.146
70+       0.052  0.039
```

#### 4.2.0.2 Plot

A função *barplot* é específica para usar em gráficos de barras horizontais e verticais. Neste caso dos dados são uma pequena matriz (classe *matrix*) com duas colunas e quatro linhas. Pode-se mostrar uma barra para cada grupo (coluna) com as informações das linhas embilhadas umas sobre as outras. Neste gráfico as cores serão em tons de cinza para gerar quatro padrões de preenchimento das barras. Além disso, uma legenda à direita das barras será adicionada, bem como uma linha no eixo x na altura zero.

Argumentos:

**col** - Especificar cores usadas no gráfico.

**density** - Especificar a densidade das linhas de sombreado para as barras. Valores em linhas por polegada.

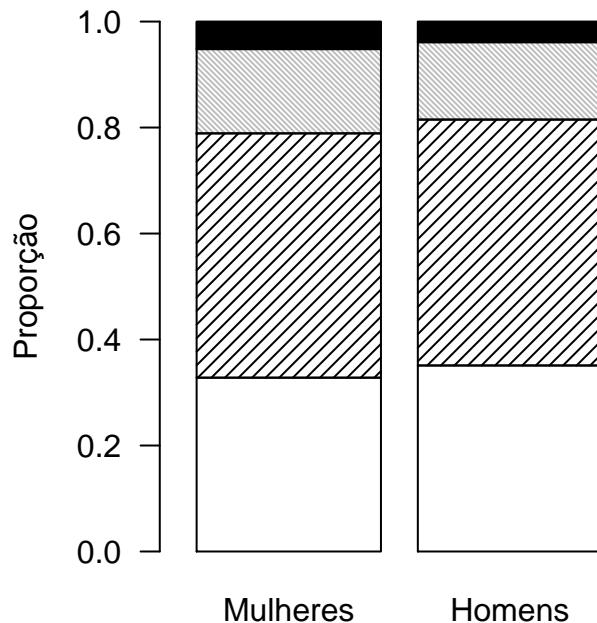
**angle** - Especificar ângulo da direção das linhas de sombreamento.

**xlim** e **ylim** - Especificar limites para os eixos x e y respectivamente.

**xaxs** - Especificar métodos para calcular os limites dos eixos.

```
barplot(dados,
         col = c("white", "black", "grey", "black"),
         density = c(0, 20, 50, 200),
         angle = c(0, 45, 135, 0),
         ylab = "Proporção",
         las = 1,
         xaxs = "i",
         main = "População Brasileira (2010)",
         ylim = c(0, 1), xlim = c(0, 3.5))
```

## População Brasileira (2010)



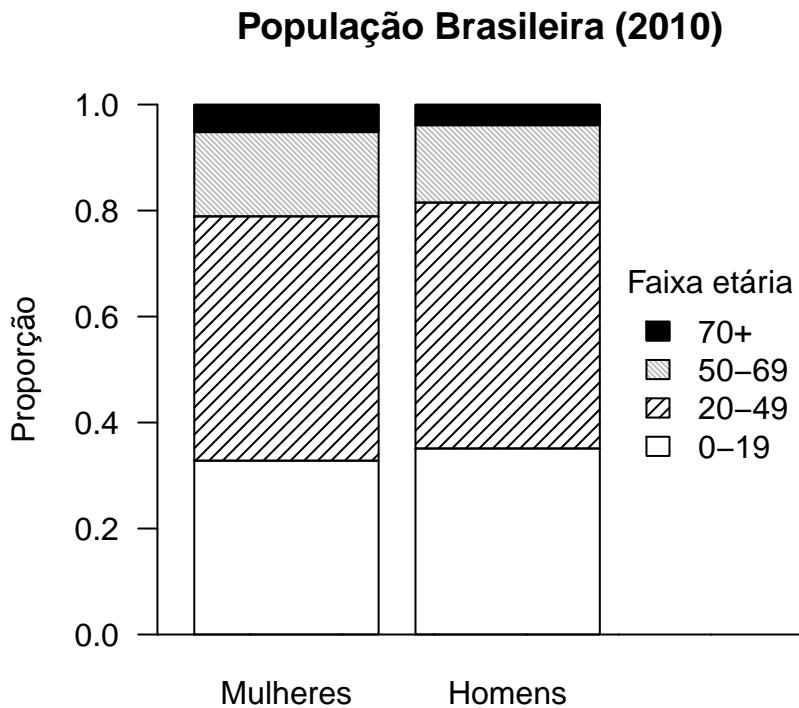
Para adicionar legenda a opção mais fácil é especificar diretamente na função *barplot*. Para adicionar uma linha no limite para o eixo x na altura do zero pode-se usar a função *axis*. Neste caso é importante que os limites dos eixos sejam calculados exatamente como o especificado pelos limites.

Argumentos:

- legend** - Especificar nomes para os itens da legenda.
- args.legend** - Especificar argumentos adicionais para a legenda.
- x** - Parte de args.legend usado para especificar posição da legenda.
- bty** - Parte de args.legend usado para especificar tipo de borda da legenda.
- title** - Parte de args.legend usado para especificar título da legenda.
- side** - Especificar posição do eixo.
- label** - Especificar nomes para os separadores do eixo.
- tck** - Especificar tamanho dos marcadores do eixo.

```
barplot(dados,
         col = c("white", "black", "grey", "black"),
         density = c(0, 20, 50, 200),
         angle = c(0, 45, 135, 0),
         ylab = "Proporção",
         las = 1,
         xaxs = "i",
         main = "População Brasileira (2010)",
         ylim = c(0, 1), xlim = c(0, 3.5),
         legend = rownames(dados),
         args.legend = list(x = "right",
                            bty = "n",
                            title = "Faixa etária"))

axis(side = 1, label = FALSE, tck = 0)
```



### 4.3 Gráfico de pontos com linha de regressão

#### 4.3.0.1 Dados

Dados *anscombe* são um conjunto usados como exemplo em regressões lineares. São 4 pares de variáveis, independente e dependente, com 11 observações.

```

data(anscombe)
dados <- anscombe
str(dados)
'data.frame':   11 obs. of  8 variables:
 $ x1: num  10 8 13 9 11 14 6 4 12 7 ...
 $ x2: num  10 8 13 9 11 14 6 4 12 7 ...
 $ x3: num  10 8 13 9 11 14 6 4 12 7 ...
 $ x4: num  8 8 8 8 8 8 19 8 8 ...
 $ y1: num  8.04 6.95 7.58 8.81 8.33 ...
 $ y2: num  9.14 8.14 8.74 8.77 9.26 8.1 6.13 3.1 9.13 7.26 ...
 $ y3: num  7.46 6.77 12.74 7.11 7.81 ...
 $ y4: num  6.58 5.76 7.71 8.84 8.47 7.04 5.25 12.5 5.56 7.91 ...

mod <- lm(y1 ~ x1, data = dados) # regressão linear
mod

Call:
lm(formula = y1 ~ x1, data = dados)

Coefficients:
(Intercept)          x1
            3.0001      0.5001

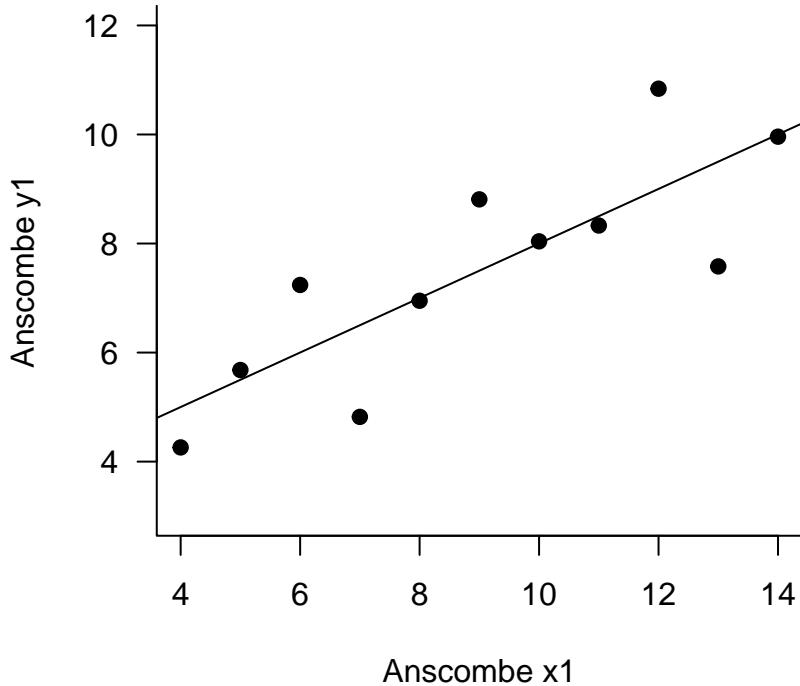
```

#### 4.3.0.2 Plot

Gráfico de pontos com linha de regressão linear e o intervalo de confiança. A opção mais simples de mostrar a linha do modelo linear é usar a função *abline*.

```
plot(dados$x1, dados$y1,
      xlab = "Anscombe x1", ylab = "Anscombe y1",
      las = 1,
      ylim = c(3, 12),
      pch = 19,
      bty = "l")

abline(mod)
```



Para restringir a linha no intervalo da variável preditora (*x1*) não é possível usar a função *abline*. Para fazer isso usa-se a função *lines* com os valores preditos pelo modelo linear. Primeiro é preciso obter os valores ajustados para o modelo que foi estimado. No caso da regressão linear simples a função *predict* usa o modelo para prever os valores para um novo intervalo da variável preditora. Uma exigência para a função *lines* funcionar corretamente é ordenar os dados na sequência da variável preditora. A função *lines* simplesmente conecta os pontos na sequência que são fornecidos, se a sequencia não estiver em ordem, crescente ou descrecente, a linha não ficará como o esperado.

```
# Primeiro passo é gerar um novo intervalo para a variável preditora
# Nesse caso a nova variável são 100 pontos, do limite mínimo e máximo da variável original
new <- data.frame(x1 = seq(min(dados$x1), max(dados$x1), length.out = 100))
fitted <- predict(mod, new, interval = "confidence") # Valores preditos para o novo intervalo
head(fitted) # Primeiras 6 linhas da tabela
  fit      lwr      upr
1 5.000455 3.422513 6.578396
2 5.050969 3.495730 6.606207
3 5.101483 3.568810 6.634156
4 5.151997 3.641747 6.662248
5 5.202511 3.714534 6.690489
6 5.253026 3.787163 6.718888
```

```

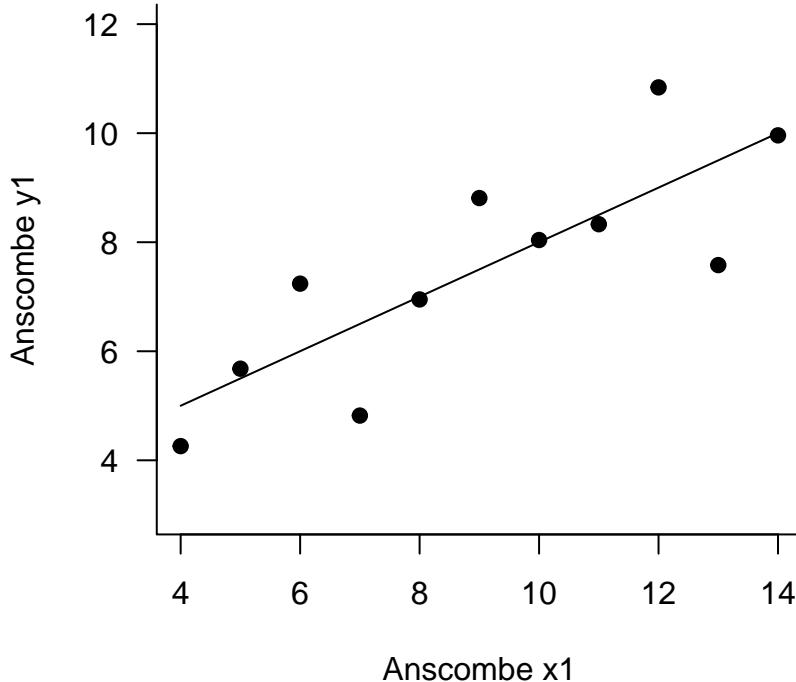
plot(dados$x1, dados$y1,
      xlab = "Anscombe x1", ylab = "Anscombe y1",
      las = 1,
      ylim = c(3, 12),
      pch = 19,
      bty = "l")

```

```

lines(new$x1, fitted[, "fit"], lty = 1)

```



Para mostrar os intervalos de confiança da regressão também usa-se a mesma função *lines*. A função *curve* (ver *?curve*) pode produzir linhas semelhante, em alguns casos com melhores resultados, mas seu funcionamento depende de determinar uma função para que a curva seja calculada e posteriormente desenhada. Adicionalmente pode-se optar por sombrear a área deste intervalo, isso é feito com a função *polygon*. Para fazer isso primeiro é necessário gerar as coordenadas para toda a área do polígono.

```

plot(dados$x1, dados$y1,
      xlab = "Anscombe x1", ylab = "Anscombe y1",
      las = 1,
      ylim = c(3, 12),
      pch = 19,
      bty = "l")

```

```

lines(new$x1, fitted[, "fit"], lty = 1)

lines(new$x1, fitted[, "lwr"], lty = 3)
lines(new$x1, fitted[, "upr"], lty = 3)

coord <- data.frame(x = c(new$x1, rev(new$x1)), y = c(fitted[, "lwr"], rev(fitted[, "upr"])))
head(coord) # Coordenadas da área do polígono
      x      y
1 4.000000 3.422513

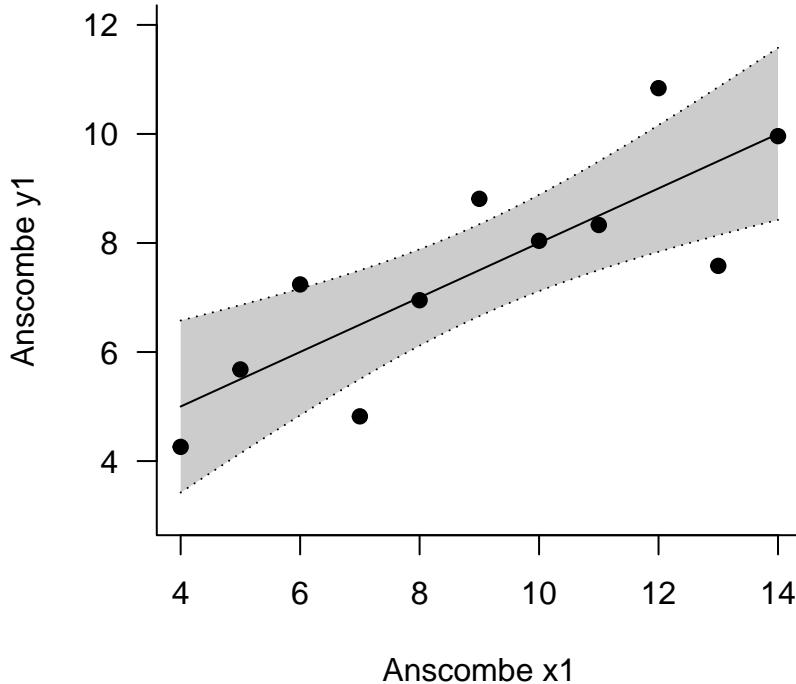
```

```

2 4.101010 3.495730
3 4.202020 3.568810
4 4.303030 3.641747
5 4.404040 3.714534
6 4.505051 3.787163

polygon(coord, border = NA, col = adjustcolor("black", alpha = 0.2))

```



## 4.4 Gráfico de pontos com ajustes não lineares

### 4.4.0.1 Dados

Conjunto de dados simulados sem relação linear, sendo valores entre 1 e 100 para a variável independente e variável dependente uma função logarítmica da variável dependente mais um componente aleatório. Quatro modelos são gerados, cada um com uma relação entre as variáveis. Os coeficientes dos modelos são organizados e servirão para construir as funções dos modelos e legenda.

```

set.seed(1)
dados <- data.frame(dx = 1:100,
                      dy = 0.4 + (4.8 * log(1:100)) + rnorm(100, 0, 1))
str(dados)
'data.frame':   100 obs. of  2 variables:
 $ dx: int  1 2 3 4 5 6 7 8 9 10 ...
 $ dy: num -0.226 3.911 4.838 8.649 8.455 ...

# Modelos
mod1 <- lm(dy ~ 1, data = dados) # Média
mod2 <- lm(dy ~ dx, data = dados) # Linear
mod3 <- nls(dy ~ a + b * log(dx), data = dados, start = list(a = 0, b = 0)) # Logaritmo
mod4 <- nls(dy ~ a + b/dx, data = dados, start = list(a = 0, b = 0)) # Assintótico

# Organização dos coeficientes

```

```

coeficientes <- data.frame(m1 = paste("y = ", round(coefficients(mod1)[1], 2), sep = ""),
                            m2 = paste("y = ", round(coefficients(mod2)[1], 2),
                                       " + (", round(coefficients(mod2)[2], 2), "*x)", sep = ""),
                            m3 = paste("y = ", round(coefficients(mod3)[1], 2),
                                       " + (", round(coefficients(mod3)[2], 2),
                                       "*log(x)", sep = ""),
                            m4 = paste("y = ", round(coefficients(mod4)[1], 2), " + (",
                                       round(coefficients(mod4)[2], 2), "/x)", sep = ""))
coeficientes # Coeficientes para cada modelo
      m1           m2           m3
1 y = 17.97 y = 11.04 + (0.14*x) y = 0.5 + (4.8*log(x))
               m4
1 y = 19.51 + (-29.64/x)

```

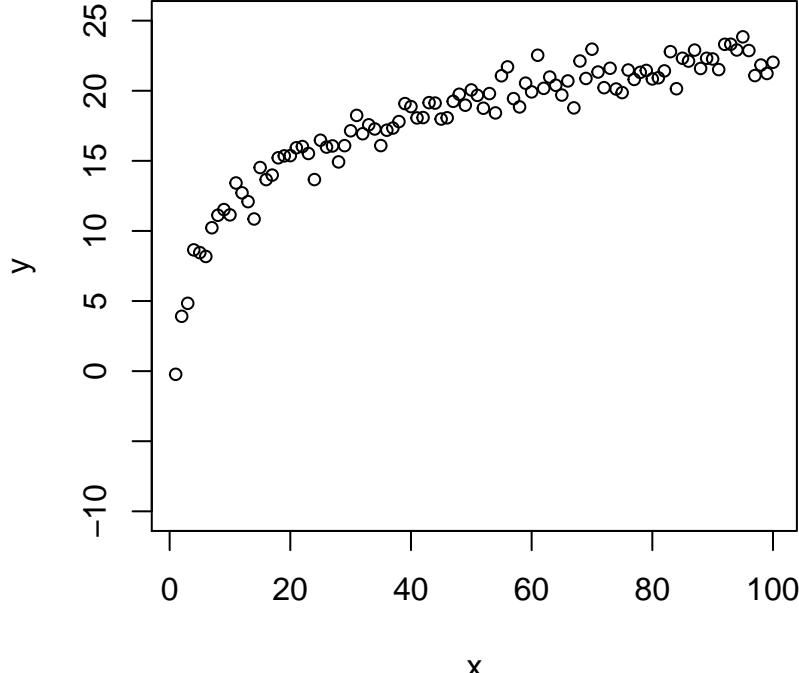
#### 4.4.0.2 Plot

Gráfico de pontos e mostrando as linha dos modelos. Além disso, adicionar a legenda com os parâmetros estimados.

```

plot(dy ~ dx,
      data = dados,
      ylim = c(-10, 25),
      pch = 21,
      cex = 0.8,
      bty = "o",
      ylab = "y", xlab = "x")

```



Para mostrar a curva com cada um dos modelos, pode-se usar a função *lines* ou a função *curve*. No caso da função *curve* deve-se construir uma função para cada uma das linhas. A função é contruída com base nos parâmetros estimados para cada modelo, para isso basta usar *function(x)* e adicionar os parâmetros do modelo correspondente.

Argumentos:

**from** - Valor inicial do limite onde a função é mostrada.  
**to** - Valor final do limite onde a função é mostrada.  
**add** - Argumento lógico para especificar se curvas devem ser adicionadas ao gráfico existente ou não.  
Padrão = FALSE.  
**lty** - Tipo de linha para a curva.  
**lwd** - Espessura da linha na curva.

```
# Funções de cada modelo
fmod1 <- function(x) 17.97 + (0 * x)
fmod2 <- function(x) 11.04 + (0.14 * x)
fmod3 <- function(x) 0.5 + (4.8 * log(x))
fmod4 <- function(x) 19.51 + (-29.64/x)

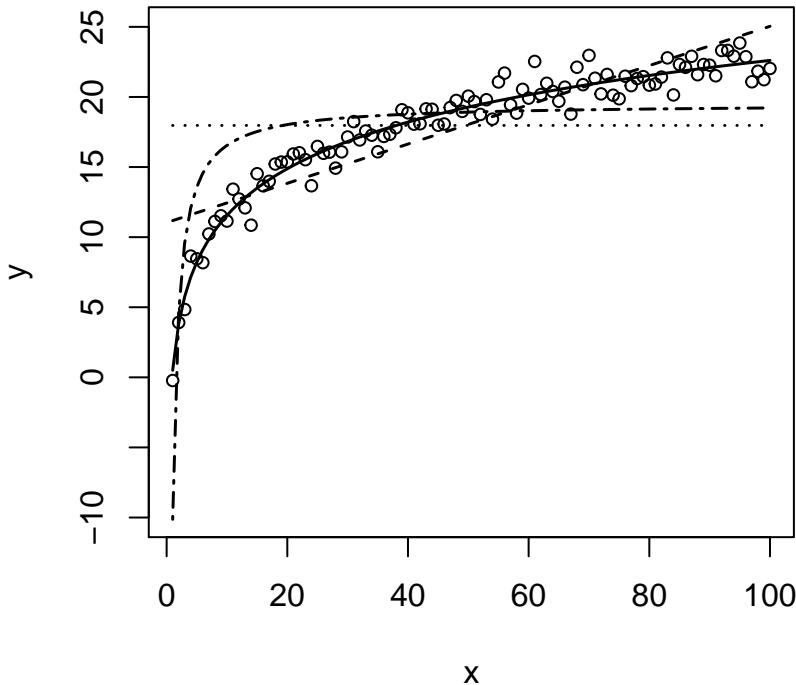
plot(dy ~ dx,
      data = dados,
      ylim = c(-10, 25),
      pch = 21,
      cex = 0.8,
      bty = "o",
      ylab = "y", xlab = "x")

curve(fmod1, from = 1, to = 100,
       add = TRUE,
       lty = 3,
       lwd = 1.5)

curve(fmod2, from = 1, to = 100,
       add = TRUE,
       lty = 2,
       lwd = 1.5)

curve(fmod3, from = 1, to = 100,
       add = TRUE,
       lty = 1,
       lwd = 1.5)

curve(fmod4, from = 1, to = 100,
       add = TRUE,
       lty = 6,
       lwd = 1.5)
```



Adicionar a legenda com os parâmetros estimados. Os parâmetros estão organizados dentro do objeto coeficientes.

Argumentos:

**legend** - Especificar nomes para os itens da legenda.

**bty** - Especificar tipo de borda da legenda.

**lty** - Tipo de linha para cada curva.

**lwd** - Espessura da linha na curva.

```
# Funções de cada modelo
fmod1 <- function(x) 17.97 + (0 * x)
fmod2 <- function(x) 11.04 + (0.14 * x)
fmod3 <- function(x) 0.5 + (4.8 * log(x))
fmod4 <- function(x) 19.51 + (-29.64/x)

plot(dy ~ dx,
      data = dados,
      ylim = c(-10, 25),
      pch = 21,
      cex = 0.8,
      bty = "o",
      ylab = "y", xlab = "x")

curve(fmod1, from = 1, to = 100,
      add = TRUE,
      lty = 3,
      lwd = 1.5)

curve(fmod2, from = 1, to = 100,
      add = TRUE,
      lty = 2,
      lwd = 1.5)
```

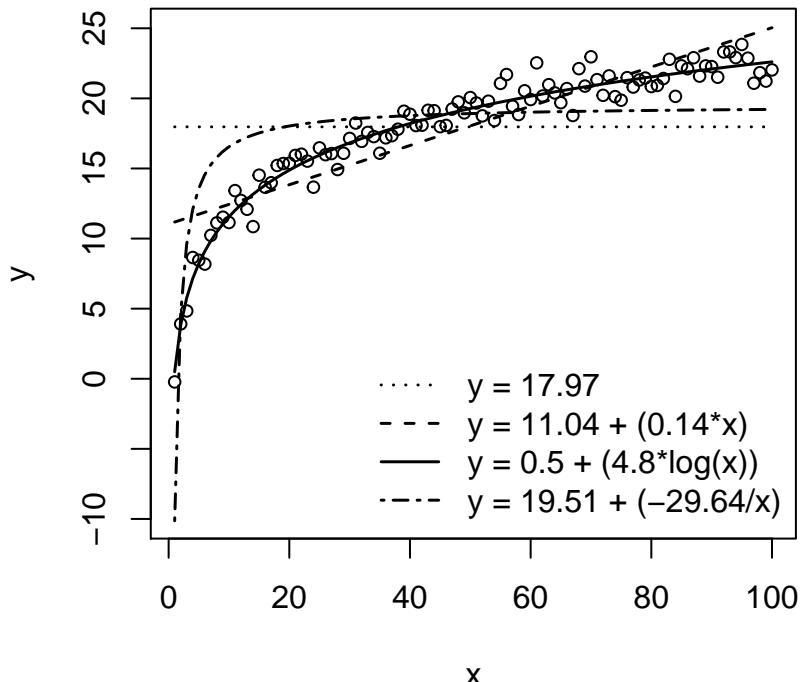
```

curve(fmod3, from = 1, to = 100,
      add = TRUE,
      lty = 1,
      lwd = 1.5)

curve(fmod4, from = 1, to = 100,
      add = TRUE,
      lty = 6,
      lwd = 1.5)

legend("bottomright",
       legend = c(paste(coeficientes$m1),
                  paste(coeficientes$m2),
                  paste(coeficientes$m3),
                  paste(coeficientes$m4)),
       bty = "n",
       lty = c(3, 2, 1, 6),
       lwd = 1.5)

```



## 4.5 Diagrama de dispersão

### 4.5.0.1 Dados

Dados sobre criminalidade dos Estados Unidos da América em 1973, descrito por quatro variáveis. Pode-se aplicar uma Análise de Componentes Principais (PCA) e usar os *scores* para construir o gráfico com os dois primeiros eixos.

```

data(USArrests)
str(USArrests)
'data.frame':   50 obs. of  4 variables:
 $ Murder   : num  13.2 10 8.1 8.8 9 7.9 3.3 5.9 15.4 17.4 ...
 $ Assault  : int  236 263 294 190 276 204 110 238 335 211 ...

```

```

$ UrbanPop: int  58 48 80 50 91 78 77 72 80 60 ...
$ Rape     : num  21.2 44.5 31 19.5 40.6 38.7 11.1 15.8 31.9 25.8 ...

require(FactoMineR)
Loading required package: FactoMineR

Attaching package: 'FactoMineR'
The following object is masked from 'package:ade4':
    reconst

Rpca <- PCA(USArrests, graph = FALSE) # PCA
str(Rpca$eig) # Autovalores
num [1:4, 1:3] 2.48 0.99 0.357 0.173 62.006 ...
- attr(*, "dimnames")=List of 2
..$ : chr [1:4] "comp 1" "comp 2" "comp 3" "comp 4"
..$ : chr [1:3] "eigenvalue" "percentage of variance" "cumulative percentage of variance"
str(Rpca$var$coord) # Correlação com os eixos
num [1:4, 1:4] 0.844 0.918 0.438 0.856 -0.416 ...
- attr(*, "dimnames")=List of 2
..$ : chr [1:4] "Murder" "Assault" "UrbanPop" "Rape"
..$ : chr [1:4] "Dim.1" "Dim.2" "Dim.3" "Dim.4"
str(Rpca$ind$coord) # Scores das unidades amostrais
num [1:50, 1:4] 0.986 1.95 1.763 -0.141 2.524 ...
- attr(*, "dimnames")=List of 2
..$ : chr [1:50] "Alabama" "Alaska" "Arizona" "Arkansas" ...
..$ : chr [1:4] "Dim.1" "Dim.2" "Dim.3" "Dim.4"

dados_unidades <- Rpca$ind$coord[, 1:2]
str(dados_unidades) # Scores das unidades amostrais para os dois primeiros eixos
num [1:50, 1:2] 0.986 1.95 1.763 -0.141 2.524 ...
- attr(*, "dimnames")=List of 2
..$ : chr [1:50] "Alabama" "Alaska" "Arizona" "Arkansas" ...
..$ : chr [1:2] "Dim.1" "Dim.2"

dados_variaveis <- Rpca$var$coord[, 1:2]
str(dados_variaveis) # Correlação das variáveis para os dois primeiros eixos
num [1:4, 1:2] 0.844 0.918 0.438 0.856 -0.416 ...
- attr(*, "dimnames")=List of 2
..$ : chr [1:4] "Murder" "Assault" "UrbanPop" "Rape"
..$ : chr [1:2] "Dim.1" "Dim.2"

```

#### 4.5.0.2 Plot

Para construir o diagrama de dispersão e mostrar simultaneamente os dados das unidades amostrais e a correlação com as variáveis o gráfico precisa ter quatro eixos, com duas escalas diferentes. A primeira coisa que deve ser feita é alterar as margens padrão do gráfico. O argumento *mar* deve ser alterado antes de começar a construir o gráfico na função *par*.

Argumentos:

**mar** - Especificar margens para o gráfico. O argumento é do tipo vetor que indica o número de linhas de margem para os quatro lados na forma c(margem inferior, à esquerda, superior, margem à direita). O padrão é c(5.1, 4.1, 4.1, 2.1).

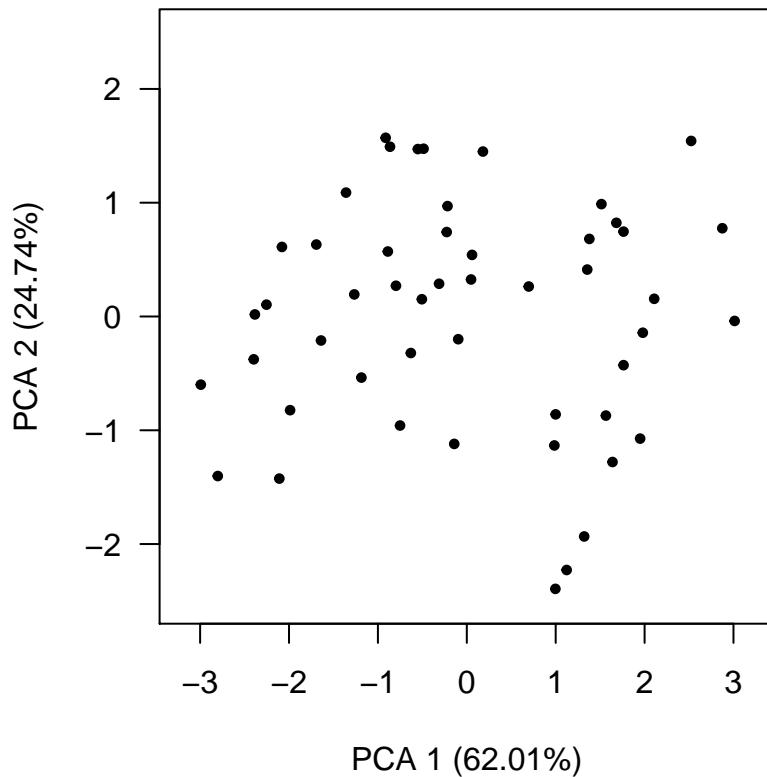
**pch** - Tipo de símbolo para os pontos.

```

op <- par(mar = c(4, 4, 3, 3))

plot(dados_unidades,
      pch = 19,
      cex = 0.6,
      ylab = "PCA 2 (24.74%)", xlab = "PCA 1 (62.01%)",
      xlim = c(-3.2, 3.2), ylim = c(-2.5, 2.5),
      las = 1)

```



A segunda parte do gráfico consiste em sobrepor sobre o mesmo gráfico o segundo conjunto de dados. A função *par* é usada novamente com o argumento *new = TRUE*, isso faz com que o próximo gráfico seja construído sobreposto ao gráfico antigo. Eixos e nomes dos eixos não devem ser mostrados, eles deverão ser adicionados pela função *axis*.

Argumentos:

**new** - Argumento lógico para especificar se o próximo gráfico deve ser construído sem apagar o gráfico antigo.

**side** - Especificar lado que o eixo deve ser mostrado. Valores aceitos 1 (eixo inferior), 2 (eixo à esquerda), 3 (eixo superior) e 4 (eixo à direita).

```

op <- par(mar = c(4, 4, 3, 3))

plot(dados_unidades,
      pch = 19,
      cex = 0.6,
      ylab = "PCA 2 (24.74%)", xlab = "PCA 1 (62.01%)",
      xlim = c(-3.2, 3.2), ylim = c(-2.5, 2.5),
      las = 1)

```

```

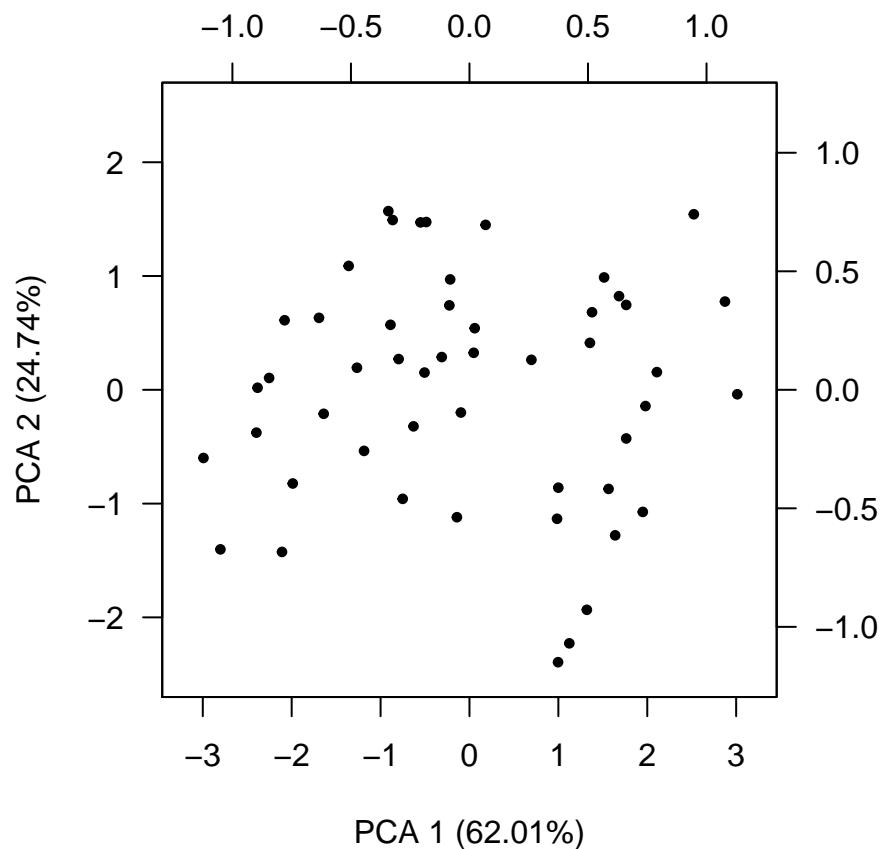
op <- par(new = TRUE)

plot(dados_variaveis,
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      type = "n",
      ylim = c(-1.2, 1.2), xlim = c(-1.2, 1.2))

axis(side = 4, las = 1)

axis(side = 3)

```



O segundo conjuntos de pontos indicando a correlação entre as variáveis e os eixos serão mostrado na forma de texto.

Argumentos:

**labels** - Especificar texto a ser adicionado em cada coordenada.

**cex** - Especificar tamanho da fonte do texto.

**adj** - Especificar se o texto deve se mostrado centrado exatamente da coordenada ou adjacente a coordenada. Argumento do tipo  $c(x, y)$ , onde os valores representam o ajuste nos eixos x e y respectivamente. Se  $adj$  for 0 os texto é mostrado justificado a esquerda da coordenada.

```

op <- par(mar = c(4, 4, 3, 3))

plot(dados_unidades,
      pch = 19,
      cex = 0.6,

```

```

ylab = "PCA 2 (24.74%)", xlab = "PCA 1 (62.01%)",
xlim = c(-3.2, 3.2), ylim = c(-2.5, 2.5),
las = 1)

op <- par(new = TRUE)

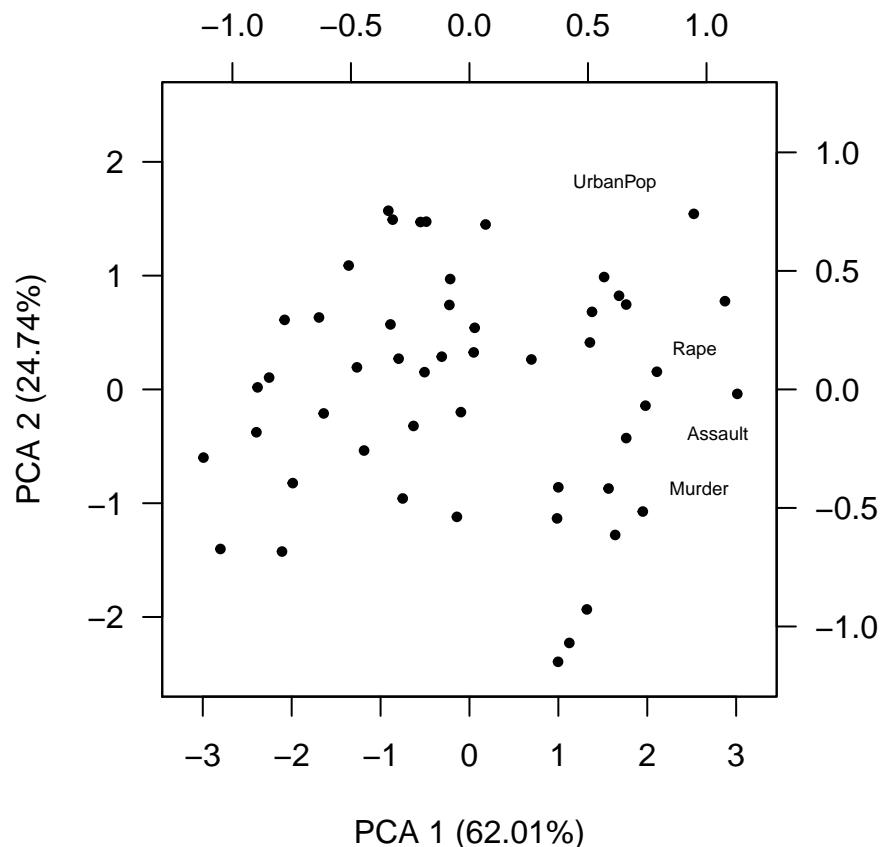
plot(dados_variaveis,
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      type = "n",
      ylim = c(-1.2, 1.2), xlim = c(-1.2, 1.2))

axis(side = 4, las = 1)

axis(side = 3)

text(dados_variaveis,
      labels = rownames(dados_variaveis),
      cex = 0.6,
      adj = 0)

```



Para finalizar adiciona-se setas que partem da origem até os valores de cada uma das coordenadas de texto e linhas simples na origem de ambos os eixos.

Argumentos:

**length** - Comprimento da ponta da seta, em polegadas.

**h** e **v** - Posição para adicionar linha na horizontal e vertical, respectivamente.

```

op <- par(mar = c(4, 4, 3, 3))

plot(dados_unidades,
      pch = 19,
      cex = 0.6,
      ylab = "PCA 2 (24.74%)", xlab = "PCA 1 (62.01%)",
      xlim = c(-3.2, 3.2), ylim = c(-2.5, 2.5),
      las = 1)

op <- par(new = TRUE)

plot(dados_variaveis,
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      type = "n",
      ylim = c(-1.1, 1.1), xlim = c(-1.1, 1.1))

axis(side = 4, las = 1)

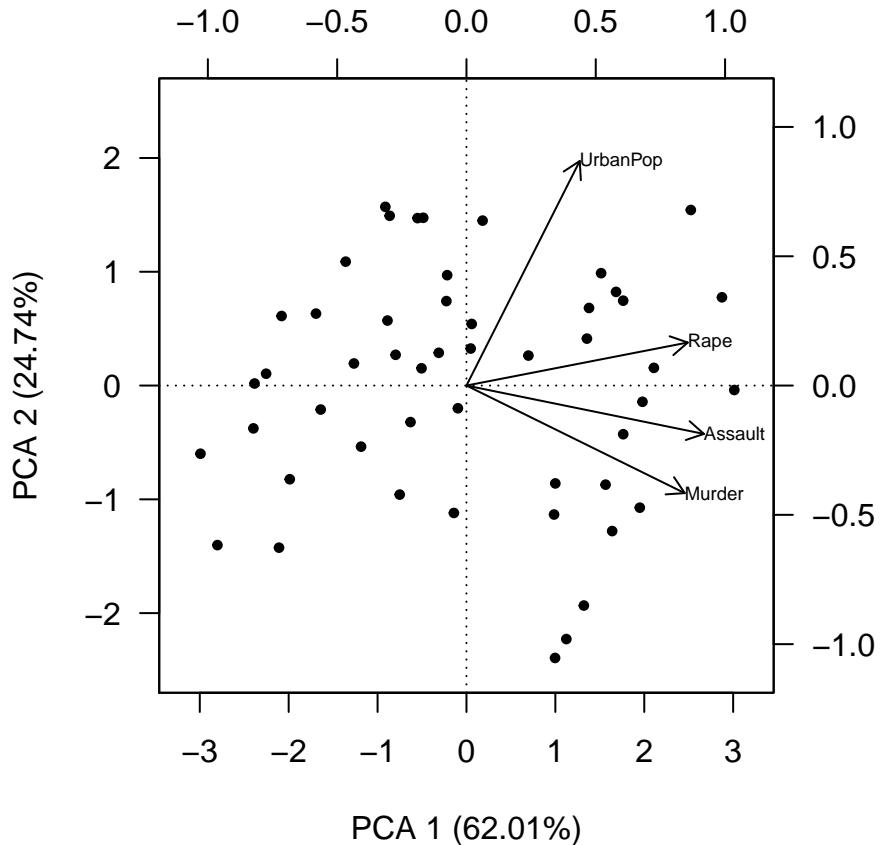
axis(side = 3)

text(dados_variaveis,
      labels = rownames(dados_variaveis),
      cex = 0.6,
      adj = 0)

arrows(x0 = 0,
       y = 0,
       x1 = dados_variaveis[, 1],
       y1 = dados_variaveis[, 2],
       length = 0.1)

abline(h = 0,
       v = 0,
       lty = 3)

```



```
par(op) # Restaurar parâmetros gráficos originais.
```

## 4.6 Boxplot

### 4.6.0.1 Dados

Dados de experimento sobre o rendimento (peso seco de plantas) obtidos sob controle e dois tratamento. Uma ANOVA seguida por um teste de Tukey foram realizados para verificar se existem diferenças significativas entre os tratamentos.

```
data(PlantGrowth)
dados <- PlantGrowth
str(dados)
'data.frame': 30 obs. of 2 variables:
 $ weight: num 4.17 5.58 5.18 6.11 4.5 4.61 5.17 4.53 5.33 5.14 ...
 $ group : Factor w/ 3 levels "ctrl","trt1",...: 1 1 1 1 1 1 1 1 1 1 ...

```

```
mod <- lm(weight ~ group, data = dados) # ANOVA
summary.aov(mod)
  Df Sum Sq Mean Sq F value Pr(>F)
group      2  3.766  1.8832   4.846 0.0159 *
Residuals 27 10.492  0.3886

```

```
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
TukeyHSD(aov(weight ~ group, data = dados)) # Tukey
Tukey multiple comparisons of means
```

```

95% family-wise confidence level

Fit: aov(formula = weight ~ group, data = dados)

$group
    diff      lwr      upr     p adj
trt1-ctrl -0.371 -1.0622161 0.3202161 0.3908711
trt2-ctrl  0.494 -0.1972161 1.1852161 0.1979960
trt2-trt1  0.865  0.1737839 1.5562161 0.0120064

```

#### 4.6.0.2 Plot

Gráfico do tipo *boxplot* permite visualizar o efeito dos tratamentos. Neste caso é usado a notação de fórmula para as variáveis.

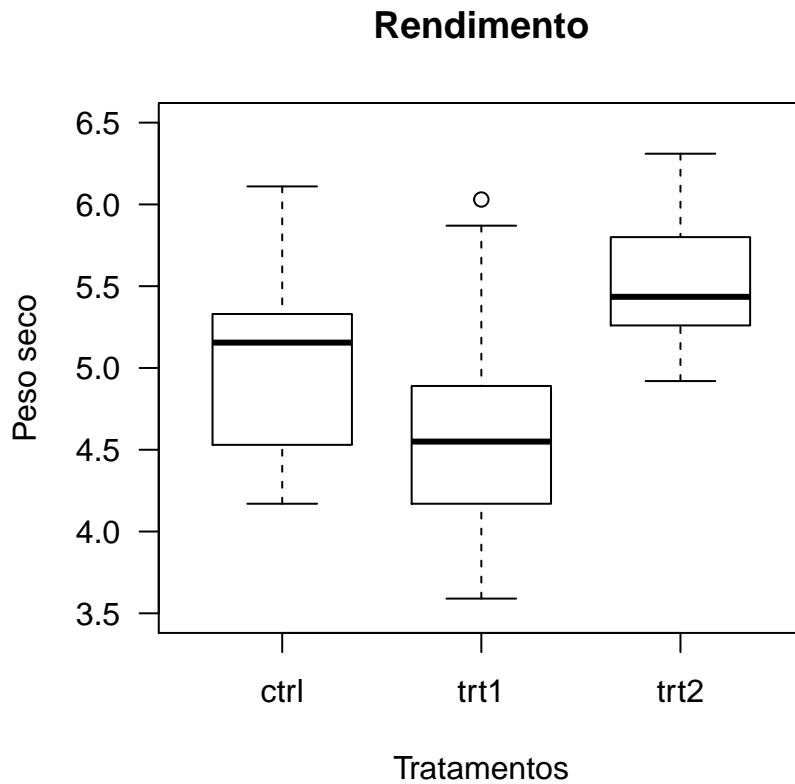
Argumento:

**boxwex** - Especificar largura das caixas.

```

boxplot(weight ~ group,
        data = dados,
        ylab = "Peso seco", xlab = "Tratamentos",
        main = "Rendimento",
        boxwex = 0.7,
        las = 1,
        ylim = c(3.5, 6.5))

```



Pode-se usar a função *text* para destacar no gráfico as diferenças significativas entre os tratamentos.

```

boxplot(weight ~ group,
        data = dados,

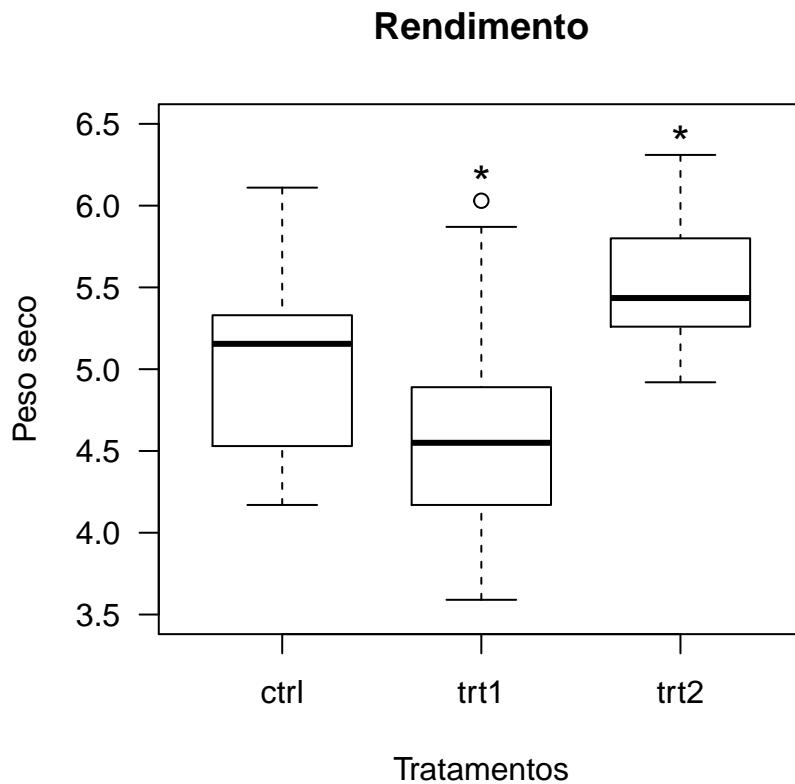
```

```

ylab = "Peso seco", xlab = "Tratamentos",
main = "Rendimento",
boxwex = 0.7,
las = 1,
ylim = c(3.5, 6.5))

text(x = c(2, 3),
      y = c(6.2, 6.45),
      labels = "*",
      cex = 1.5)

```



Para mudar a ordem dos tratamentos no gráfico é preciso alterar a ordem dos fatores nos dados. Além disso, pode-se inverter as cores das caixas e trocar o símbolo dos outliers.

Argumentos:

**col** - Cor das caixas.

**border** - Cor das bordas das caixas.

**whiskcol** - Cor das linhas.

**staplecol** - Cor das barras limites.

**outpch** - Tipo de símbolo para os outliers.

```

dados$group <- factor(dados$group,
                      levels = c("ctrl", "trt2", "trt1"))
str(dados)
'data.frame':   30 obs. of  2 variables:
 $ weight: num  4.17 5.58 5.18 6.11 4.5 4.61 5.17 4.53 5.33 5.14 ...
 $ group : Factor w/ 3 levels "ctrl","trt2",...: 1 1 1 1 1 1 1 1 1 ...

```

```

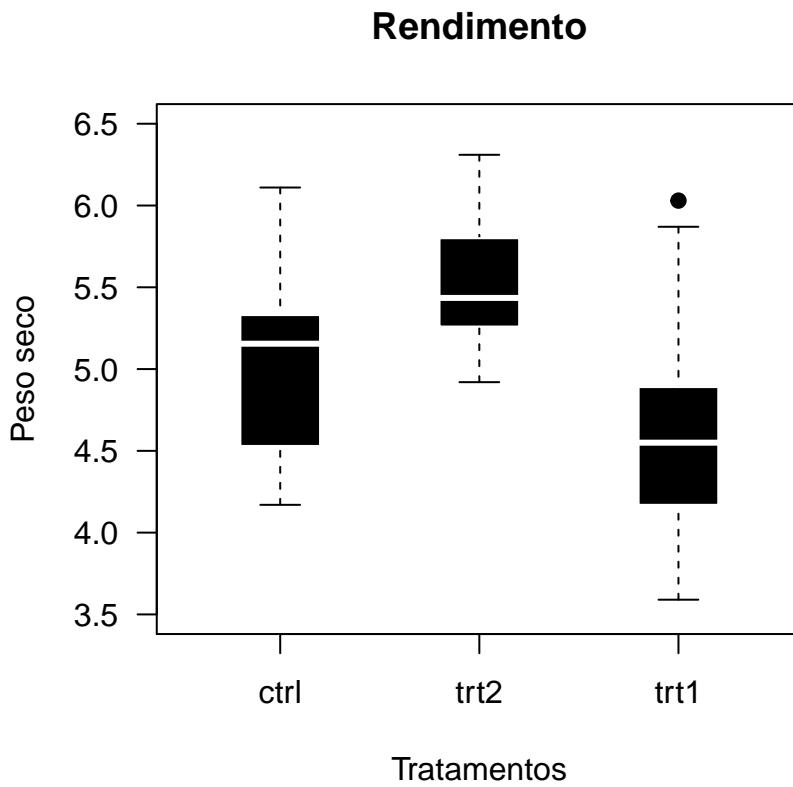
boxplot(weight ~ group,
        data = dados,

```

```

ylab = "Peso seco", xlab = "Tratamentos",
main = "Rendimento",
las = 1,
ylim = c(3.5, 6.5),
col = "black",
border = "white",
whiskcol = "black",
staplecol = "black",
outcol = "black",
outpch = 19,
outpch = 19,
boxwex = 0.4)

```



Para finalizar pode-se alterar os nomes dos tratamentos e alterar o alcance das barras do *boxplot*. Por padrão o alcance das barras estendem-se até 1.5 desvios quantílicos, que é a medida da diferença entre o primeiro e o terceiro quartil da variável.

Argumentos:

**names** - Alterar nome dos níveis.  
**medcol** - Alterar cores da linha do meio.

**range** - Determinar o range das barras. Padrão é 1.5, se for 0 as barras são mostradas até o limites dos dados.

```

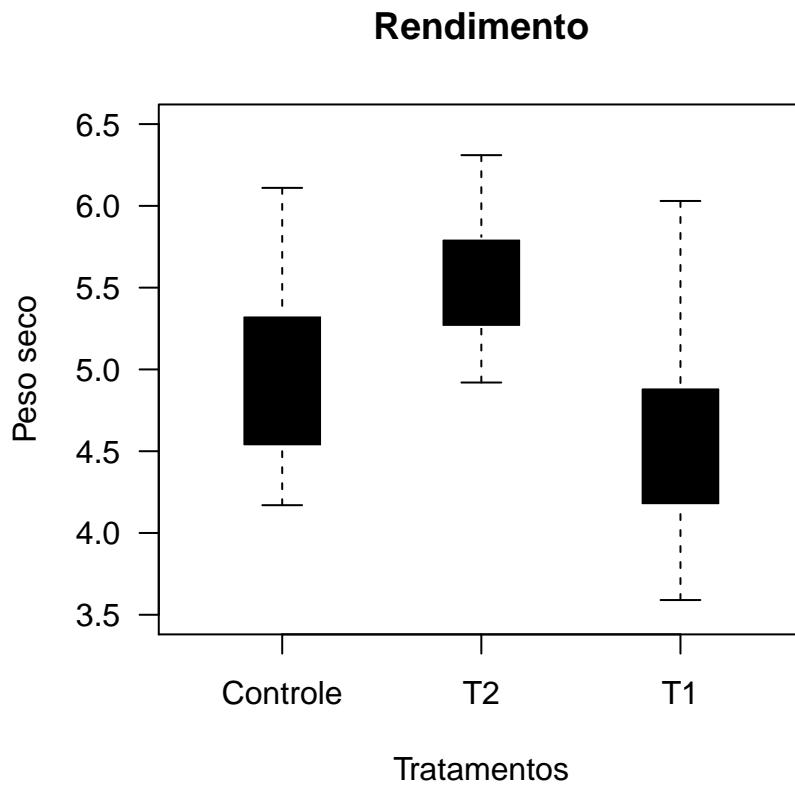
boxplot(weight ~ group,
        data = dados,
        ylab = "Peso seco", xlab = "Tratamentos",
        main = "Rendimento",
        las = 1,
        ylim = c(3.5, 6.5),
        col = "black",
        border = "white",

```

```

whiskcol = "black",
staplecol = "black",
outcol = "black",
medcol = "black",
outpch = 19,
range = 0,
boxwex = 0.4,
names = c("Controle", "T2", "T1"))

```



## 4.7 Beanplot

Os beanplot podem ser uma alternativa bastante interessante para os *boxplot*, já que além de mostrar a distribuição geral dos dados é possível mostrar todas as observações individuais. Os gráficos são realizados com a ajuda do pacote *beanplot*.

### 4.7.0.1 Dados

Os dados *ToothGrowth* são respostas do comprimento dos odontoblastos (células responsáveis pelo crescimento dentário) em 60 cobaias. Cada animal recebeu um dos três níveis de dose de vitamina C (0,5, 1 e 2 mg/dia) por um dos dois métodos de entrega, suco de laranja (OJ) ou ácido ascórbico (VC).

```

require(beanplot)
Loading required package: beanplot
data(ToothGrowth)
dados <- ToothGrowth
dados$dose <- as.factor(dados$dose)
str(dados)
'data.frame':   60 obs. of  3 variables:
 $ len : num  4.2 11.5 7.3 5.8 6.4 10 11.2 11.2 5.2 7 ...

```

```
$ supp: Factor w/ 2 levels "0J","VC": 2 2 2 2 2 2 2 2 2 ...
$ dose: Factor w/ 3 levels "0.5","1","2": 1 1 1 1 1 1 1 1 1 ...
```

#### 4.7.0.2 Plot

Gráfico do tipo beanplot permite visualizar o efeito dos tratamentos. Neste caso é usado a notação de fórmula para as variáveis. Além disso, os dois conjuntos de tratamento serão mostrados no mesmo gráfico para facilitar a visualização. Essa função apresenta muitas opções, ver `?beanplot` para mais detalhes.

Argumentos:

**boxwex** - Fator de escala aplicado para definir a largura das distribuições.

**at** - Posição de cada variável.

**subset** - Subconjunto de dados utilizado no gráfico atual.

**log** - Argumento para forçar ou não a transformação dos eixos em escala logarítmica.

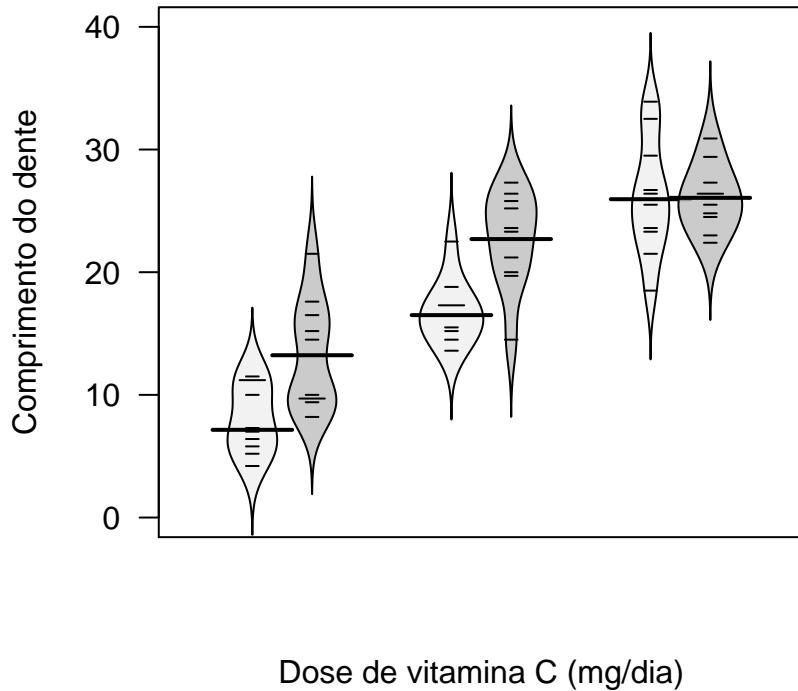
**what** - Vetor de quatro elementos descrevendo o que mostrar no gráfico, sendo que 0 significa não mostrar e 1 mostrar. Na forma c(média total, distribuições, média de cada categoria, observações individuais). Padrão é `c(1,1,1,1)`.

**beanlines** - método usado para determinar as linhas médias. Valores aceitos são “mean”, “median” e “quantiles”.

**add** - Argumento lógico para indicar se as informações devem ser adicionadas ao gráfico existente ou não. Padrão é FALSE.

```
beanplot(len ~ dose,
         data = dados,
         boxwex = 0.4,
         at = 1:3 - 0.15,
         subset = supp == "VC",
         col = adjustcolor("grey", alpha.f = 0.2),
         ylim = c(0, 40),
         xlab = "Dose de vitamina C (mg/dia)",
         ylab = "Comprimento do dente",
         log = "",
         las = 1,
         what = c(0,1,1,1),
         beanlines = "median",
         xaxt = "n")

beanplot(len ~ dose, data = dados,
         boxwex = 0.4,
         at = 1:3 + 0.15,
         subset = supp == "0J",
         col = adjustcolor("grey", alpha.f = 0.8),
         what = c(0,1,1,1),
         xaxt = "n",
         add = TRUE
)
```



Para finalizar adiciona-se o eixo e as legendas.

Argumentos:

**lwd.ticks** - espessura da linha das marcas de escala.

**fill** - cor do preenchimento usado nas caixas ao lado do texto da legenda.

```
beanplot(len ~ dose,
         data = dados,
         boxwex = 0.4,
         at = 1:3 - 0.15,
         subset = supp == "VC",
         col = adjustcolor("grey", alpha.f = 0.2),
         ylim = c(0, 40),
         xlab = "Dose de vitamina C (mg/dia)",
         ylab = "Comprimento do dente",
         log = "",
         las = 1,
         what = c(0,1,1,1),
         beanlines = "median",
         xaxt = "n")

beanplot(len ~ dose, data = dados,
         boxwex = 0.4,
         at = 1:3 + 0.15,
         subset = supp == "OJ",
         col = adjustcolor("grey", alpha.f = 0.8),
         what = c(0,1,1,1),
         xaxt = "n",
         add = TRUE
)

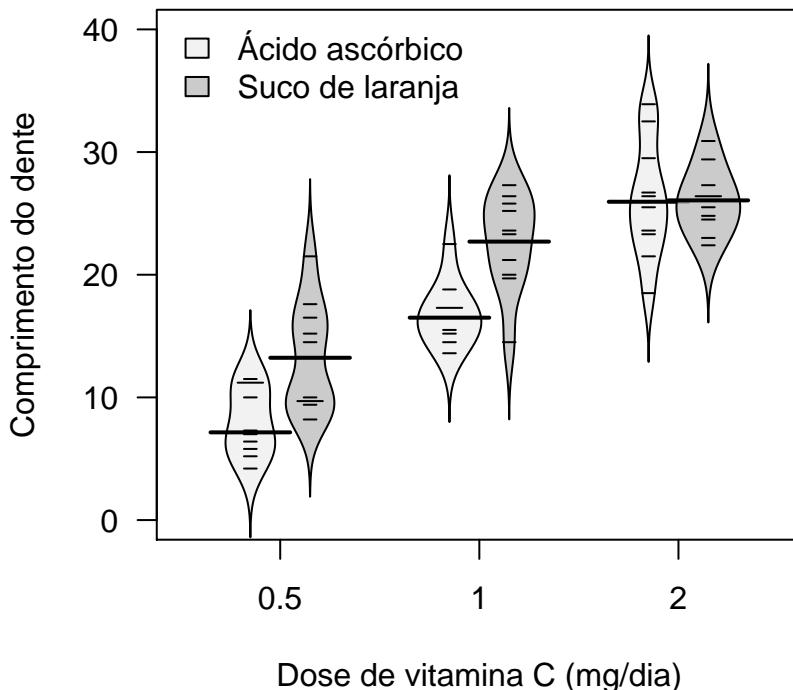
axis(side = 1,
```

```

at = 1:3,
label = levels(dados$dose),
lwd = 0,
lwd.ticks = 1)

legend("topleft",
      bty = "n",
      legend = c("Ácido ascórbico", "Suco de laranja"),
      fill = c(adjustcolor("grey", alpha.f = 0.2),
              adjustcolor("grey", alpha.f = 0.8)))

```



## 4.8 Gráfico de interação

### 4.8.0.1 Dados

Dados *iron* disponível no pacote *AICcmodavg* descrevem o teor de ferro de alimentos cozidos em diferentes tipos de panelas. Nos dados são dois factores e uma variável dependente. Pode-se calcular a média para cada nível dos factores combinados.

```

require(AICcmodavg)
Loading required package: AICcmodavg
data(iron)
str(iron)
'data.frame':   36 obs. of  3 variables:
 $ Pot : Factor w/ 3 levels "aluminium","clay",...: 1 1 1 1 2 2 2 2 3 3 ...
 $ Food: Factor w/ 3 levels "legumes","meat",...: 2 2 2 2 2 2 2 2 2 2 ...
 $ Iron: num  1.77 2.36 1.96 2.14 2.27 1.28 2.48 2.68 5.27 5.17 ...

```

```

dados <- tapply(iron[, 3], list(iron[, 1], iron[, 2]), mean)
dados
      legumes   meat vegetables
aluminium  2.3300  2.0575    1.2325

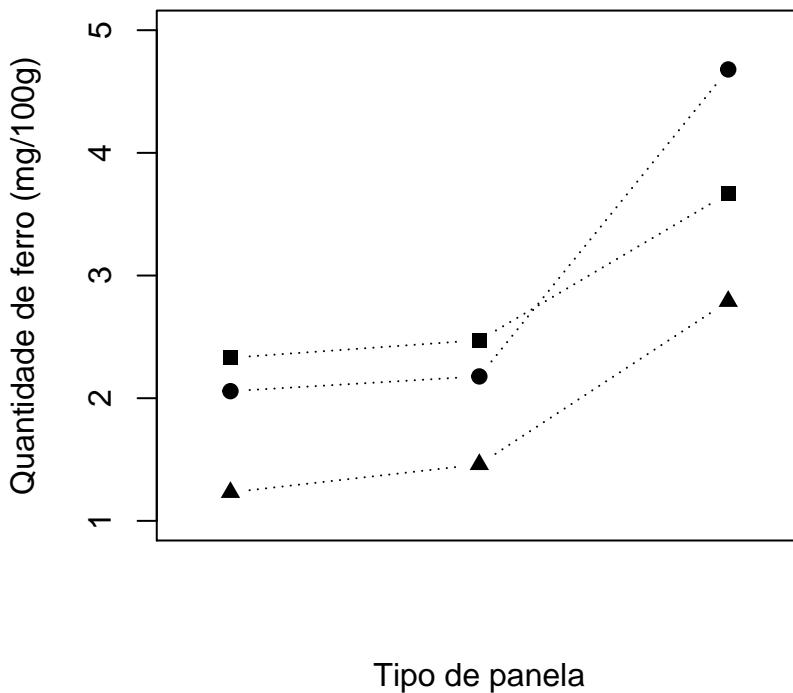
```

clay	2.4725	2.1775	1.4600
iron	3.6700	4.6800	2.7900

#### 4.8.0.2 Plot

O gráfico de interação pode mostrar a influência dos dois fatores sobre a variável dependente. Pode-se usar a função *matplot*, que funciona muito bem para este tipo de dados, mas não é tão simples. Alternativamente pode-se usar a função *interaction.plot*, que já calcula a média para cada tratamento e constrói o gráfico, mas não permite tantas personalizações. Primeiramente define-se os limites para o eixo x. Neste caso, os dados são três colunas e três linhas, os limites podem ser definidos com valores um pouco abixo de um e um pouco acima de três, pois cada nível será mostrado sobre o número inteiro de um a três. Além disso, não é mostrado nenhuma informação sobre o eixo x.

```
matplot(dados,
         type = "b",
         xlim = c(0.8, 3.2), ylim = c(1, 5),
         lty = 3,
         lwd = 1,
         pch = c(15, 19, 17),
         col = "black",
         xaxt = "n",
         xlab = "Tipo de panela", ylab = "Quantidade de ferro (mg/100g)")
```



Para adicionar o eixo x personalizado, basta definir os marcadores para o eixo e os nomes conforme os níveis da variável. É importante verificar qual dos níveis é mostrado no eixo x e qual dos níveis é mostrado nas linhas.

Argumentos:

**side** - Lado para mostrar o eixo.

**at** - Localização dos marcadores do eixo. Um vetor com todos os separadores.

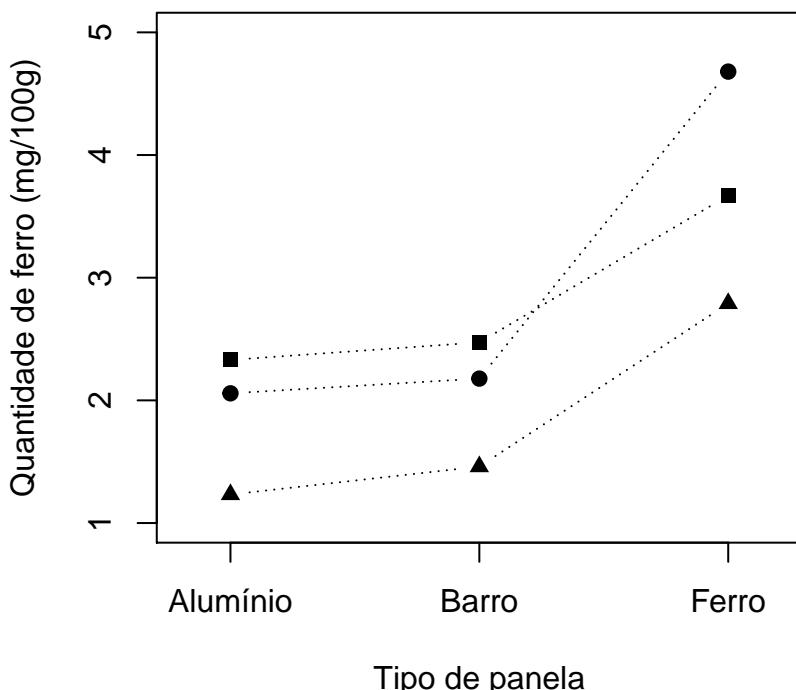
**labels** - Texto para adicionar nos marcadores.

```

matplot(dados,
        type = "b",
        xlim = c(0.8, 3.2), ylim = c(1, 5),
        lty = 3,
        lwd = 1,
        pch = c(15, 19, 17),
        col = "black",
        xaxt = "n",
        xlab = "Tipo de panela", ylab = "Quantidade de ferro (mg/100g)")

axis(side = 1,
      at = 1:3,
      labels = c("Alumínio", "Barro", "Ferro"))

```



Para finalizar adicionar a legenda.

Argumentos:

**pch** - Tipo de marcado para cada nível. Nesta caso também é importante verificar a ordem de cada nível nos dados.

**inset** - Deslocamento da inserção da legenda, tendo como referência a palavra-chave da localização.

**title** - Título para a legenda.

```

matplot(dados,
        type = "b",
        xlim = c(0.8, 3.2), ylim = c(1, 5),
        lty = 3,
        lwd = 1,
        pch = c(15, 19, 17),
        col = "black",
        xaxt = "n",
        xlab = "Tipo de panela", ylab = "Quantidade de ferro (mg/100g)")

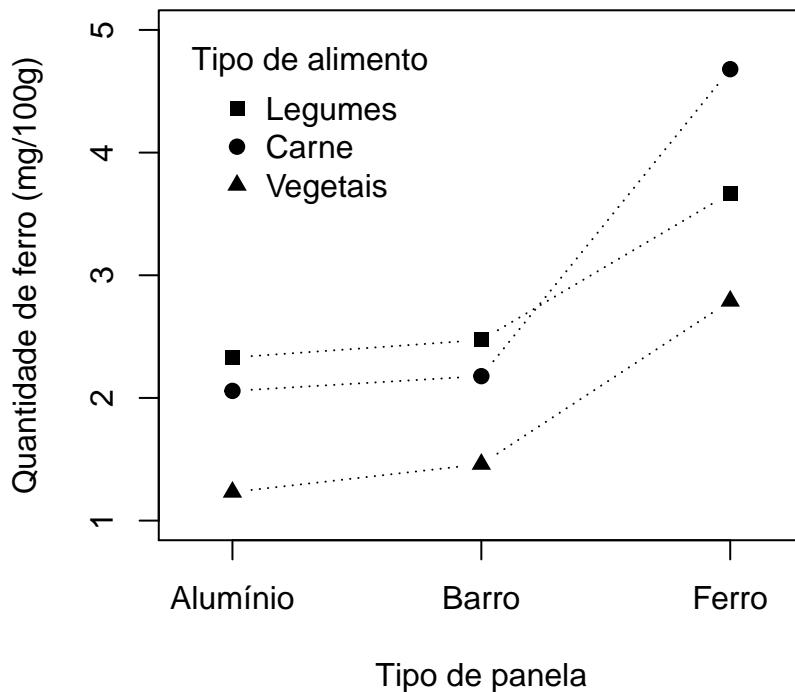
```

```

axis(side = 1,
     at = 1:3,
     labels = c("Alumínio", "Barro", "Ferro"))

legend("topleft",
       legend = c("Legumes", "Carne", "Vegetais"),
       pch = c(15, 19, 17),
       bty = "n",
       inset = 0.04,
       title = "Tipo de alimento")

```



## 4.9 Gráfico de pontos com barras de erro

### 4.9.0.1 Dados

As contagens de insetos em unidades experimentais agrícolas tratados com diferentes inseticidas. Pode-se calcular a média e o desvio padrão para cada inseticida.

```

data(InsectSprays)
str(InsectSprays)
'data.frame': 72 obs. of 2 variables:
 $ count: num 10 7 20 14 14 12 10 23 17 20 ...
 $ spray: Factor w/ 6 levels "A","B","C","D",...: 1 1 1 1 1 1 1 1 1 1 ...

dados <- tapply(InsectSprays$count, InsectSprays$spray, mean)
dados
      A          B          C          D          E          F 
14.500000 15.333333  2.083333  4.916667  3.500000 16.666667 

dados_sd <- tapply(InsectSprays$count, InsectSprays$spray, sd)
dados_sd

```

A	B	C	D	E	F
4.719399	4.271115	1.975225	2.503028	1.732051	6.213378

#### 4.9.0.2 Plot

Gráfico de pontos representando a média de cada tratamento e linhas representando os desvios padrão. Neste caso os valores do eixo x são categoricos, então oculta-se valores do eixo e define-se os alcance dos eixos. Os nomes dos inseticidas são adicionados em um segundo passo pela função *axis*.

Argumentos:

**side** - Lado para mostrar o eixo.

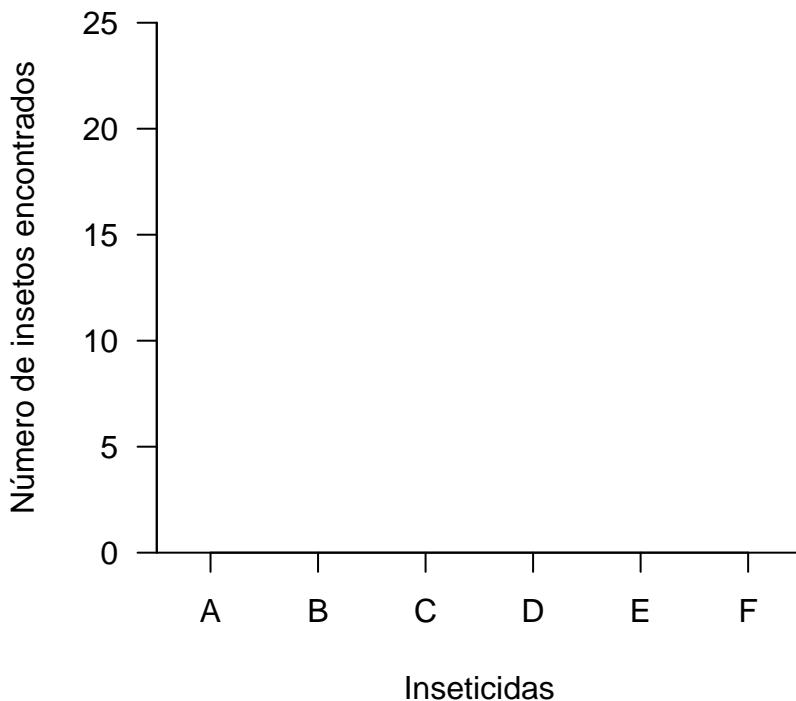
**at** - Localização dos marcadores do eixo. Um vetor com todos os separadores.

**labels** - Texto para adicionar em cada posição do marcador.

```
plot(1:6, dados,
  type = "n",
  xaxt = "n",
  ylim = c(0, 25), xlim = c(0.5, 6.5),
  bty = "l",
  xaxs = "i", yaxs = "i",
  ylab = "Número de insetos encontrados", xlab = "Inseticidas",
  las = 1,
  main = "Eficácia de inseticida")

axis(side = 1,
  at = 1:6,
  labels = rownames(dados))
```

**Eficácia de inseticida**



Para mostrar as barras de erro a função *plotCI* do pacote *plotrix* é fácil de usar.

Argumentos:

**x** e **y**- Coordenadas para o ínicio da barra de erro.  
**uiw** e **liw** - Comprimento da linha de erro para a parte superior e inferior.  
**add** - Argumento lógico para especificar se linhas de erro devem ser adicionadas ao gráfico existente ou não. Padrão é FALSE.

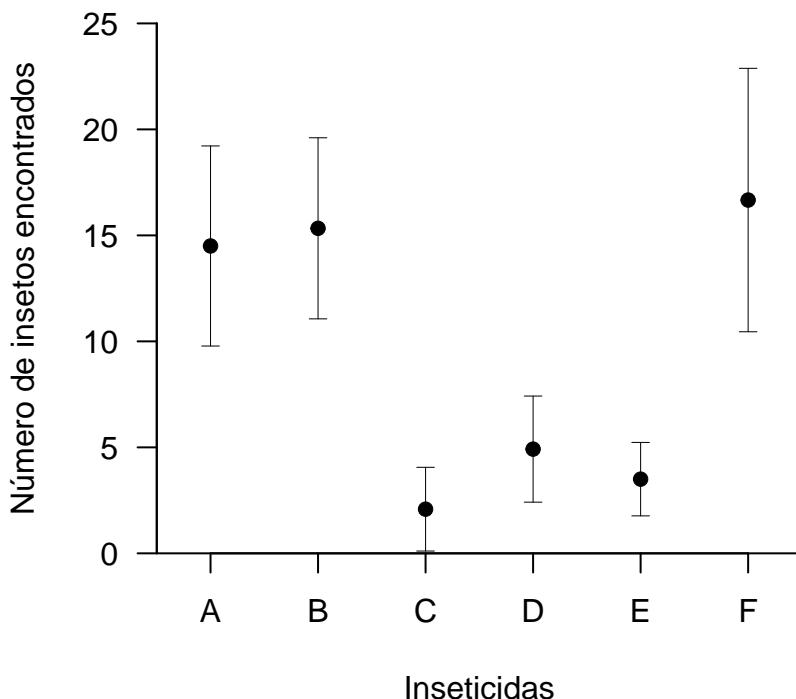
```
require(plotrix)

plot(1:6, dados,
  type = "n",
  xaxt = "n",
  ylim = c(0, 25), xlim = c(0.5, 6.5),
  bty = "l",
  xaxs = "i", yaxs = "i",
  ylab = "Número de insetos encontrados", xlab = "Inseticidas",
  las = 1,
  main = "Eficácia de inseticida")

axis(side = 1,
  at = 1:6,
  labels = rownames(dados))

plotCI(1:6, dados,
  uiw = dados_sd,
  add = TRUE,
  pch = 19,
  lwd = 0.5)
```

## Eficácia de inseticida



## 4.10 Gráfico de linhas com duas escalas

### 4.10.0.1 Dados

Os dados *airquality* descrevem a qualidade do ar de New York em alguns meses de 1973.

```
data(airquality)
dados <- airquality
str(dados)
'data.frame': 153 obs. of 6 variables:
 $ Ozone : int 41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int 190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num 7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int 67 72 74 62 56 66 65 59 61 69 ...
 $ Month  : int 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int 1 2 3 4 5 6 7 8 9 10 ...
```

### 4.10.0.2 Plot

Gráfico de linhas representado duas séries temporais com duas escalas no eixo y. Primeiramente deve-se redefinir as margens da figura. Depois mostrar a primeira série ocultando os valores e marcadores do eixo x.

Argumentos:

**mar** - Especificar margens para o gráfico. O argumento é do tipo vetor que indica o número de linhas de margem para os quatro lados na forma c(margem inferior, à esquerda, superior, margem à direita). O padrão é c(5.1, 4.1, 4.1, 2.1).

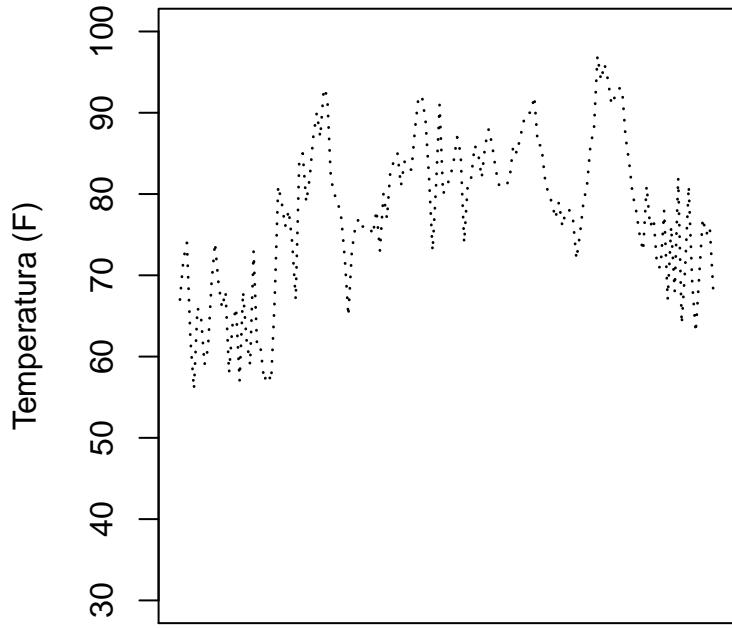
**type** - Altera o tipo de gráfico que será construído. Vários valores são aceito, os principais são “p” (gráfico de pontos), “l”(gráfico de linhas) e “b”(gráficos com pontos e linhas).

**lty** - Especificar tipo de linha.

**lwd** - Especificar espessura da linha.

```
op <- par(mar = c(3, 4, 4, 4))

plot(dados$Temp, type = "l",
      lty = 3,
      ylim = c(30, 100),
      lwd = 1.4,
      xaxt = "n",
      ylab = "Temperatura (F)", xlab = "")
```



Os intervalos de marcação no eixo x são estabelecidos com base na divisão mensal, transformando o eixo contínuo em categórico.

Argumentos:

**side** - Posição para o eixo.

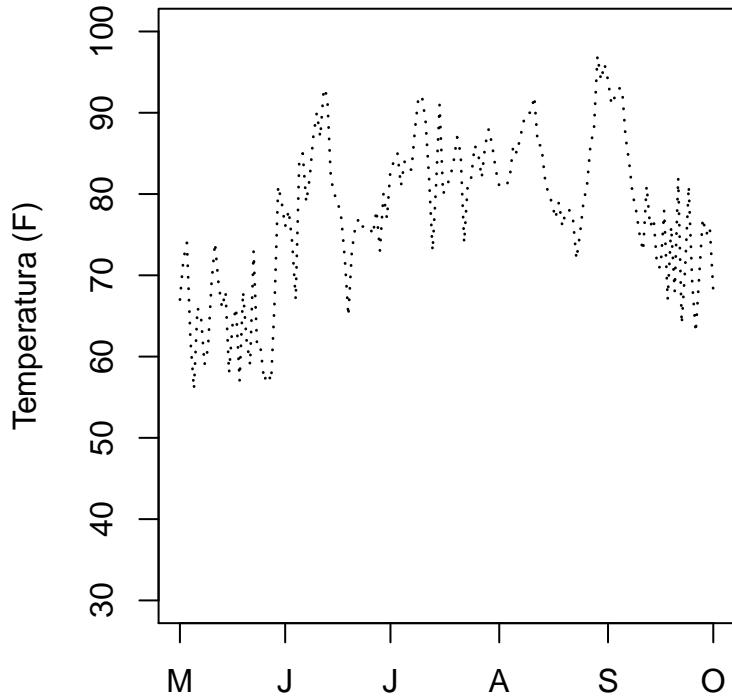
**at** - Especificar posição dos marcadores do eixo.

**label** - Especificar nome para os marcadores do eixo.

```
op <- par(mar = c(3, 4, 4, 4))

plot(dados$Temp, type = "l",
      lty = 3,
      ylim = c(30, 100),
      lwd = 1.4,
      xaxt = "n",
      ylab = "Temperatura (F)", xlab = "")

axis(side = 1,
     at = c(1, cumsum(table(dados[, 5]))),
     label = c("M", "J", "J", "A", "S", "O"))
```



A próximo passo consiste em sobrepor sobre o mesmo gráfico o segundo conjunto de dados. A função *par* é usada novamente com o argumento *new = TRUE*, isso faz com que o próximo gráfico seja construído sobreposto ao gráfico antigo. Eixos e nomes dos eixos não devem ser mostrados, eles deverão ser adicionados pela função *axis*.

Argumento:

**new** - Argumento lógico para especificar ser próximo gráfico deve ser construído sem apagar o gráfico antigo.

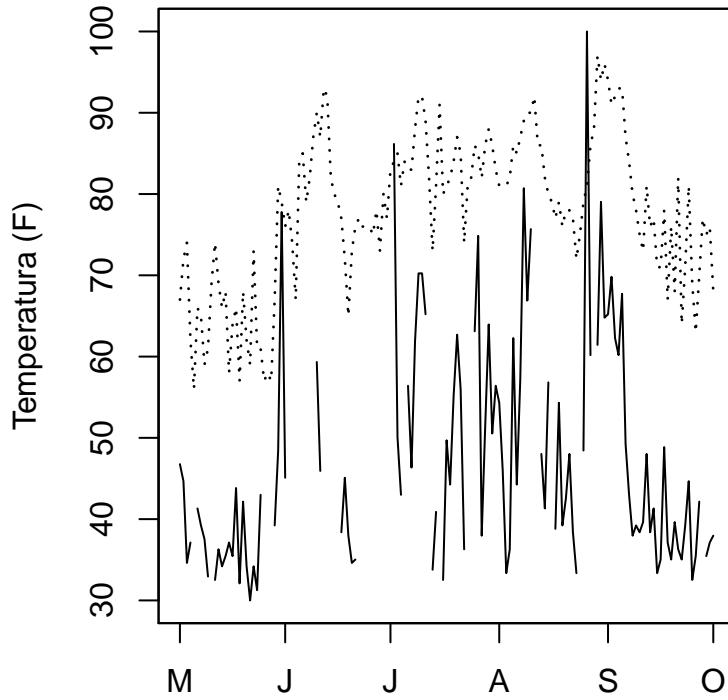
```
op <- par(mar = c(3, 4, 4, 4))

plot(dados$Temp, type = "l",
      lty = 3,
      ylim = c(30, 100),
      lwd = 1.4,
      xaxt = "n",
      ylab = "Temperatura (F)", xlab = "")

axis(side = 1,
     at = c(1, cumsum(table(dados[, 5]))),
     label = c("M", "J", "J", "A", "S", "O"))

op <- par(new = TRUE)

plot(dados$Ozone,
      type = "l",
      xlab = "", ylab = "",
      yaxt = "n", xaxt = "n",
      lty = 1)
```



Pode-se adicionar o eixo e nome do eixo na margem direita do gráfico. A função *mtext* é usada para adicionar texto nas margens do gráfico.

Argumentos:

**side** - Posição para o eixo e texto na função *mtext*.

**line** - Linha para mostrar texto. Valores próximo a 1 corresponde a posição próximo ao eixo.

```
op <- par(mar = c(3, 4, 4, 4))

plot(dados$Temp, type = "l",
      lty = 3,
      ylim = c(30, 100),
      lwd = 1.4,
      xaxt = "n",
      ylab = "Temperatura (F)", xlab = "")

axis(side = 1,
     at = c(1, cumsum(table(dados[, 5]))),
     label = c("M", "J", "J", "A", "S", "O"))

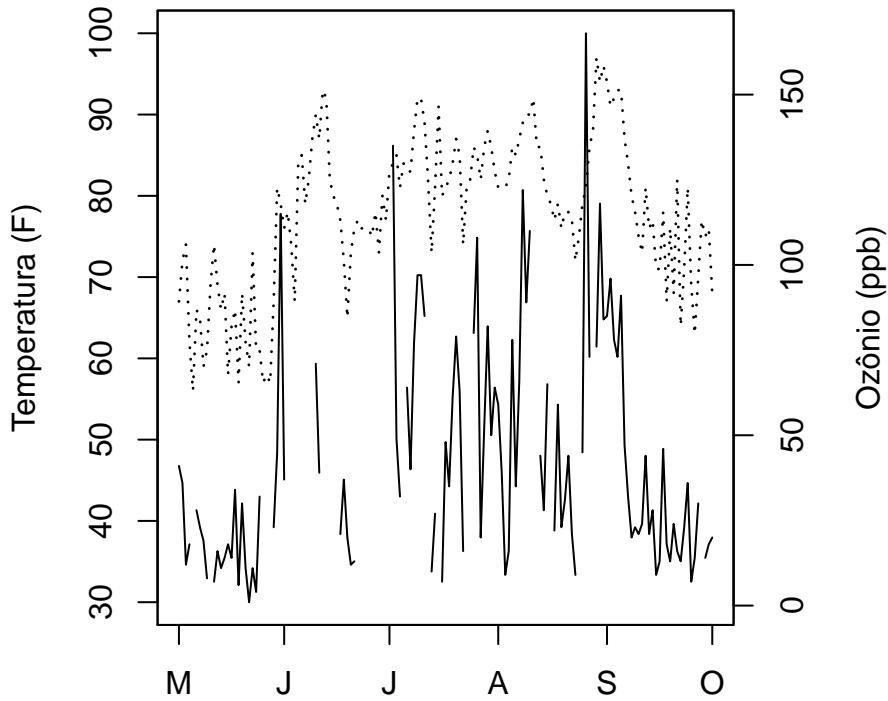
op <- par(new = TRUE)

plot(dados$Ozone,
      type = "l",
      xlab = "", ylab = "",
      yaxt = "n", xaxt = "n",
      lty = 1)

axis(side = 4, las = 3)

mtext(text = "Ozônio (ppb)",
      side = 4,
```

```
las = 0,
line = 3)
```



O título e subtítulo podem ser adicionados ao gráficos usando a função *title*.

Argumentos:

**line** - Linha para mostrar texto. Valores próximo a 1 corresponde a posição próximo ao eixo.  
**cex** - Tamanho do texto.

```
op <- par(mar = c(3, 4, 4, 4))

plot(dados$Temp, type = "l",
      lty = 3,
      ylim = c(30, 100),
      lwd = 1.4,
      xaxt = "n",
      ylab = "Temperatura (F)", xlab = "")

axis(side = 1,
     at = c(1, cumsum(table(dados[, 5]))),
     label = c("M", "J", "J", "A", "S", "O"))

op <- par(new = TRUE)

plot(dados$Ozone,
      type = "l",
      xlab = "", ylab = "",
      yaxt = "n", xaxt = "n",
      lty = 1)

axis(side = 4, las = 3)
```

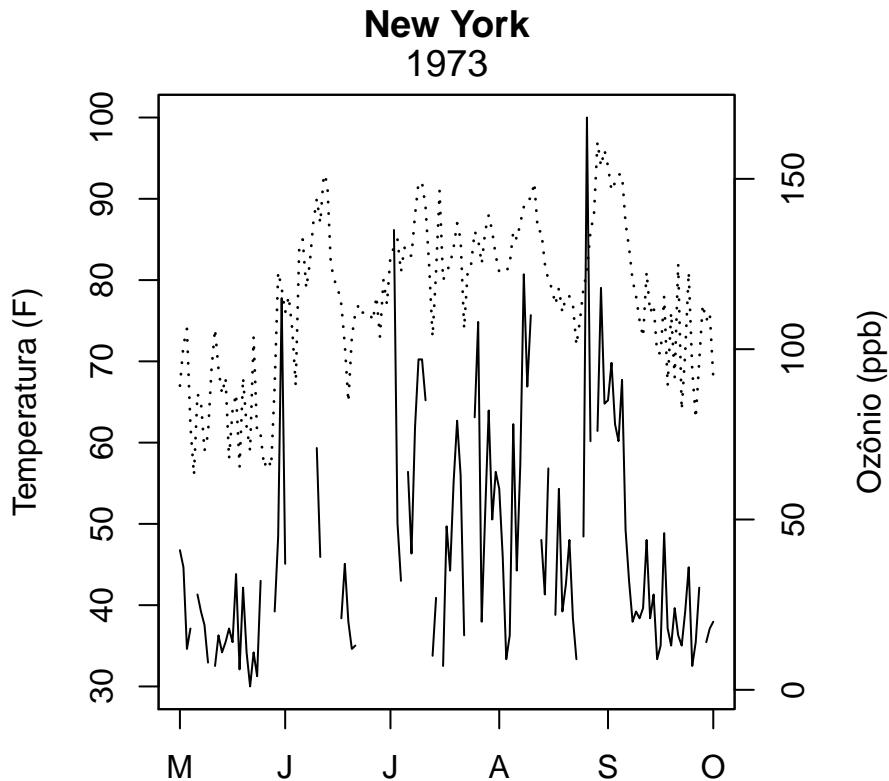
```

mtext(text = "Ozônio (ppb)",
      side = 4,
      las = 0,
      line = 3)

title(main = "New York", line = 1.5)

title(main = "1973", line = 0.5,
      cex = 0.8,
      font.main = 1)

```



Para finalizar pode-se adicionar legenda ao gráfico.

Argumentos:

**legend** - Especificar o texto para a legenda.

**lty** e **lwd** - Especificar tipo e espessura das linhas de cada nível.

**bty** - Especificar tipo de borda da legenda.

**ncol** - Especificar número de colunas para a legenda.

```

op <- par(mar = c(3, 4, 4, 4))

plot(dados$Temp, type = "l",
      lty = 3,
      ylim = c(30, 100),
      lwd = 1.4,
      xaxt = "n",
      ylab = "Temperatura (F)", xlab = "")

axis(side = 1,

```

```

at = c(1, cumsum(table(dados[, 5]))),
label = c("M", "J", "J", "A", "S", "O"))

op <- par(new = TRUE)

plot(dados$Ozone,
      type = "l",
      xlab = "", ylab = "",
      yaxt = "n", xaxt = "n",
      lty = 1)

axis(side = 4, las = 3)

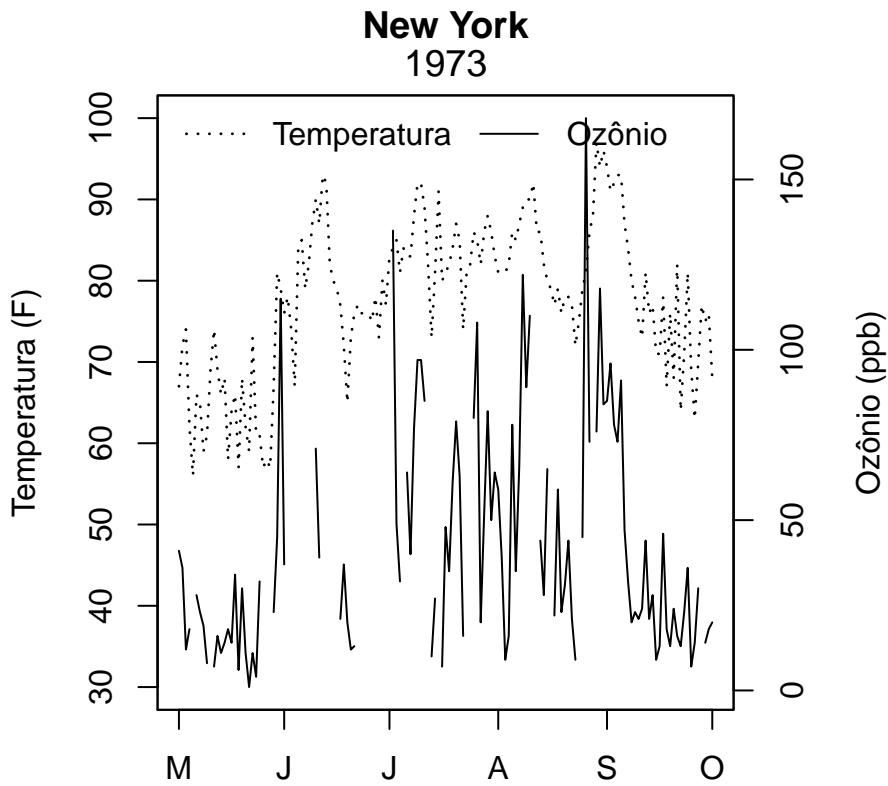
mtext(text = "Ozônio (ppb)",
      side = 4,
      las = 0,
      line = 3)

title(main = "New York", line = 1.5)

title(main = "1973", line = 0.5,
      cex = 0.8,
      font.main = 1)

legend("topleft",
       legend = c("Temperatura", "Ozônio"),
       lty = c(3, 1),
       lwd = c(1.4, 1),
       bty = "n",
       ncol = 2)

```



```
par(op) # Resetar parâmetros gráficos
```

## 4.11 Histogramas

### 4.11.0.1 Dados

Dados do tempo de espera entre erupções e a duração da erupção do gêiser Old Faithful em Yellowstone National Park, Wyoming, EUA.

```
data(faithful)
dados <- faithful
str(dados)
'data.frame': 272 obs. of 2 variables:
 $ eruptions: num 3.6 1.8 3.33 2.28 4.53 ...
 $ waiting   : num 79 54 74 62 85 55 88 85 51 85 ...
```

### 4.11.0.2 Plot

O histograma é construído pela função *hist*.

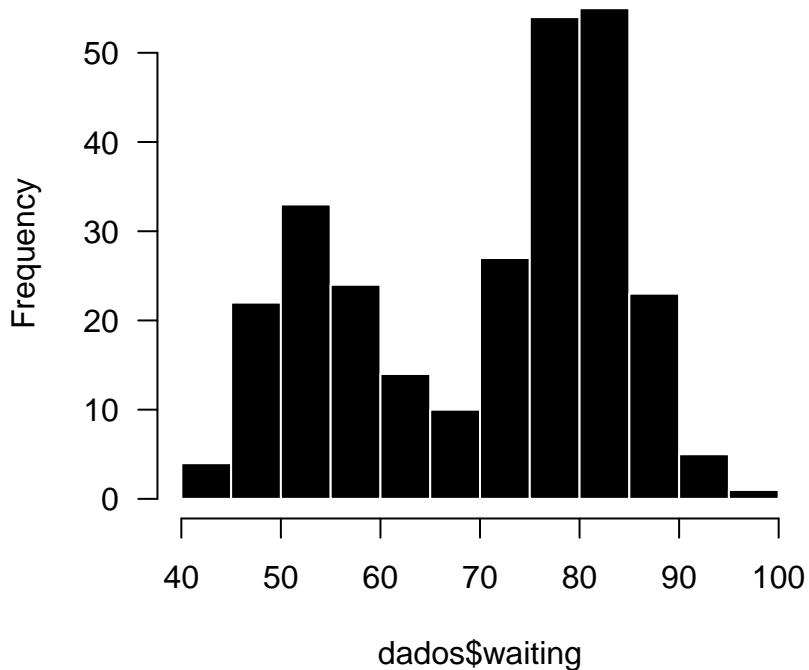
Argumentos:

**col** - Cor para as barras.

**border** - Cor para as bordas da barra.

```
hist(dados$waiting,
  las = 1,
  col = "black",
  border = "white")
```

## Histogram of dados\$waiting

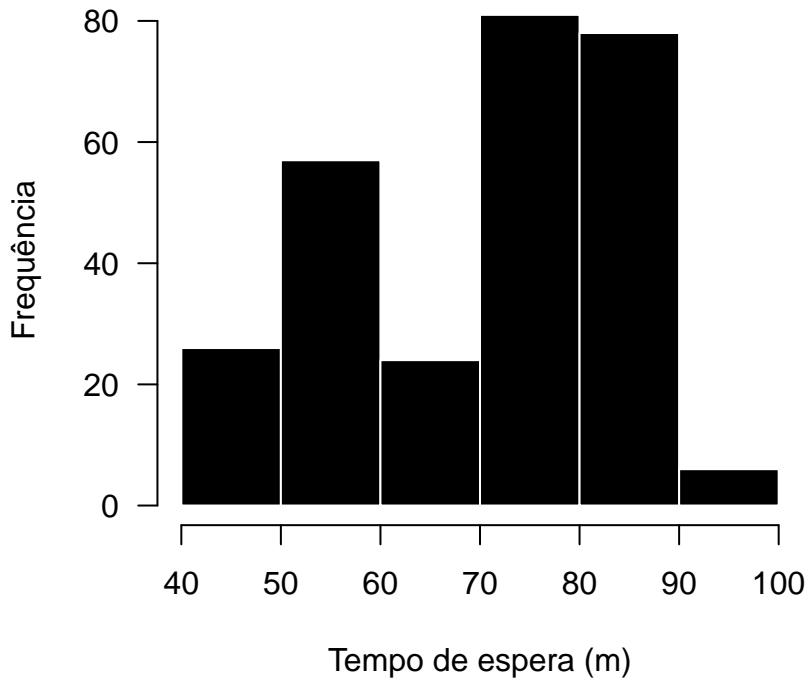


Pode-se alterar o critério dos limites para cada barra do histograma.

Argumento:

**breaks** - Especificar os limites de para o agrupamento dos valores em cada barra.

```
hist(dados$waiting,
  las = 1,
  col = "black",
  border = "white",
  breaks = c(40, 50, 60, 70, 80, 90, 100),
  xlab = "Tempo de espera (m)", ylab = "Frequência",
  main = "")
```



Pode-se ainda fazer com que os eixos fiquem juntos. Para isso é preciso definir o método de calcular os limites do eixo antes de fazer o histograma na função *par*. Além disso é preciso usar a função *box* para adicionar um contorno ao gráfico.

Argumentos:

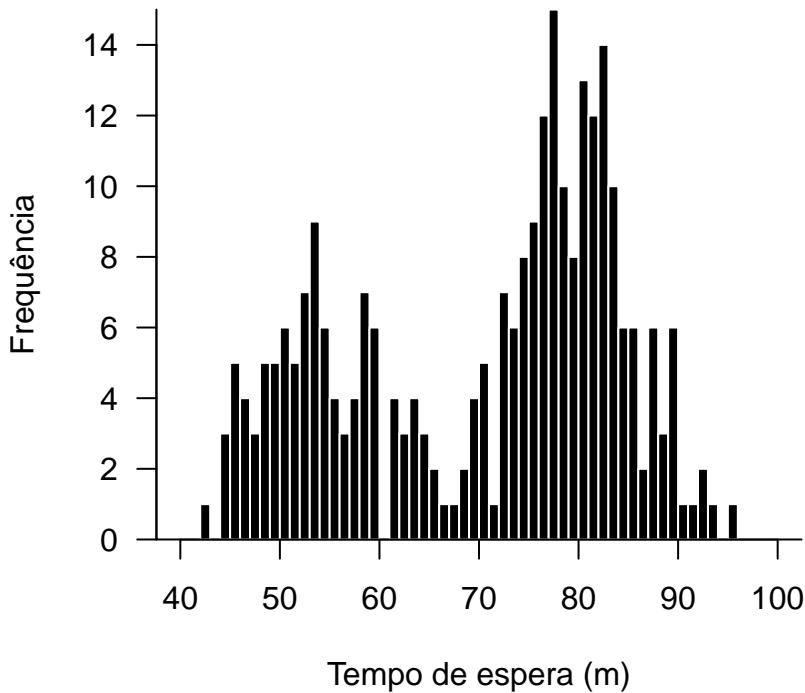
- yaxs** - Alterar o estilo de cálculo dos limites dos eixos.
- bty** - Altera o tipo de contorno usado envolta do gráfico.

```
op <- par(yaxs = "i")

hist(dados$waiting, las = 1,
      col = "black",
      border = "white",
      breaks = c(40:100),
      xlab = "Tempo de espera (m)", ylab = "Frequência",
      main = "Old Faithful")

box(bty = "1")
```

## Old Faithful



```
par(op) # Resetar parâmetros gráficos
```

A função *hist* permite também mostrar a densidade dos valores, não apenas a contagem de observações em cada intervalo. Adicionalmente pode-se adicionar a linha de densidade sobre o histograma.

Argumentos:

**prob** - Argumento lógico para especificar se densidade deve ser mostrada.

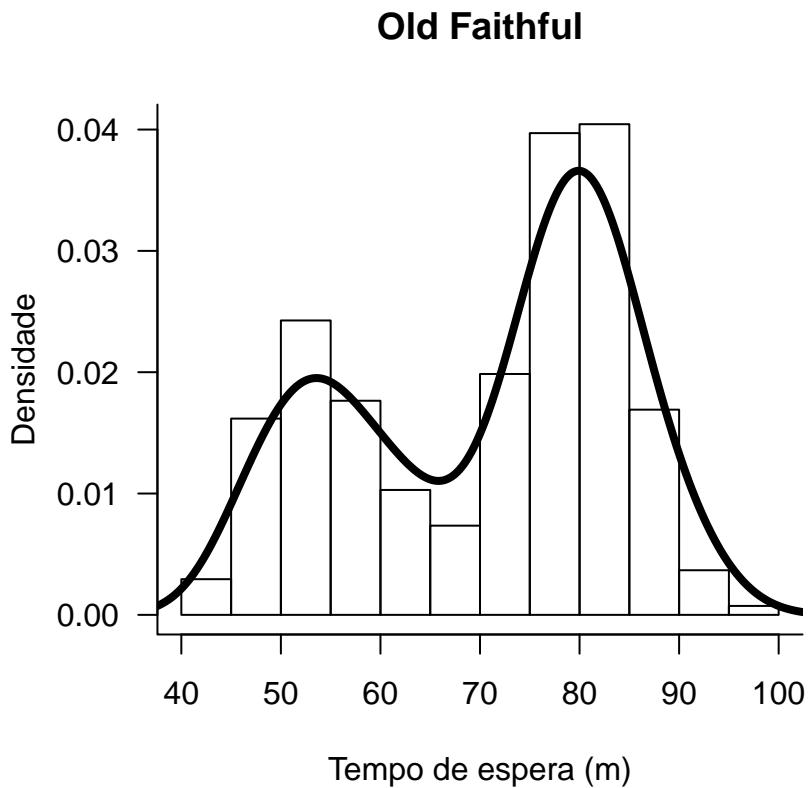
**lwd** - Espessura da linha.

**bty** - Tipo de contorno para o gráfico.

```
hist(dados$waiting,
  prob = TRUE,
  las = 1,
  xlab = "Tempo de espera (m)", ylab = "Densidade",
  main = "Old Faithful")

lines(density(dados$waiting),
      lwd = 4)

box(bty = "1")
```



## 4.12 Pares gráficos

### 4.12.0.1 Dados

Dados *iris* descrevem três espécies de iris com medidas do comprimento e largura de pétalas e sépalas.

```
data(iris)
dados <- iris[, 1:4]
str(dados)
'data.frame': 150 obs. of 4 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
```

### 4.12.0.2 Plot

A função *pairs* permite visualizar relações entre variáveis. O painel resultante será composto por tantas linhas e colunas de gráficos quanto variáveis. Adicionalmente pose-se construir funções para adicionar elementos específicos na diagonal, painéis inferiores ou superiores. Abaixo são mostrados dois exemplos modificados das funções mostrada da ajuda da função *pairs*.

```
panel.cor <- function(x, y, ...) {
  par(usr = c(0, 1, 0, 1))
  res <- cor.test(x, y, ...)
  txt <- as.character(round(res$estimate, 2))
  txt2 <- paste("p =", round(res$p.value, 3))
  text(0.5, 0.7, txt, cex = 2)
  text(0.5, 0.25, txt2, cex = 1.5)
}
```

```

panel.hist <- function(x, ...) {
  usr <- par("usr")
  on.exit(par(usr))
  par(usr = c(usr[1:2], 0, 1.5))
  h <- hist(x, plot = FALSE)
  breaks <- h$breaks
  nB <- length(breaks)
  y <- h$counts
  y <- y/max(y)
  rect(breaks[-nB], 0, breaks[-1], y, col = "black", border = "white", ...)
}

```

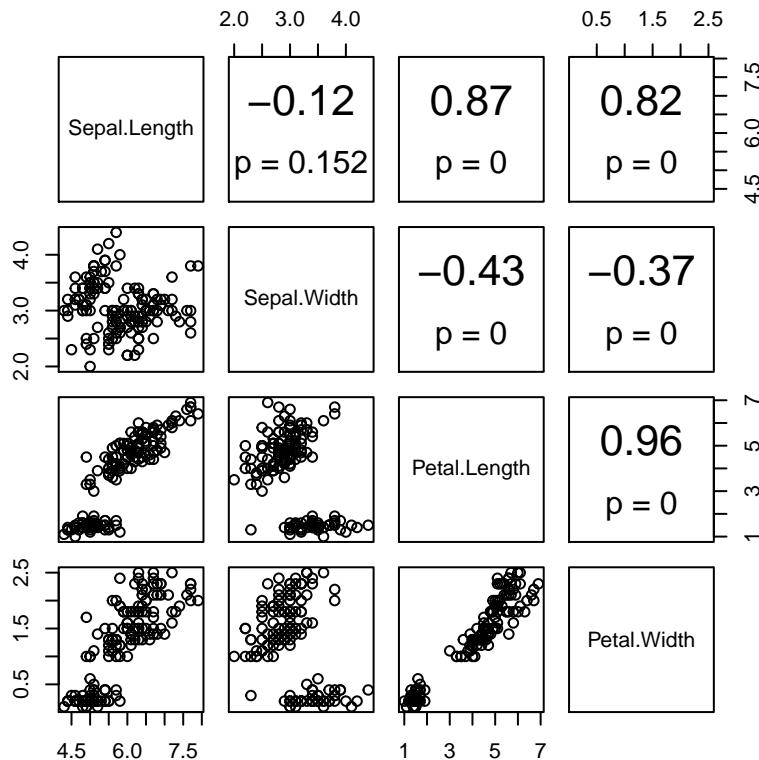
Argumento:

**upper.panel** - Função usada para os gráficos do painel superior.

```

pairs(dados,
      upper.panel = panel.cor)

```



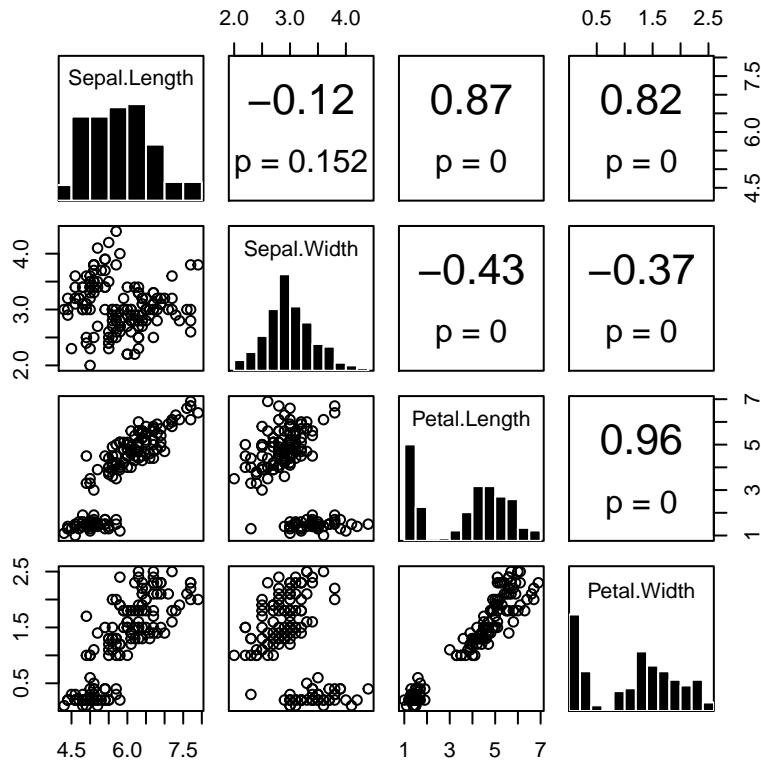
Argumento:

**diag.panel** - Função usada para os gráficos da diagonal do painel.

```

pairs(dados,
      upper.panel = panel.cor,
      diag.panel = panel.hist)

```

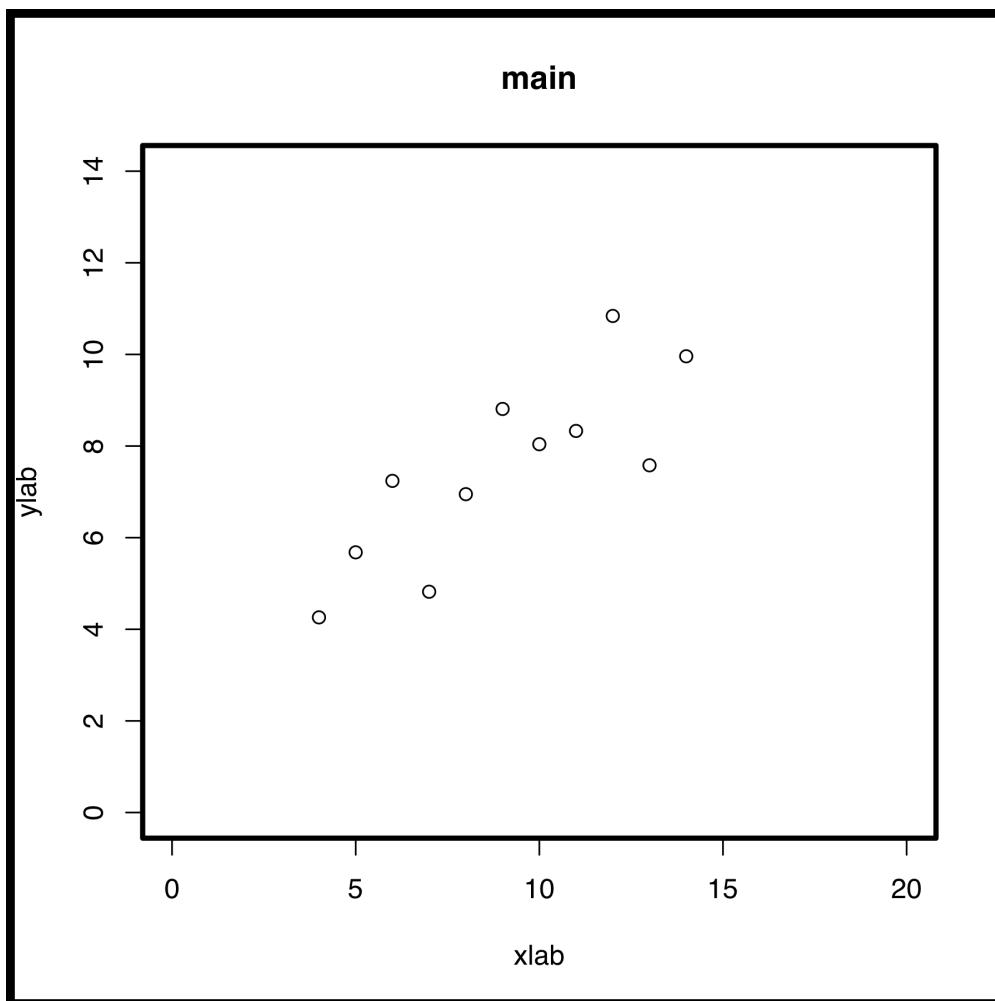


## 5 Painéis gráficos

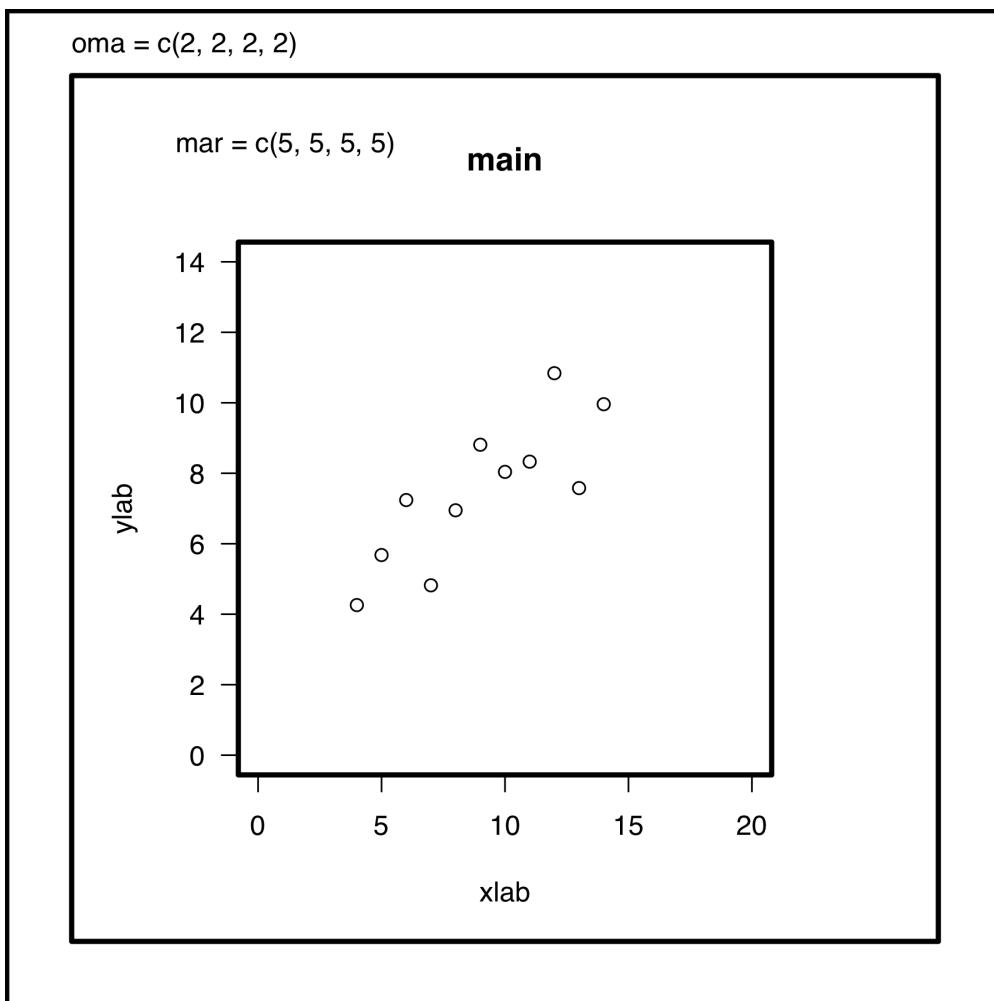
Um dos recursos disponíveis no R é a possibilidade de construir vários gráficos agrupados lado a lado. Estes podem ser exportados juntos formando uma mesma figura. É preciso entender o funcionamento das margens do dispositivo para personalizar o layout dos gráficos múltiplos.

### 5.1 Margens

Um gráfico no R é composto por vários elementos, na figura abaixo são mostrados os dois elementos principais de um gráfico. O primeiro deles é a área central chamada de área do gráfico, onde encontram-se os pontos, até os eixos. O segundo elemento é a margem da figura, que é definida entre os eixos e a borda exterior. O argumento *mar* deve ser especificado antes de começar a construção do gráfico, na função *par*. Esse argumento é um vetor numérico que indica o número de linhas em cada uma das margens, segundo a forma *c(margem inferior, margem à esquerda, margem superior, margem à direita)*. O padrão é *mar = c(5.1, 4.1, 4.1, 2.1)*, sendo que nessa configuração padrão a margem esquerda é bem menor que os demais.



Adicionalmente é possível definir um terceiro elemento usando o argumento *oma* na função *par*. Este argumento define o tamanho da margem externa da figura como um todo. Da mesma forma que o argumento *mar*, o argumento *oma* é um vetor numéricico que indica o número de linhas em cada uma das margens externas a figura, segundo a forma *c*(margem inferior, margem à esquerda, margem superior, margem à direita). O padrão de *oma* = *c*(0, 0, 0, 0). A figura abaixo mostra o resultado da alteração dos parâmetros *mar* e *oma* em um gráfico.



## 5.2 Configurar painéis

Uma maneira simples de configurar os painéis é usar os argumentos *mfrow* e *mfcol* na função *par*. Estes argumentos devem ser especificadas antes de começar a construção dos gráficos. Os dois argumentos funcionam da mesma maneira, fornecendo um vetor da forma de *c(nr, nc)*, sendo que *nr* especifica o número de linhas e *nc* especifica o número de colunas de gráficos no dispositivo. Por exemplo, se fornecido *c(4, 2)* o painel será composto por oito gráficos. A diferença entre *mfrow* e *mfcol* é que no primeiro os gráficos são construídos pelas linhas, da esquerda para a direita e depois passam para a segunda linha. Ja se o argumento *mfcol* for usado os gráficos são construídos pelas colunas, antes a primeira coluna e depois a segunda coluna. Alternativamente pode-se usar as funções *layout* e *split.screen* para produzir outros layout gráficos. O exemplo abaixo ilustra o funcionamento da função *mfrow* sem alterar os argumentos *mar* e *oma*. As linhas da borda da figura e os limites de cada gráfico foram destacadas para ilustrar o funcionamento das margens. Originalmente essas linhas não são mostradas.

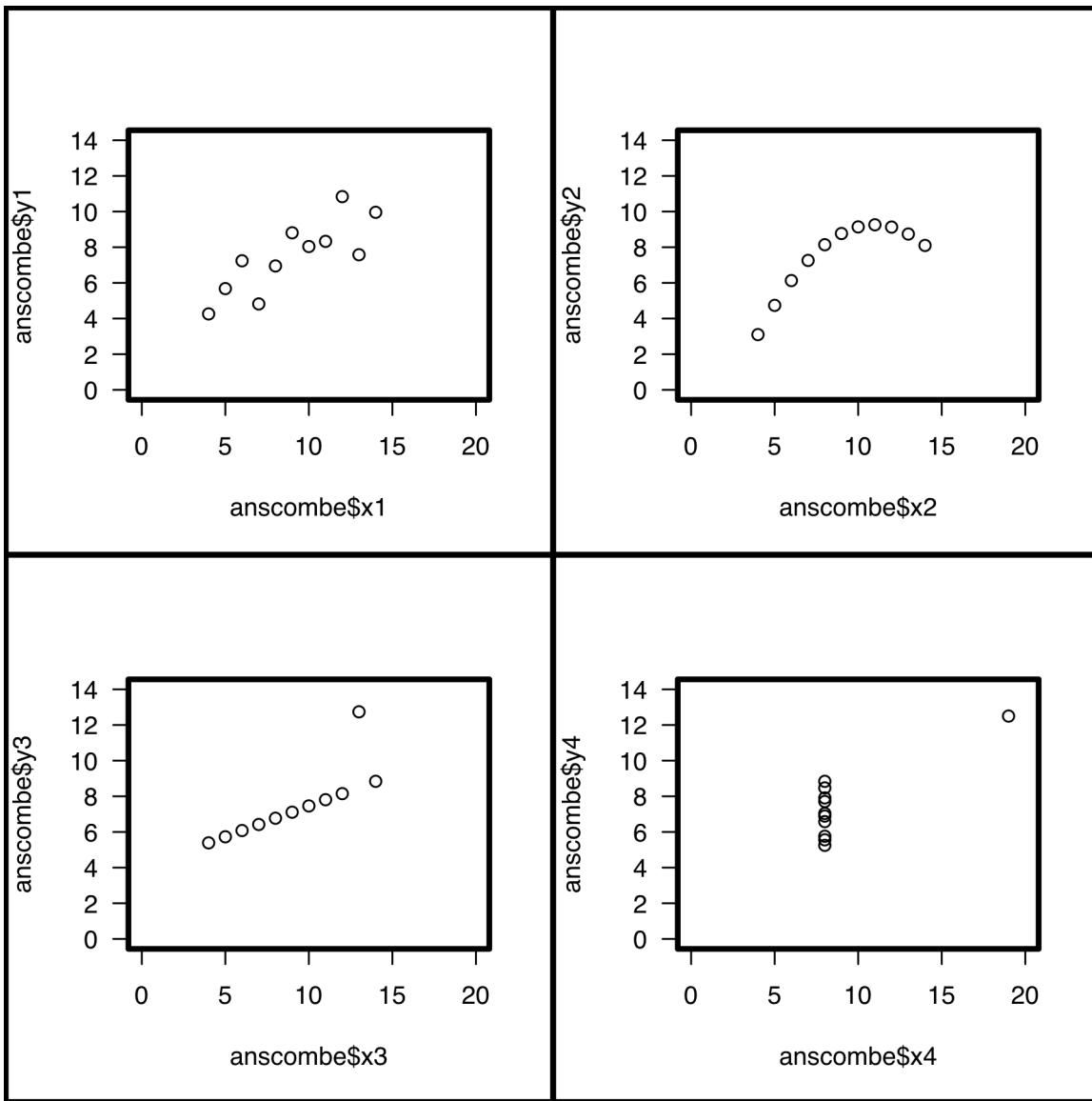
```

op <- par(mfrow = c(2, 2),
          las = 1)

plot(anscombe$x1, anscombe$y1, xlim = c(0, 20), ylim = c(0, 14))
plot(anscombe$x2, anscombe$y2, xlim = c(0, 20), ylim = c(0, 14))
plot(anscombe$x3, anscombe$y3, xlim = c(0, 20), ylim = c(0, 14))
plot(anscombe$x4, anscombe$y4, xlim = c(0, 20), ylim = c(0, 14))

```

```
par(op) # Restaurar parâmetros gráficos originais.
```

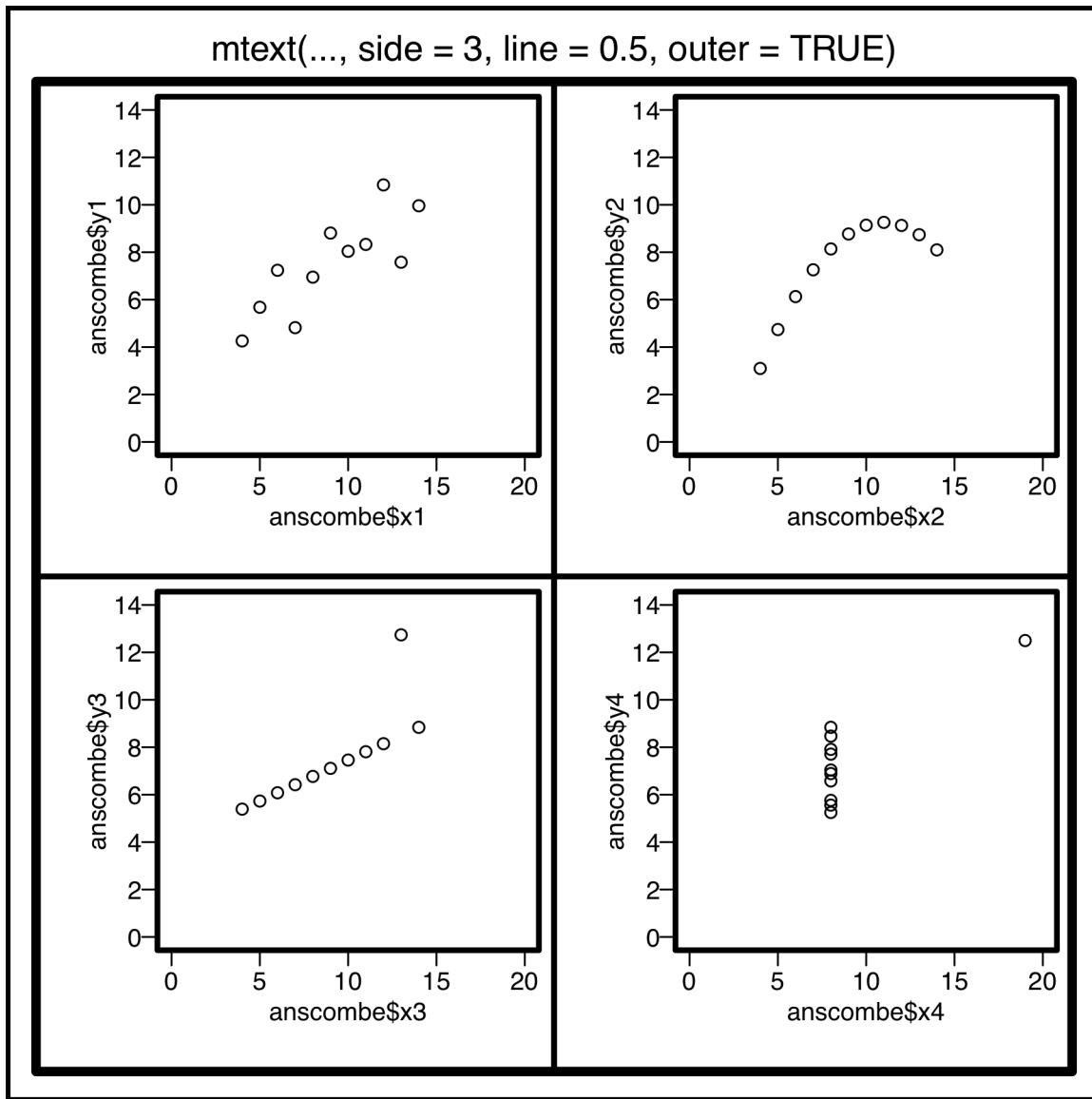


Na figura acima o espaço entre os gráficos poderia ser melhor aproveitado se as margens dos gráficos fossem alteradas. Abaixo segue o mesmo exemplo alterando os argumentos *mar* e *oma*. Novamente as linhas que limitam os gráficos foram destacadadas para visualização. A função *mtext* pode ser usada para adicionar um título geral para o conjunto de gráficos.

```
op <- par(mfrow = c(2, 2),
          las = 1,
          mar = c(4, 4, 0.5, 0.5),
          oma = c(1, 1, 2.5, 1))

plot(anscombe$x1, anscombe$y1, xlim = c(0, 20), ylim = c(0, 14))
plot(anscombe$x2, anscombe$y2, xlim = c(0, 20), ylim = c(0, 14))
plot(anscombe$x3, anscombe$y3, xlim = c(0, 20), ylim = c(0, 14))
plot(anscombe$x4, anscombe$y4, xlim = c(0, 20), ylim = c(0, 14))
```

```
par(op) # Restaurar parâmetros gráficos originais.
```



### 5.3 Subgráficos

Outro recurso disponível é sobrepor gráficos para criar um subgráfico. Para fazer isso é preciso alterar os argumentos *fig* e *new* na função *par*. Da mesma forma que os outros argumentos que controlam as opções gerais da figura o argumento *fig* é um vetor numérico que indica as coordenadas externas da figura, segundo a forma *c(x1, x2, y1, y2)*, sendo o padrão *fig = c(0.25, 0.50, 0.60, 0.80)*. Quando *new = TRUE* o próximo gráfico é construído sobreposto ao gráfico antigo. O argumento *mar* pode ser usado para especificar margens do gráfico, nessa caso as margens foram removidas.

```
curve(dnorm(x, 20, 1),
      from = 10, to = 25,
      ylab = "Densidade probabilística", xlab = "Valor de x",
      main = "Distribuição Normal - Média 20; SD = 1",
      cex.main = 0.8)
```

```

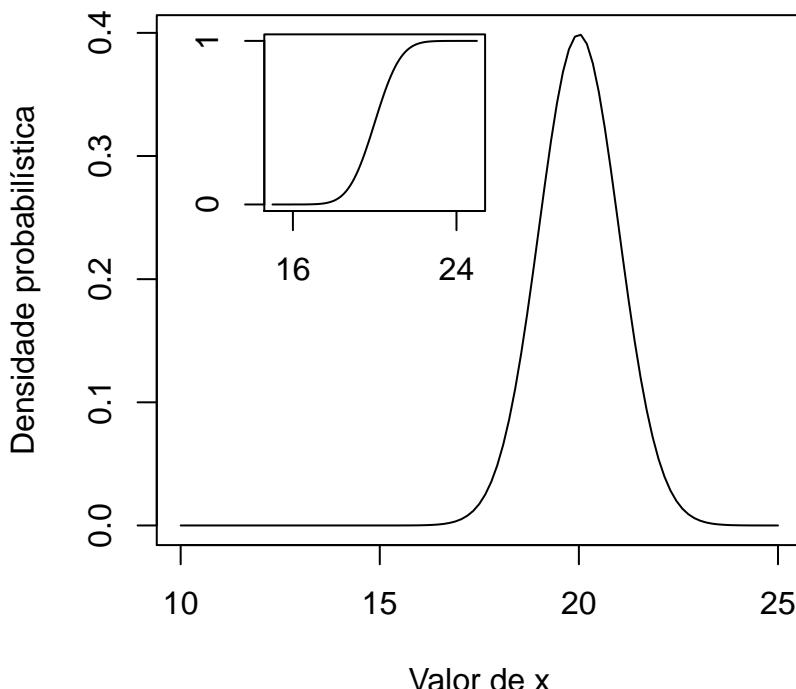
op <- par(fig = c(0.3, 0.55, 0.6, 0.8),
           new = TRUE,
           mar = c(0, 0, 0, 0))

curve(pnorm(x, 20, 1),
      from = 15, to = 25,
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n")

axis(1, at = c(16, 24))
axis(2, at = c(0, 1))

```

**Distribuição Normal – Média 20; SD = 1**



```
par(op) # Resetar parâmetros gráficos
```

## 5.4 Escala de cores

A função *image* pode ser utilizada para gerar grade de cores, como matrizes ou dados espaciais. Essa função também pode ser usada para adicionar legenda em escala de cores personalizada. A legenda é gerada na forma de subgráficos da gráfico principal. No exemplo abaixo os dados são duas variáveis numéricas, *x* e *y* representando as respectivas coordenadas de 50 observações. O primeiro passo é obter o mínimo e máximo da variável *x* e definir um número arbitrário de intervalos para a variável (no exemplo 100 intervalos, no objeto *intervalos*).

```

set.seed(100)
dados <- data.frame(x = runif(50))
dados$y <- cos(dados$x)
str(dados)

```

```

'data.frame':   50 obs. of  2 variables:
 $ x: num  0.3078 0.2577 0.5523 0.0564 0.4685 ...
 $ y: num  0.953 0.967 0.851 0.998 0.892 ...

# Definir 100 intervalos entre o mínimo e máximo da variável x
intervalos <- seq(min(dados$x, na.rm = TRUE),
                     max(dados$x, na.rm = TRUE),
                     length.out = 100)
str(intervalos)
num [1:100] 0.0564 0.0658 0.0752 0.0847 0.0941 ...

```

A seguir é definido um gradiente de cores para cada intervalo definido. As funções *heat.colors*, *colorRampPalette* e *two.colors* dos pacotes *fields* e *RColorBrewer* são exemplos de funções que podem ser usadas para definir os gradiente de cores. Ver seção Cores para mais informações. Após isso, a função *findInterval* é usada para categorizada a variável *x* nos intervalos definidos.

```

require(fields)
Loading required package: fields
Loading required package: spam
Loading required package: dotCall164
Loading required package: grid
Spam version 2.2-1 (2018-12-20) is loaded.
Type 'help( Spam)' or 'demo( spam)' for a short introduction
and overview of this package.
Help for individual functions is also obtained by adding the
suffix '.spam' to the function name, e.g. 'help( chol.spam)'.

Attaching package: 'spam'
The following objects are masked from 'package:base':

  backsolve, forwardsolve
Loading required package: maps
See www.image.ucar.edu/~nychka/Fields for
  a vignette and other supplements.

Attaching package: 'fields'
The following object is masked from 'package:AICcmodavg': 

  predictSE
The following object is masked from 'package:plotrix': 

  color.scale
require(RColorBrewer)
Loading required package: RColorBrewer

# Definir uma cor para cada um dos intervalos previamente definidos
cores <- two.colors(length(intervalos), start = "blue", end = "red", middle = "green")
str(cores)
chr [1:100] "#0000FF" "#0008F9" "#000FF2" "#0017EC" "#001FE5" ...

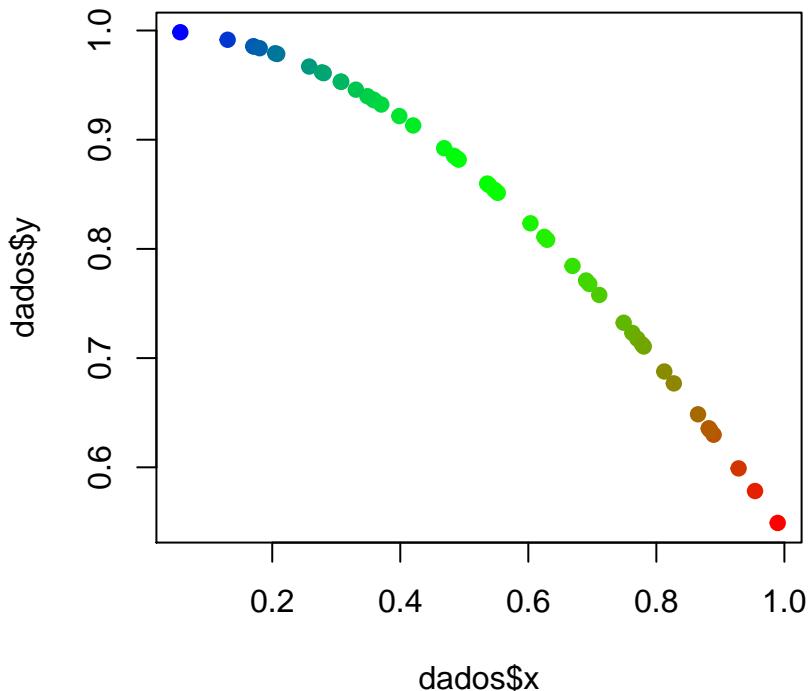
# Outros exemplos de funções para definir o gradiente de cores
# cores <- heat.colors(length(intervalos))
# cores <- colorRampPalette(brewer.pal(8, 'Oranges'))(length(intervalos))

```

```
# Variável x é categorizada em um dos intervalos previamente definidos
quaisintervalos <- findInterval(dados$x,
                                intervalos)
str(quaisintervalos)
int [1:50] 27 22 53 1 44 46 81 34 53 13 ...
```

Próximo passo é construir o gráfico organizando o vetor de cores conforme o intervalo que a variável  $x$  pertence.

```
plot(dados$x, dados$y,
      pch = 19,
      col = cores[quaisintervalos])
```



A legenda pode então ser adicionada ao gráfico com a função *image*. A legenda será adicionada na forma de um subgráfico, para isso é definido uma nova área para no novo gráfico usando a função *par* alterando os argumentos *fig*, *new* e *mar*. Ver mais detalhes na seção Subgráficos acima. As opções de cores e intervalos definidas anteriormente são usadas para a construção da legenda. Próximo bloco de código mostra as opções para a legenda na orientação vertical.

```
plot(dados$x, dados$y,
      pch = 19,
      col = cores[quaisintervalos])

# A legenda ocupará toda a área definida pelo argumento fig
op <- par(fig = c(0.82, 0.85, 0.5, 0.75),
           new = TRUE,
           mar = c(0,0,0,0))

# Opções para escala de cores na orientação vertical.
# Ver próximo bloco de código para orientação horizontal.
# As próximas 7 linhas do comando não precisam ser alteradas para uma legenda vertical
```

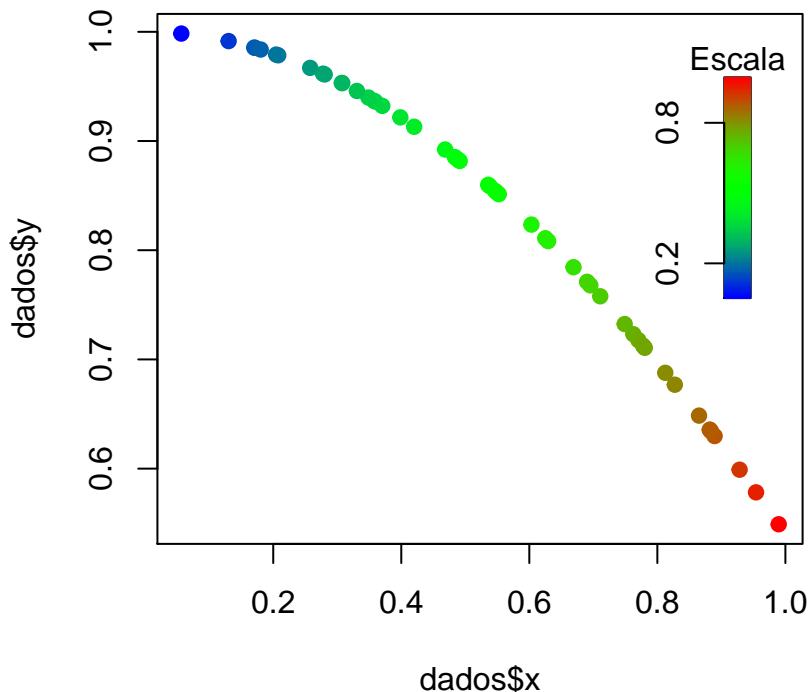
```

nl <- length(intervalos)
nc <- 1
image(nc, intervalos, matrix(intervalos, nc, nl),
      col = cores,
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      add = FALSE)

# Adicionar eixos
axis(2, at = c(0.2, 0.8))

# Adicionar título. Como a margem foram removidas, a função mtext é usada neste caso.
mtext("Escala")

```



```
par(op) # Resetar parâmetros gráficos
```

Agora a legenda é adicionada na orientação horizontal.

```

plot(dados$x, dados$y,
      pch = 19,
      col = cores[quaisintervalos])

op <- par(fig = c(0.2, 0.5, 0.32, 0.35),
           new = TRUE,
           mar = c(0,0,0,0))

# Opções para escala de cores na orientação horizontal.
# Ver bloco anterior de código para orientação vertical.
# As próximas 7 linhas do comando não precisam ser alteradas para uma legenda horizontal.

```

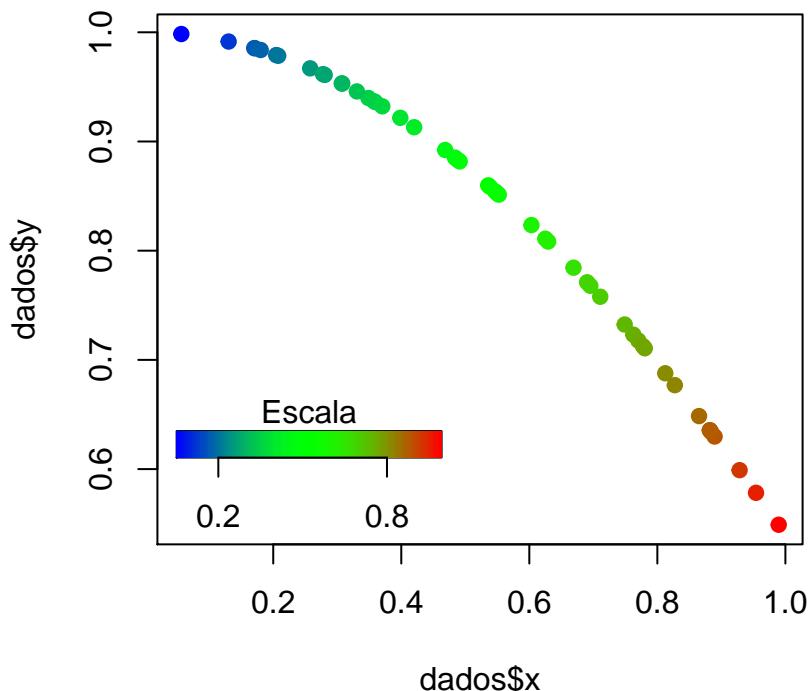
```

nl <- 1
nc <- length(intervalos)
image(intervalos, nc, matrix(intervalos, nc, nl),
      col = cores,
      xlab = "", ylab = "",
      xaxt = "n", yaxt = "n",
      add = FALSE)

# Adicionar eixos
axis(1, at = c(0.2, 0.8))

# Adicionar título. Como a margem foi removida, a função mtext é usada neste caso.
mtext("Escala")

```



```
par(op) # Resetar parâmetros gráficos
```

## 6 Exportar gráficos

Normalmente as figuras são geradas no dispositivos de visualização e posteriormente exportadas usando outro dispositivo. Os dispositivos para exportar são em dois grupos, o primeiro deles exporta arquivos de imagens que são dependentes de resoluções como `jpeg()`, `bmp()`, `tiff()` e `png()` e o segundo deles exporta arquivos vetoriais independentes de resoluções como `pdf()`, `svg()` e `postscript()`.

A estrutura básica para exportar é a seguinte:

```

pdf("Nome do arquivo.pdf")

# Comandos gráficos

dev.off()

```

No exemplo acima do dispositivo de exportação é iniciado pela função `pdf()` que gera um arquivo chamado *Nome do arquivo.pdf*. Os comandos gráficos são então executados e o gráfico é construído dentro do arquivo pdf. Se mais de um gráfico é gerado todos serão construídos dentro do mesmo arquivo. A função `dev.off()` é usada para finalizar o dispositivo de exportação, essa função é obrigatória para que o arquivo pdf possa ser aberto por outro aplicativo.

Para exportar um gráfico de exemplo pode-se utilizar o conjunto de dados *anscombe*.

```
data(anscombe)
dados <- anscombe
```

## 6.1 Exportar como imagem

No caso dos arquivos imagens pode-se exportar em usando a função `png()`. É preciso especificar as dimensões da figura e resolução da imagem que será exportada.

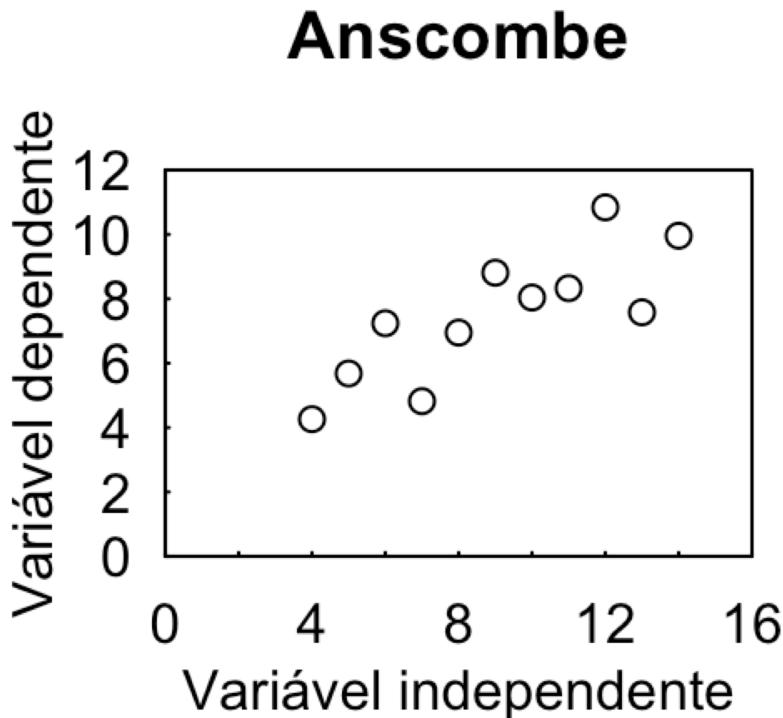
Argumentos:

- res** - Resolução em ppi.
- height** - Altura em pixels.
- width** - Largura em pixels.

```
png("Anscombe.png",
  res = 200,
  height = 600,
  width = 600)

plot(dados$x1, dados$y1,
  xlab = "Variável independente", ylab = "Variável dependente",
  main = "Anscombe",
  las = 1,
  xlim = c(0, 16), ylim = c(0, 12),
  xaxs = "i", yaxs = "i",
  xaxp = c(0, 16, 8),
  yaxp = c(0, 12, 6),
  tck = 0.01,
  mgp = c(1.5, 0.5, 0))

dev.off()
```



A imagem exportada perde a formatação e o layout do dispositivo de visualização. Para reajustar é preciso alterar alguns parâmetros gráficos, como por exemplo, *mex*, *cex*, *cex.main*, *cex.lab* e *cex.axis*.

## 6.2 Exportar como imagem vetorial

Para exportar em arquivo vetorial a resolução não precisa ser ajustada, apenas as dimensões do arquivo de saída.

Argumentos:

**height** - Altura em polegadas.

**width** - Largura em polegadas.

```
pdf("Anscombe.pdf",
  width = 6,
  height = 6)

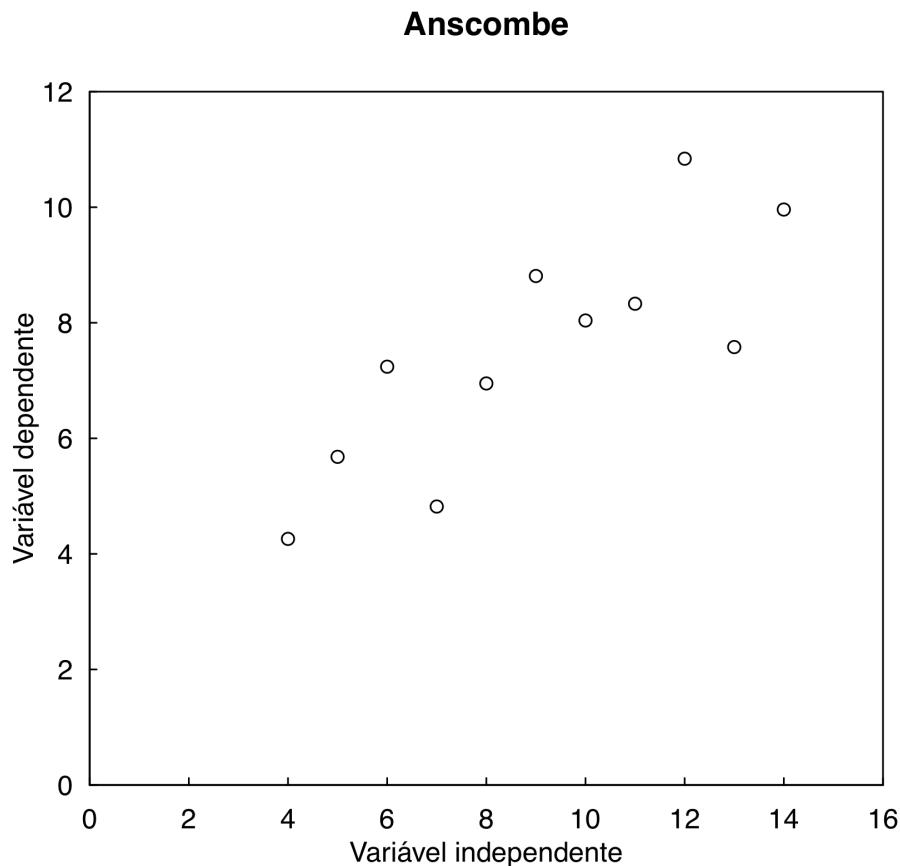
plot(dados$x1, dados$y1,
      xlab = "Variável independente", ylab = "Variável dependente",
      main = "Anscombe",
      las = 1,
      xlim = c(0, 16), ylim = c(0, 12),
      xaxs = "i", yaxs = "i",
      xaxp = c(0, 16, 8),
```

```

yaxp = c(0, 12, 6),
tck = 0.01,
mgp = c(1.5, 0.5, 0))

dev.off()

```



Nesse caso a formatação e o layout fica mais próximo ao observado no dispositivo de visualização. Ainda assim, é possível ajustar os parâmetros gráficos para um melhor resultado. A exportação no formato vetorial é recomendada, por ser mais fácil e por não perder a resolução ao aplicar zoom.

### 6.3 Função recordPlot()

É possível salvar o gráfico atual do dispositivo em um objeto. Isso permite renderizar novamente o mesmo gráfico e adicionar diferentes elementos a partir de um gráfico base ou salvar vários gráficos em diferentes objetos para posterior exportação. No exemplo as variáveis numéricas  $x$  e  $y$  são duas sequência de 1 a 10 representando as respectivas coordenadas dos pontos no gráfico.

```

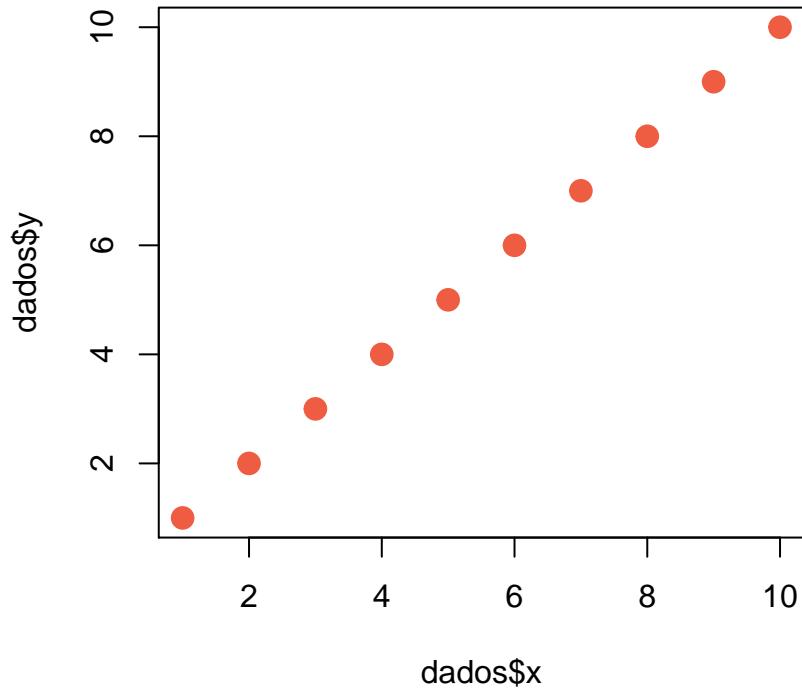
dados <- data.frame(x = 1:10, y = 1:10)
str(dados)
'data.frame':   10 obs. of  2 variables:
$ x: int  1 2 3 4 5 6 7 8 9 10
$ y: int  1 2 3 4 5 6 7 8 9 10

```

O gráfico é construído com todas as opções e posteriormente salvo com a função `recordPlot`.

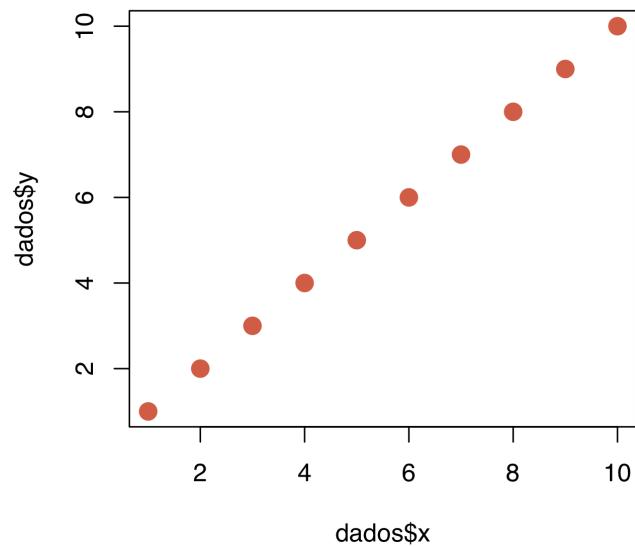
```
plot(dados$x, dados$y,  
      pch = 19,  
      col = "tomato2",  
      cex = 1.5)  
  
title(main = "Exemplo")
```

## Exemplo

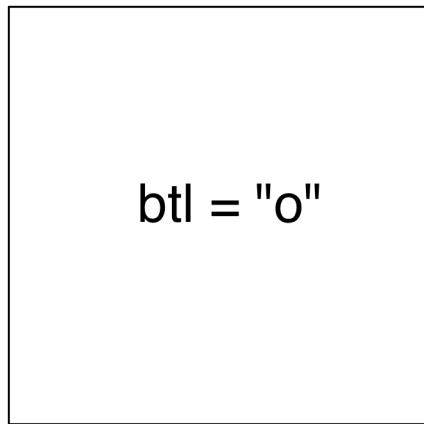


```
pl <- recordPlot() # Gráfico salvo no objeto pl  
pl # Gráfico originalmente salvo é repetido com todas as opções utilizadas
```

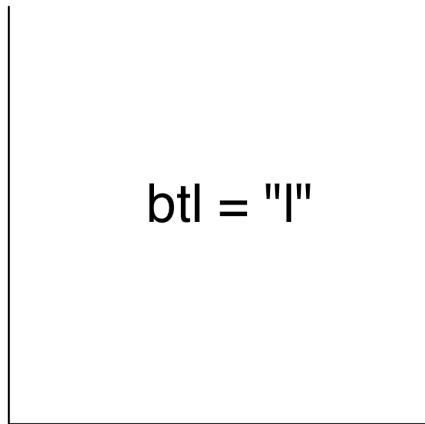
### Exemplo



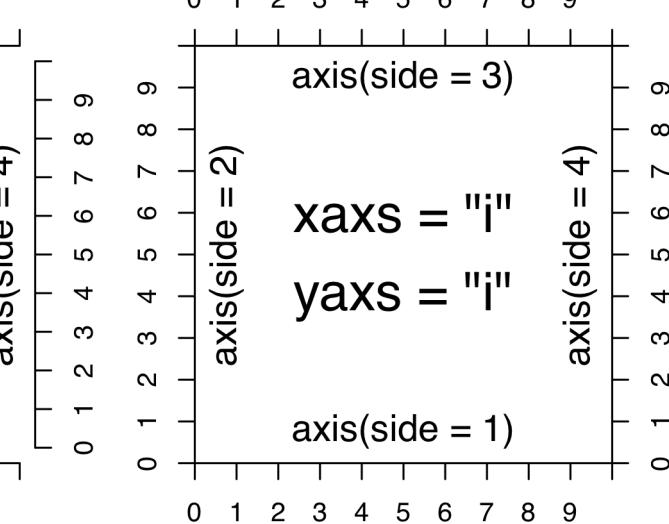
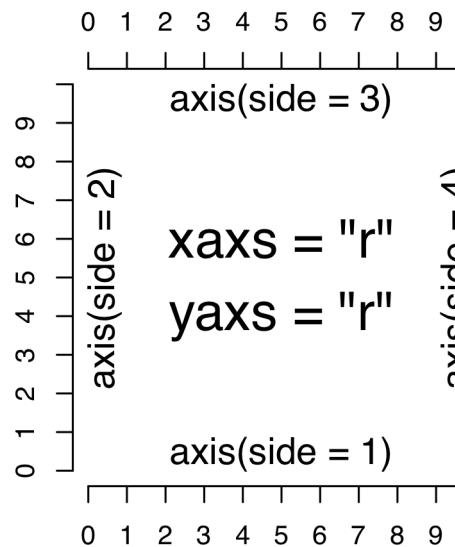
## 7 Guia de ajuda rápida



**btl = "o"**



**btl = "l"**



```

title(main = ...)

mtext(..., side = 3)
  mtext(..., side = 3, line = -1)
  mtext(..., side = 3, line = -2, adj = 0)

title(ylab = ...)

mtext(..., side = 2)
  
$$\int_a^b f(x)dx$$

  expression(integral(f(x) * dx, a, b))

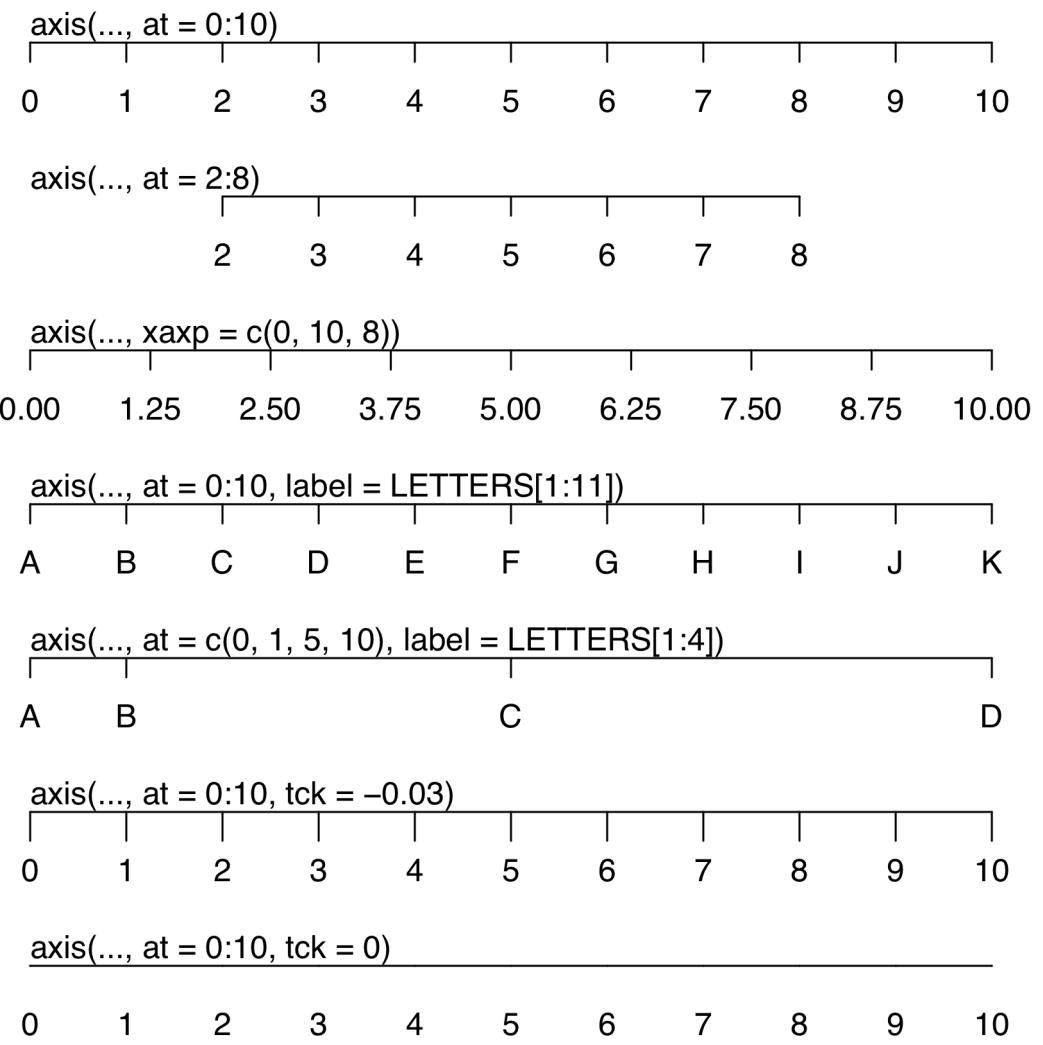
  text(x, y)
  text(x, y, adj = c(0, 0))

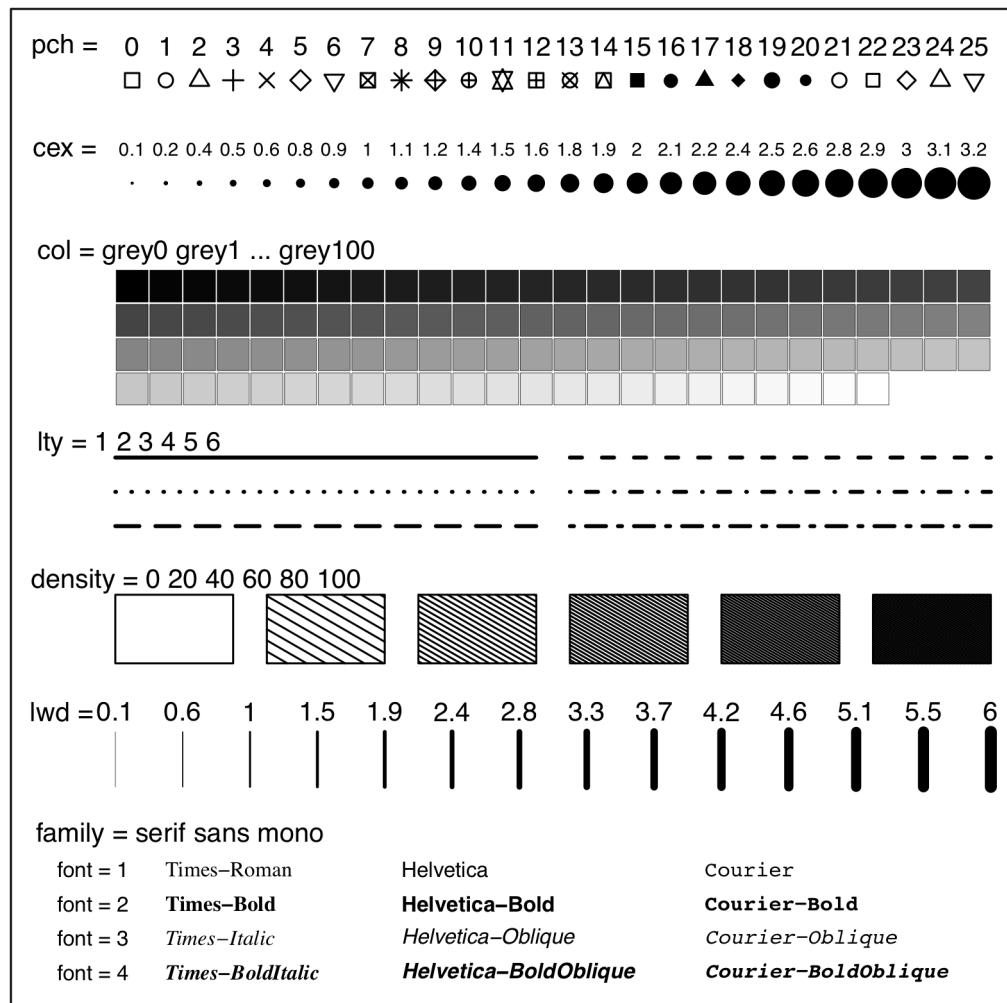
mtext(..., side = 1)

mtext(..., line = -1), side = 4)
  mtext(..., side = 4)

title(xlab = ...)

```





Objetivo do gráfico	Gráfico	Funções básicas
Rankeamento		<i>barplot</i>
Rankeamento		<i>dotchart</i>
Desvio		<i>barplot</i>
Desvio		<i>plot ou points</i>
Distribuição		<i>hist</i>
Distribuição		<i>boxplot</i>
Distribuição		<i>beanplot</i>
Distribuição		<i>density e plot</i>
Correlação		<i>image ou heatmap</i>
Parte do todo		<i>pie</i>
Parte do todo		<i>barplot</i>
Tendências no tempo		<i>plot e points</i>
Magnitude		<i>barplot</i>
Magnitude		<i>plotCI</i>
Relação entre variáveis		<i>plot e points</i>
Modelos estatísticos		<i>curve</i>

Elementos adicionais	Gráfico	Funções
Pontos		<i>points</i>
Linhas conectadas		<i>points ou lines</i>
Curvas		<i>curve</i>
Linhas simples		<i>abline</i>
Textos	Texto1 Texto2	<i>text</i>
Polígonos		<i>polygon</i>
Eixos		<i>axis</i>
Legendas		<i>legend</i>
Títulos	Eixo y Título	<i>title</i>
Setas		<i>arrows</i>
Segmentos		<i>segments</i>
Contorno do gráfico		<i>box</i>
Grid		<i>grid</i>
Localizar coordenadas	X achar o x	<i>locator</i>
Texto nas margens	Margem	<i>mtext</i>

## 8 Conclusão

O objetivo deste texto foi apresentar as principais funções e argumentos utilizados na construção de gráficos utilizando as funções gráficas básicas do R. Os principais elementos gráficos, opções de personalização e exportação foram mostradas e exemplificadas. Espero que este texto tenha sido útil e, por favor, avise-me se tiver dúvidas ou sugestões sobre este texto.

## 9 Mais informações

Outros textos e tutoriais sobre R podem ser encontrados em <https://vanderleidebastiani.github.io/tutoriais>.

## 10 Referências

- Chang, Winston. 2012. **R Graphics Cookbook**. O'Reilly Media Inc. Sebastopol, CA.
- Crawley, Michael J. 2007. **The R book**. John Wiley & Sons, Chichester.
- Vandemeulebroecke, Marc, 2018. **How can we make better graphs? An initiative to increase the graphical expertise and productivity of quantitative scientists**. <https://doi.org/10.1002/pst.1912>