

- [Tutorial](#)
- [Exercícios](#)
- [Apostila](#)

5a. Criação e Edição de Gráficos no R

Refletindo sobre a representação dos dados



Video

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

Gráficos na Tela

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

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

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

O sistema do pacote `graphics`



Video

Funções de Alto Nível

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

Criando Dados

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

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

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

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

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

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

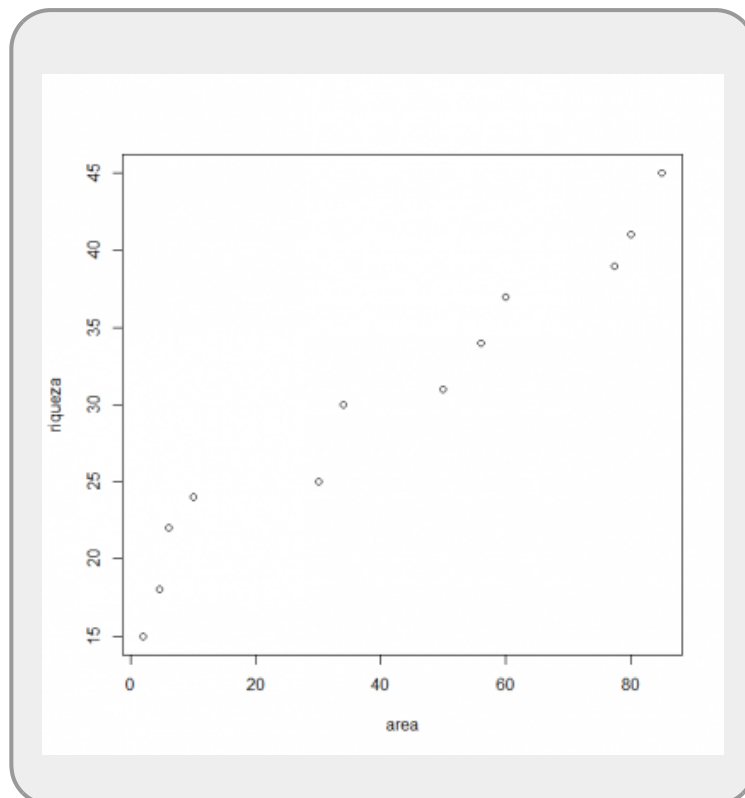
Criando Gráficos

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

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

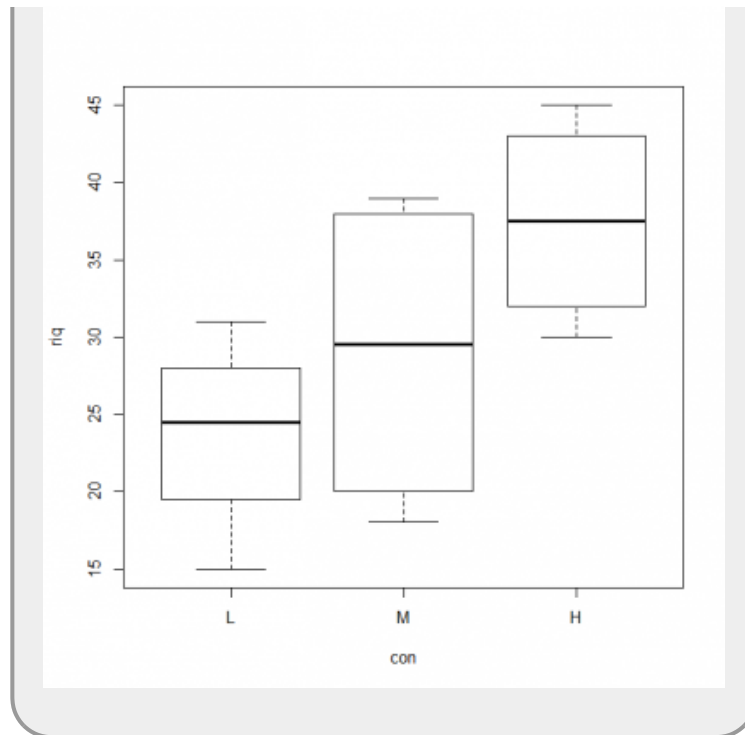
Abaixo podemos ver os gráficos associados a cada uma das variáveis preditoras com a saída padrão das funções `plot` e `boxplot`:

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



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





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

Editando Gráficos

Parâmetros Locais e Globais

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

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

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

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

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

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

```
help(par)
```

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

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

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

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

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

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

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

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

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

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

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

Parâmetros Vetorizados

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

CoRes

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

white	aliceblue	antiquewhite	antiquewhite1	antiquewhite2	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
antiquewhite3	antiquewhite4	aquamarine	aquamarine1	aquamarine2	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
aquamarine3	aquamarine4	azure	azure1	azure2	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
azure3	azure4	beige	bisque	bisque1	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84
bisque2	bisque3	bisque4		blanchedalmond	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105
blue	blue1	blue2	blue3	blue4	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126
blueviolet	brown	brown1	brown2	brown3	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147
brown4	burlywood	burlywood1	burlywood2	burlywood3	148	149	150	151	152																
burlywood4	cadetblue	cadetblue1	cadetblue2	cadetblue3	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173
cadetblue4	chartreuse	chartreuse1	chartreuse2	chartreuse3	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194
chartreuse4	chocolate	chocolate1	chocolate2	chocolate3	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215
chocolate4	coral	coral1	coral2	coral3	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231					
coral4	cornflowerblue	cornsilk	cornsilk1	cornsilk2	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252
cornsilk3	cornsilk4	cyan	cyan1	cyan2	253	254	255	256	257	258	259	260													
cyan3	cyan4	darkblue	darkcyan	darkgoldenrod	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281
darkgoldenrod1	darkgoldenrod2	darkgoldenrod3	darkgoldenrod4	darkgray	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302
darkgreen	darkgrey	darkkhaki	darkmagenta	darkolivegreen	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323
darkolivegreen1	darkolivegreen2	darkolivegreen3	darkolivegreen4	darkorange	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344
darkorange1	darkorange2	darkorange3	darkorange4	darkorchid	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365
darkorchid1	darkorchid2	darkorchid3	darkorchid4	darkred	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386
darksalmon	darkseagreen	darkseagreen1	darkseagreen2	darkseagreen3	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407
darkseagreen4	darkslateblue	darkslategray	darkslategray1	darkslategray2	408	409	410	411	412	413	414	415	416	417	418	419	420								
darkslategray3	darkslategray4	darkslategray5	darkturquoise	darkviolet	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441
deeppink	deeppink1	deeppink2	deeppink3	deeppink4	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462
deepskyblue	deepskyblue1	deepskyblue2	deepskyblue3	deepskyblue4	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483
					484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504
					505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524	525
					526	527	528	529	530	531	532	533	534	535	536	537	538	539	540	541	542	543	544	545	546
					547	548	549	550	551	552	553	554	555	556	557	558	559	560	561	562	563	564	565	566	567
					568	569	570	571	572	573	574	575	576	577	578	579	580	581	582	583	584	585	586	587	588
					589	590	591	592	593	594	595	596	597	598	599	600	601	602	603	604	605	606	607	608	609
					610	611	612	613	614	615	616	617	618	619	620	621	622	623	624	625	626	627	628	629	630
					631	632	633	634	635	636	637	638	639	640	641	642	643	644	645	646	647	648	649	650	651

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

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

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

Inserindo mais Informações em Gráficos

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

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

lines()

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

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

points()

Para inserir novos pontos no gráfico:

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

text()

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

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

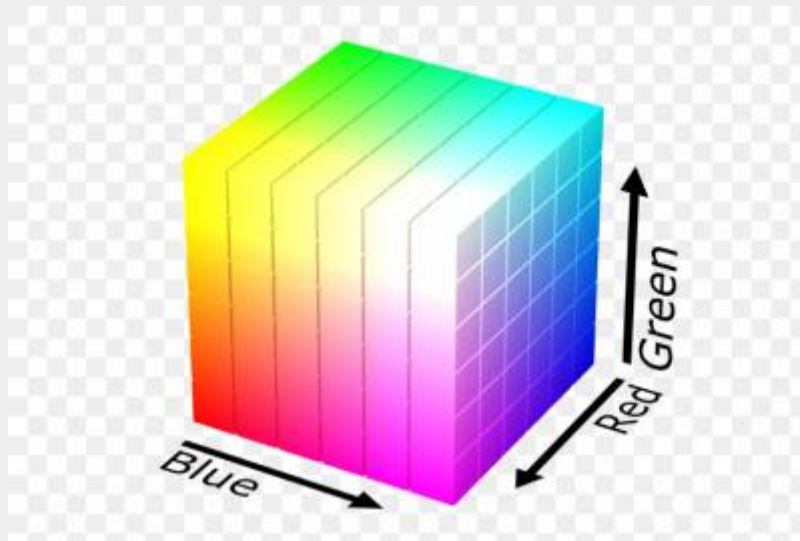
arrows(), rect(), polygon(), segments() ...

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

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

coRes

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



Ajustando o Gráfico

Um exemplo

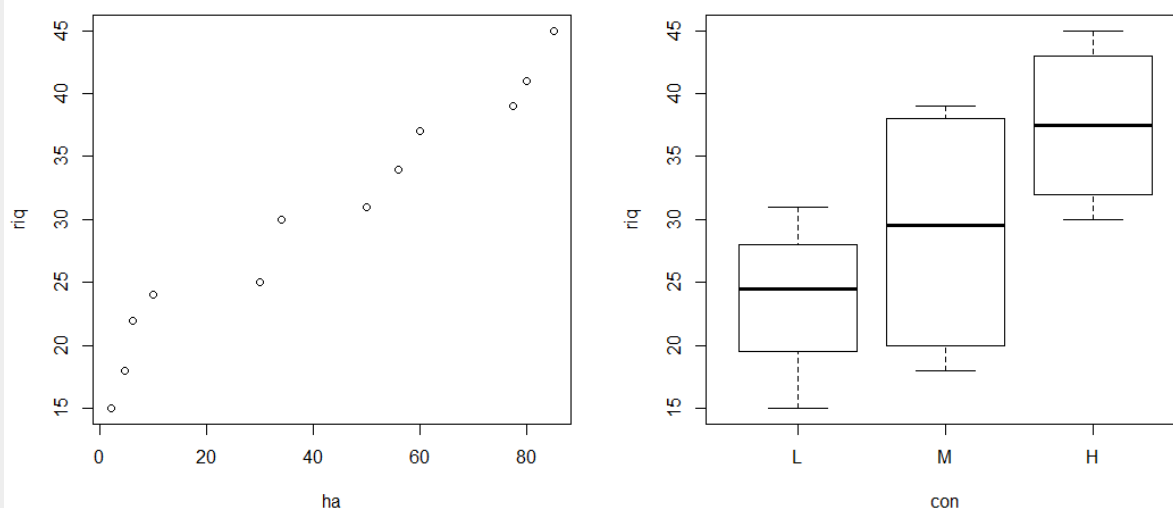


Video

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

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

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



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

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

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

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

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

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

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

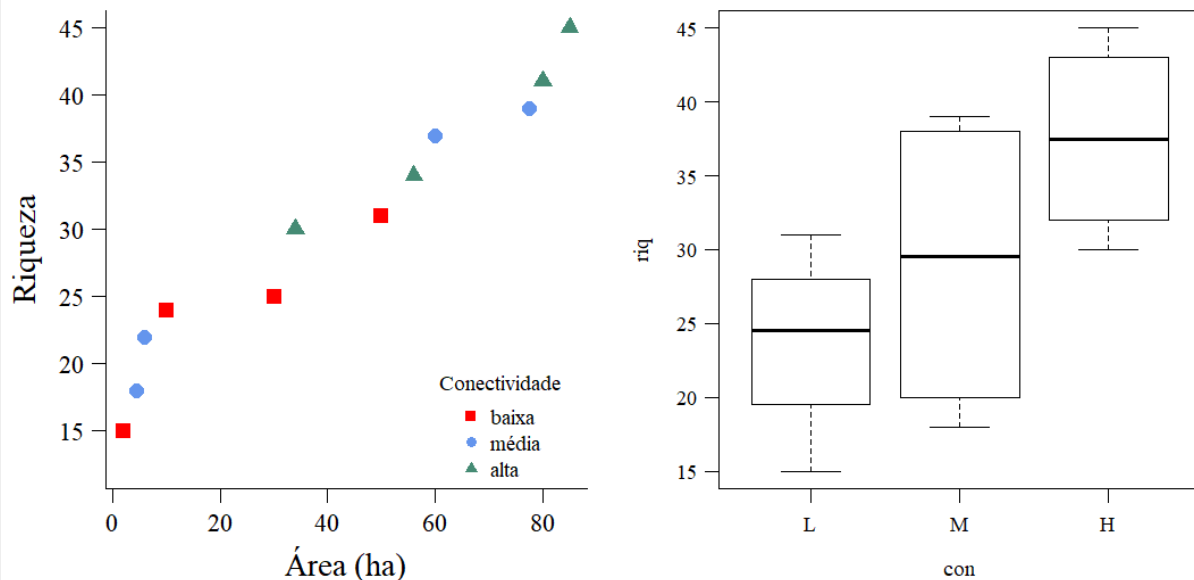
Note a indexação que foi feita no `pch` e no `col`. Para fornecer o mesmo número de símbolo e nome de cor para as observação de um mesmo nível de fator, fizemos a indexação desses parâmetros gráficos pelo fator conectada. Para entender, primeiro vamos investigar o que foi produzido nas indexações:

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

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

Agora vamos inserir uma legenda para as cores e símbolos:

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



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

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

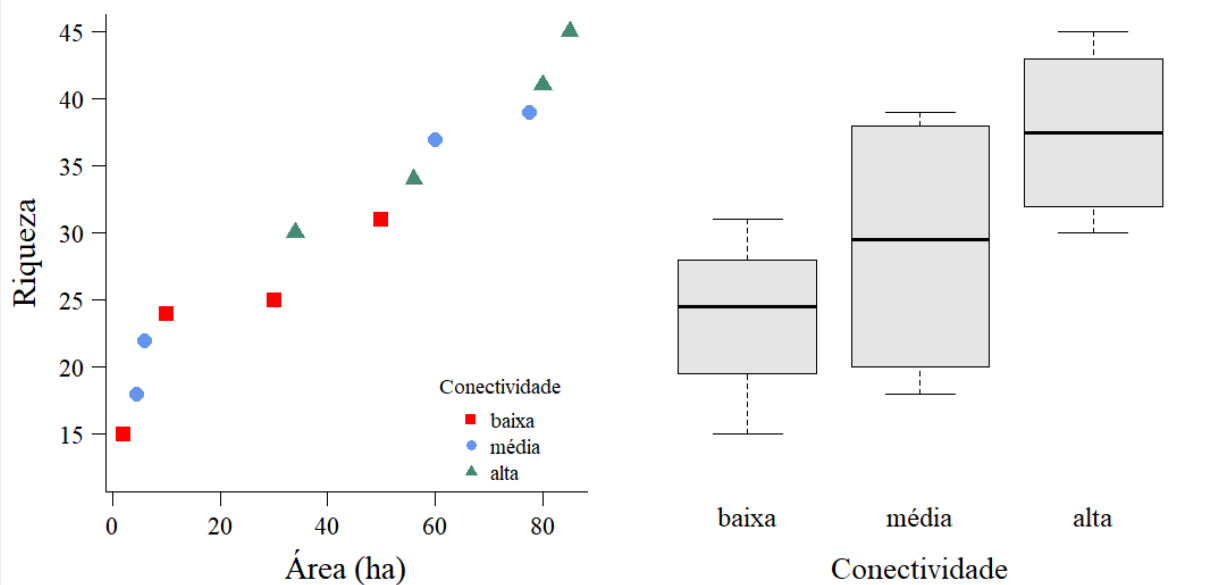
No código do diagrama de caixas acima, para ter mais controle sobre título e legendas dos eixos, solicitamos na função que estes não fossem grafados. Agora podemos incluir esses elementos utilizando a função `mtext` que inclui caracteres nas bordas do gráfico:

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

```

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

```



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

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

Dispositivos de Arquivos

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

```

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

```

```

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

```

Gerenciando dispositivos

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

Próximos passos

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

- 1) caso esteja usando um Mac, pode ter problemas para abrir a janela, nesse caso use `quartz()`
- 2) a ordem dos lados de um gráfico sempre obedece o sentido horário partindo do eixo x
- 3) entre o título, legenda e linha do eixo

From:
<http://ecor.ib.usp.br/> - **ecoR**

Permanent link:
http://ecor.ib.usp.br/doku.php?id=02_tutoriais:tutorial5:start&rev=1692731603

Last update: **2023/08/22 16:13**