

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/336742836>

Workshop Técnicas de Tratamento de Dados em R (RStudio)

Book · September 2019

CITATIONS

0

READS

9

2 authors:



[Pedro M. Nogueira](#)
Universidade de Évora

61 PUBLICATIONS 76 CITATIONS

[SEE PROFILE](#)



[Miguel Maia](#)
Universidade de Évora

26 PUBLICATIONS 2 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Arqueometria [View project](#)



Barragens Republica Dominicana [View project](#)



XII

CONGRESSO

CONGRESO

IBÉRICO DE GEOQUÍMICA

Workshop

Técnicas de Tratamento de Dados em R (RStudio)

22 Setembro de 2019
Septiembre

Évora, Portugal

CIG 2019

XII Congresso Ibérico de Geoquímica | XX Semana da Geoquímica

22 setembro de 2019, Évora, Portugal



Workshop

Técnicas de Tratamento de Dados em R (RStudio)

Editores: Pedro Nogueira & Miguel Maia

CONTEÚDO

1- Introdução e objetivos	3
2- Instalação de software e de módulos	4
3- Linha de comandos, Variáveis, operações simples, lógica e funções pré-definidas.....	8
Utilizando a consola como uma calculadora científica!	9
4- Tipos de dados	13
Variáveis: tipos e nomes	13
Matrizes, vetores, arrays e listas.....	13
Data Frames	15
Fatores.....	16
5- Preparação de dados	18
6- Importação e exportação de dados numéricos de fontes externas (Excel e csv).....	21
A pasta de trabalho	21
Leitura de ficheiros csv.....	22
7- Organização de dados	25
8- Tratamento de dados.....	29
Funções de tendência central	29
Função <i>summary</i>	29
Agregar colunas ou linhas	31
Cálculo agregado	31
9- gráficos: dispersão, linhas barras e histogramas	34
Gráficos de dispersão e de linhas.....	34
Gráficos de funções.....	36
Gráficos de barras	37
Histogramas.....	39
10- Etiquetas, grelhas, legendas, linhas e textos	43
Eixos, nomes dos eixos e título	43
Grelhas	44
Legendas.....	45
Linhas	46
Textos em gráficos	48
11- Composição de múltiplos gráficos e exportação dos gráficos	50
Gráficos múltiplos	50
Exportar gráficos	52
12- Gráficos avançados: o ggplot	54
Geometrias.....	55

Agrupamento de gráficos em ggplot.....	57
Diagramas normalizados.....	59
Diagramas triangulares.....	61
13- Criação de funções em R.....	81
14- Modelação simples: Correlações.....	83
Correlação e variância.....	83
Modelação linear.....	84
Diagramas de correlação.....	86
15- Análise de agrupamentos- <i>clusters</i> (hclust).....	88
16- Análise de componentes principais (PCA).....	94
ANEXO 1.....	100
Cábulas de funções.....	100
Operadores aritméticos.....	100
Operadores lógicos.....	100
Funções numéricas e constantes.....	101
Funções estatísticas.....	102
Outras funções.....	102
Cábulas de argumentos para gráficos.....	103
Símbolos.....	103
Tipos de linhas.....	103
Cores.....	104

1- INTRODUÇÃO E OBJETIVOS

Este curso está desenhado para que os participantes possam fazer um percurso de aprendizagem que vai desde a instalação do software passando pelos primeiros passos na utilização da linha de comandos, a compreensão da lógica de funcionamento dos programas de análise e tratamento de dados (R/RStudio) e, por fim, centrando-se na aplicação de métodos matemáticos e estatísticos a dados geológicos.

Os não especialistas em informática tem muitas vezes aversão a programas de computador que funcionam com linha de comandos, mas depois de algum tempo de treino, compreensão e habituação à lógica de funcionamento esta é a forma mais rápida e eficiente de se obter resultados.

O primeiro módulo servirá de introdução e de habituação ao software, assim como para que possa entender os diversos tipos de dados que o software suporta. O segundo módulo trata de descrever como se deve organizar e preparar os dados para um projeto, como estes podem ser importados para o software e formas simples de manipulação dos dados. O terceiro módulo trata de apresentar formas simples de representação de dados sob a forma de gráficos, incluindo a parametrização de legendas, eixos e símbolos. O quarto módulo trata da realização de alguns cálculos estatísticos, desde a estatística descritiva simples até algumas funções de análise bivariada e multivariada. O quinto módulo apresenta um *package* específico para tratamento de dados geoquímicos ligados a processos ígneos, o GCDKIT. No sexto e último módulo cada participante deverá desenvolver um pequeno projeto de análise e tratamento de dados, desde a sua preparação, passando pelo tratamento de dados até à interpretação, modelação e apresentação dos resultados.

O curso está desenhado para ser o menos expositivo possível, procurando que cada aluno possa, ao seu ritmo, ir entendendo os assuntos em discussão e aprendendo através da resolução dos exercícios propostos. Nunca hesite em conversar com o formador sobre qualquer assunto das matérias lecionadas.

2- INSTALAÇÃO DE SOFTWARE E DE MÓDULOS

O curso vai abordar o software que serve de base para a linguagem de programação R. Esta linguagem tem uma interface gráfica própria, porém neste curso vamos recorrer a um interface gráfico avançado (IDE-Integrated Development Environment) que se chama RStudio.

A linguagem R é um projeto de software livre, o que significa que o seu código é aberto e que qualquer um pode contribuir quer sinalizando e corrigindo problemas detetados quer criando módulos de aplicações (*packages*) específicos. O facto de ser livre significa também que pode ser descarregado e utilizado de forma gratuita.

A instalação dos programas necessários para este curso começa pela visita à página <http://www.r-project.org> e na secção download escolher qual o seu sistema operativo (Mac, Linux ou Windows) e qual o seu processador 32 ou 64-bits. Caso tenha dúvidas pergunte ou veja nas definições do seu sistema (Figura 2.1).

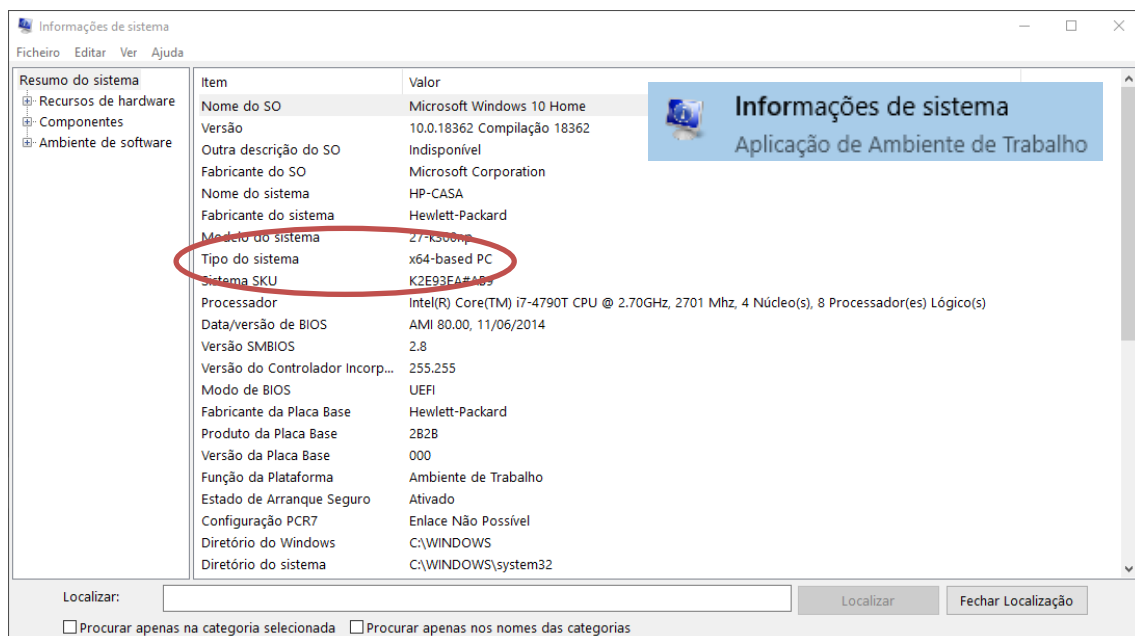


Figura 2.1. Informações do sistema.

As informações do sistema podem ser visualizadas com o programa [Informação do sistema], no resumo onde aparece o Tipo de sistema, neste caso x64 significa que se trata de um sistema de 64 bits.

Na página r-projet.org no lado esquerdo debaixo do texto download aparece a opção CRAN e na nova página deve escolher um dos servidores internacionais que possuem cópias das últimas versões do programa.

Depois de descarregar o software instale-o utilizando as opções por omissão.

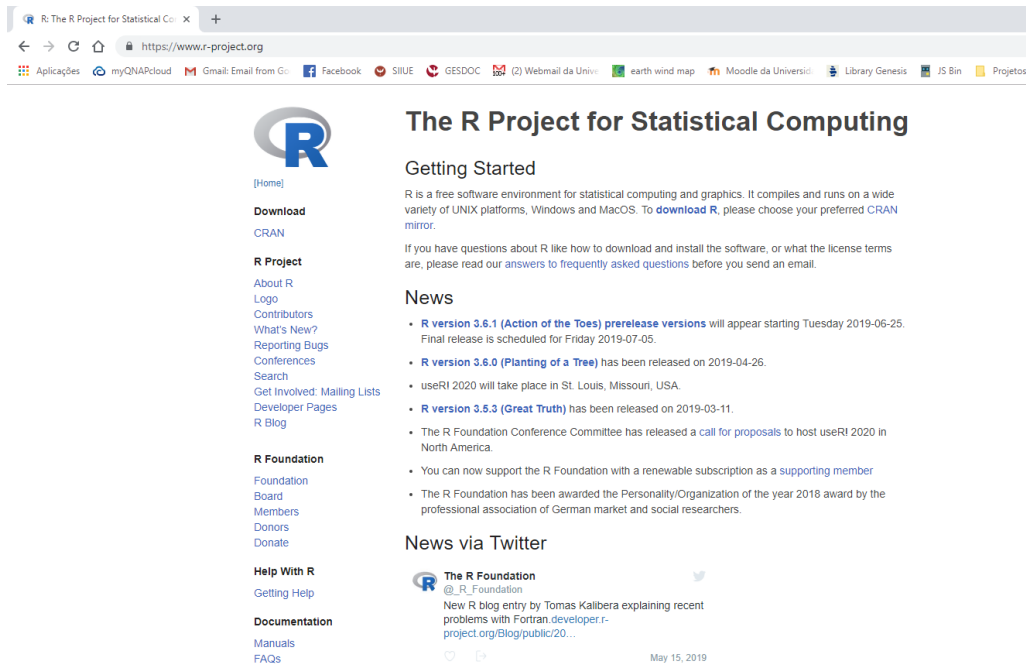


Figura 2.2. A página inicial do software R.

De seguida deve visitar a página do software RStudio que corresponde à interface gráfica avançada para trabalhar em R, e que facilita o processo de trabalho. A página web é <http://www.rstudio.com>. Na página de descarga escolha a opção RStudio Desktop que é gratuita e permite aceder a todas as opções necessárias para este curso.

Na página do RStudio pode ainda encontrar alguns módulos (*packages*) que pode descarregar nas respetivas páginas. Existe, porém, uma forma direta dentro do R (ou RStudio) de descarregar e instalar estes mesmos plugins.

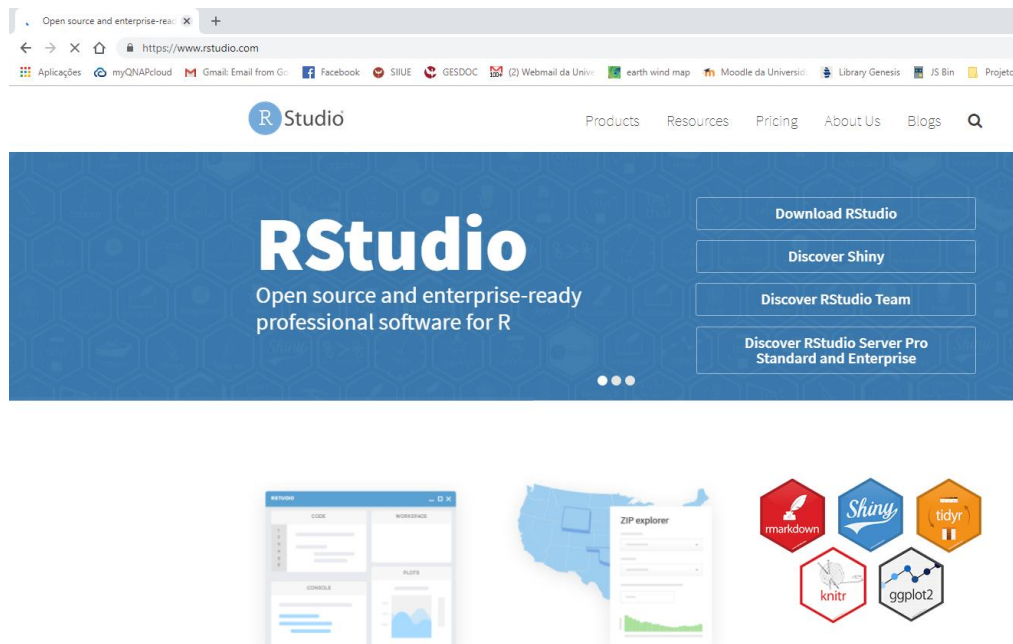


Figura 2.3. A página inicial do RStudio.

Depois de instalar o software com as opções padrão poderá iniciar a sua viagem pela análise e tratamento de dados aplicado à geologia.

Iniciando o software RStudio ele automaticamente vai executar em segundo plano a linguagem de programação R. A janela inicial quando arranca o RStudio deve ser semelhante à apresentada na Figura 2.4.

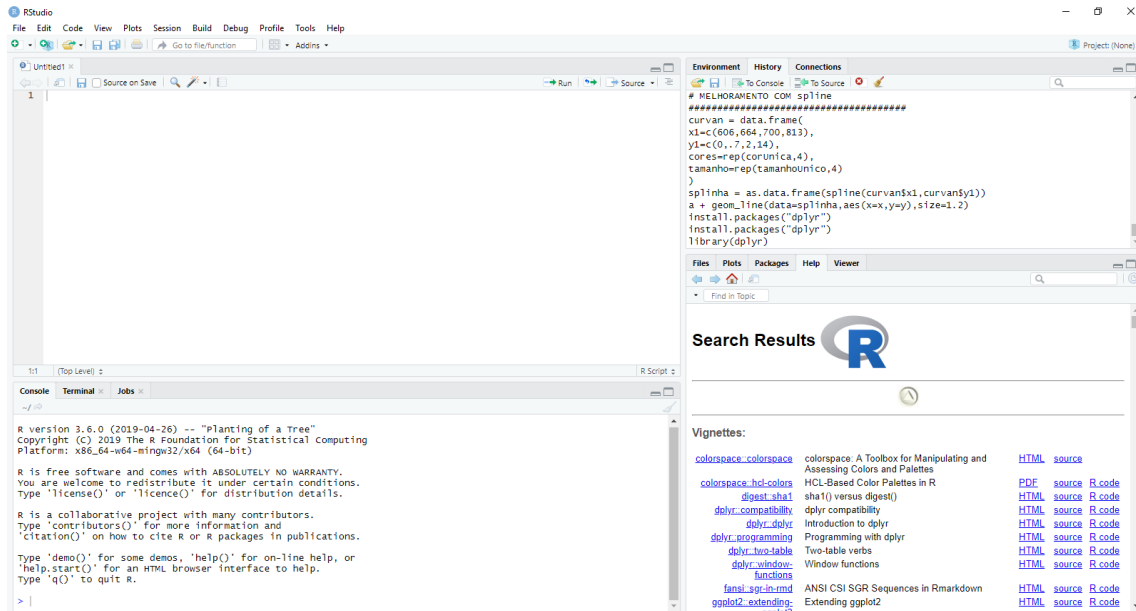


Figura 2.4. A interface do RStudio.

Além das funções e comandos que o R tem definidos existe um grande número de ferramentas extra que permitem fazer tarefas específicas de acordo com os interesses e necessidades de cada utilizador, os módulos ou *packages* em inglês.

Estas ferramentas podem ser instaladas através de uma instrução na linha de comandos:

```
install.packages("tidyverse")
```

Ou através do menu [Tools>Install Packages...] que abre uma subjanela onde é possível escolher qual o módulo que se pretende instalar (Figura 2.5).

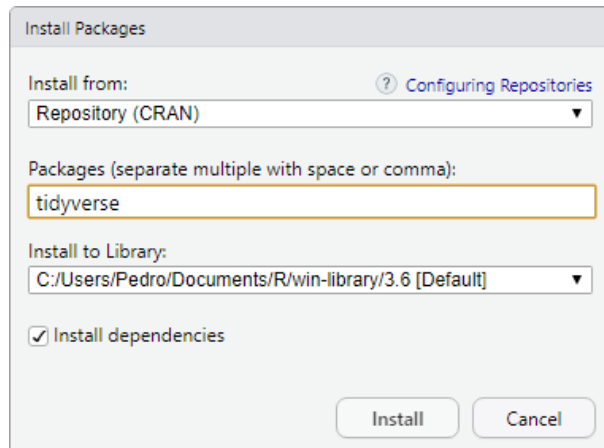


Figura 2.5. A subjanela de instalar módulos/packages.

Para estes primeiros módulos do nosso curso sugerimos a instalação dos *packages* [tidyr], [ggplot2] e [Ternary].

Exercício 2.1

Instale o [tidyr] através do menu do RStudio.

Instale o [ggplot2] através da linha de comandos.

Instale o [Ternary] através da linha de comandos.

Anote e comente os avisos que estas instalações geraram.

3- LINHA DE COMANDOS, VARIÁVEIS, OPERAÇÕES SIMPLES, LÓGICA E FUNÇÕES PRÉ-DEFINIDAS

O R é essencialmente uma linguagem de programação que funciona por execução de instruções numa linha de comandos, mas não se assustem que esta forma vai ser mais útil e fácil do que se pode pensar ao início. Assim abandonem o rato por uns momentos e vamos usar o teclado...

No RStudio o equivalente da linha de comandos é designado por consola (Figura 3.1), e é aqui que vamos escrever as instruções para o interpretador R. Depois de cada instrução a consola deverá apresentar o resultado da instrução. Se o resultado for uma nova variável ou um gráfico estes vão aparecer no separador das respetivas subjanelas (Variáveis-*Environment* e Gráficos-*Plot*).

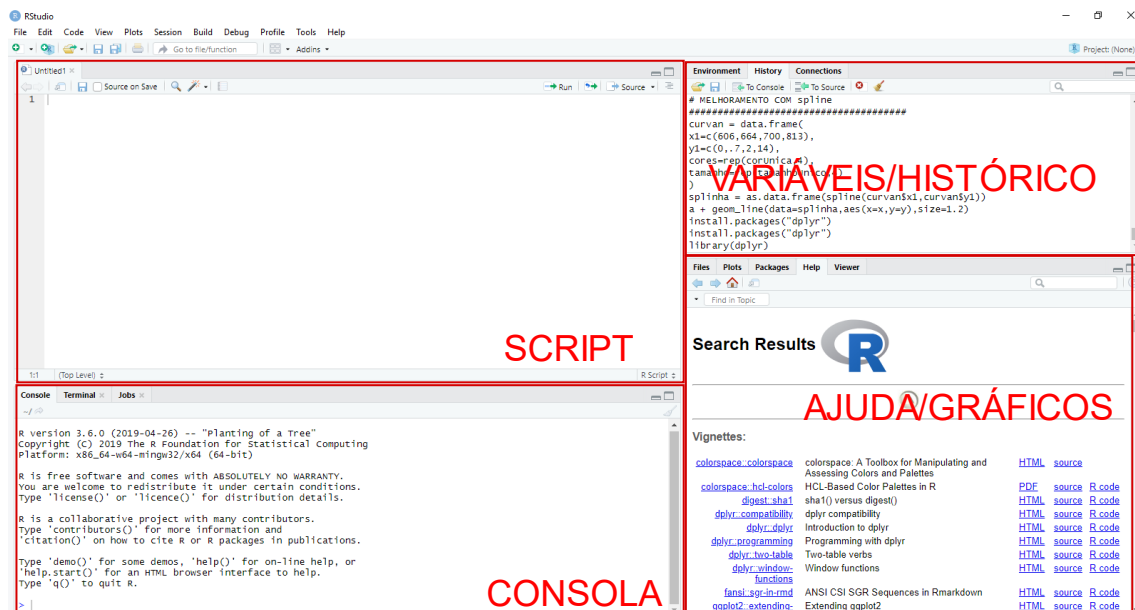


Figura 3.1. As diferentes subjanelas do RStudio.

Para este curso encorajamos que escreva os comandos sempre que sejam suficientemente curtos, isso permite-lhe tomar contacto com as diferentes partes de cada instrução e questionar-se sobre o seu significado. Caso haja comandos mais longos ou seqüências de comandos pode criar um *script* (Figura 3.2.) onde, através do copiar e colar, pode analisar e alterar cada novo comando e, adicionalmente, entender o significado de cada parâmetro antes do executar na linha de comandos.

Nestes primeiros exercícios basta escrever os comandos sugeridos na consola e, de seguida, observar e interpretar os resultados. Na subjanela de variáveis mantenha o separador *Environment* visível para poder ir observando as variáveis criadas, o seu tipo e conteúdo.

Pode recorrer a ajuda sobre o funcionamento de qualquer comando escrevendo na consola:

```
help (comando)
```

ou em alternativa:

```
? (comando)
```

Uma ajuda aparece na subjanela respectiva.

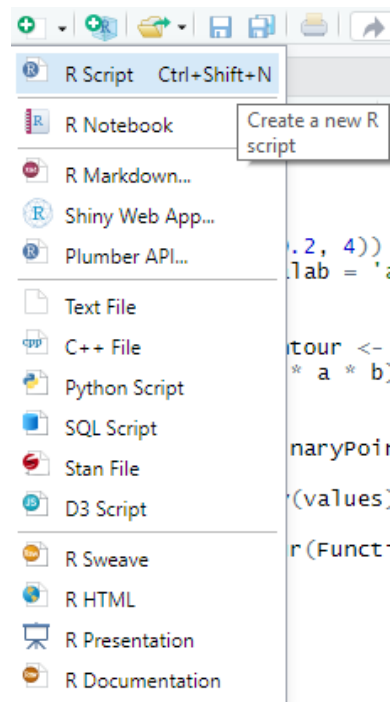


Figura 3.2. Menu de criação de novo *script*.

Utilizando a consola como uma calculadora científica!

O R pode ser utilizado como uma calculadora científica bastante avançada. Para isso basta escrever as expressões que pretende calcular; por exemplo, para somar dois valores basta escrever a respetiva expressão e carregar [RETURN].

Podem ser criadas variáveis em que o nome da variável é uma sequência de caracteres começada por uma letra. Para atribuir um valor à variável utiliza-se o sinal de `<-` que significa atribuir um valor a uma variável, pode, em alternativa, ser utilizado o sinal de igual (`=`). Por questões convencionais em R utiliza-se geralmente o primeiro. Nesta sebenta, muitas vezes é o sinal de igual que aparece escrito, mas ambos podem ser utilizados.

O R como uma consola de cálculo:

```
> 1 + 1
[1] 2
> x <- 2
> x
[1] 2
> y = 3 * 5
> y
[1] 15
```

O sinal de maior (`>`) aqui apresentado significa que o computador está à espera dos seus comandos. No exemplo acima o `[1]` corresponde à primeira resposta à sua instrução.

Analisemos então as respostas da consola de R.

O primeiro exemplo é um cálculo direto.

O segundo exemplo cria uma variável designada **x** e atribui-lhe o valor de 2. a segunda linha questiona qual o valor de **x**.

O terceiro exemplo cria uma variável **y** e atribui-lhe o valor da operação 3 x 5.

Note a utilização indiferenciada de **<-** e **=**.

O R não se preocupa com espaços entre operadores e utiliza a ordem convencional de resolução de prioridades no cálculo. Vejamos alguns exemplos mais:

```
> 4/3
[1] 1.333333
> 10^2
[1] 100
> 10e2
[1] 1000
```

Os operadores na consola R são iguais aos utilizados em muitos outros programas além das operações elementares como + - * / () existe um conjunto de operadores matemáticos e lógicos que estão listados numa cábula no anexo I.

A primeira função R, que vamos usar muito extensivamente neste curso, é a função **c** que corresponde a combinar valores num vetor ou lista. Assim para criar um vetor com os valores 3, 5 e 7 utilize o comando:

```
c(3, 5, 7)
```

Um operador que também usado frequentemente é o símbolo de dois pontos (**:**) que permite referir uma sequência de valores. Assim, **1:2** corresponde à sequência de valores 1, 2; **3:5** corresponde aos valores 3, 4 e 5. Pode saber mais se efetuar o comando:

```
?(":")
```

Exercício 3.1.

Experimente correr os seguintes comandos:

```
# Vamos testar
x <- 1:10
x
c(3, 5, 7)
```

Exercício 3.2.

Explique os resultados obtidos!!!

O que significa o comando # ?

Além dos operadores existe um conjunto de funções que realizam diversos cálculos matemáticos e estatísticos básicos. Exemplos disso são funções como $\log(x)$, \sqrt{x} , $\sin(x)$, π , etc. Estes operadores são razoavelmente autoexplicativos e uma seleção deles está também referida nas tabelas do anexo I.

Vejamos alguns exemplos e seus resultados:

```
> log(1)
[1] 0
> cos(pi)
[1] -1
> log(-1)
[1] NaN
Warning message:
In log(-1) : NaNs produced
> Inf+Inf
[1] Inf
> Inf/0
[1] [Inf]
> 1/Inf
[1] 0
```

Neste exemplo além de usarmos algumas funções apresentamos-lhe algumas das notificações que o R utiliza. **NaN** significa *Not a Number* e é utilizado quando o resultado é impossível. No exemplo acima é também produzido um aviso (*Warning*) referindo que o resultado produz respostas impossíveis. Outros símbolos utilizados são **Inf** para valores infinitos e **NA** para valores inexistentes (*Not Available*).

Algumas operações envolvem resultados que não são numéricos, mas lógicos, para isso o R tem a designação TRUE (T) e FALSE (F) para referir o resultado de uma operação lógica.

Vejamos então alguns exemplos simples.

```
> 3 == 2
[1] FALSE
> 3 > 2
[1] TRUE
> cos(pi)
[1] -1
> sin(pi) <= cos(pi)
[1] FALSE
> sin(pi) == 0
[1] FALSE
```


UPS, desta última não estava à espera, então o seno de pi não é igual a zero?

Exercício 3.3.

Explique este resultado.

Exercício 3.4.

Experimente correr os seguintes comandos:

```
# Vamos testar
x <- c(1:10)
> x > 8
> x < 5
> x > 8 | x < 5
> x[c(T, T, T, T, F, F, F, F, T, T)]
```

Exercício 3.5.

Explique os resultados obtidos!!!

4- TIPOS DE DADOS

Variáveis: tipos e nomes

O R pode ter variáveis simples designadas por nomes que devem começar por letras e conter depois da primeira letra, outras letras, números ou o carácter de *underscore* (`_`). Os nomes das variáveis são sensíveis à grafia (ex. a variável `ab` é diferente da `Ab`).

São exemplos de nomes de variáveis:

```
a, x, y1, a_1
```

Uma variável simples contém um valor único que pode ser de diferentes tipos, seja isso números inteiros, decimais, caracteres ou valores lógicos. Os exemplos abaixo apresentam exemplos de criação de variáveis simples.

```
a = 3
y1 <- 2.235
nome = "Manuel"
teste = TRUE
```

Matrizes, vetores, arrays e listas

Os vetores são conjuntos de dados do mesmo tipo e que agregam informação que pode ser referida a partir de um índice.

```
amostras = c("Am1", "Am2", "Am3", "Am4")
Si = c(65.2, 73.2, 66.3, 56.7)
Lamina = c(TRUE, FALSE, FALSE, TRUE)
```

Para se referir um elemento de um vetor deve-se recorrer ao seu índice.

```
> amostras[2]
[1] "Am2"
> Si[4]
[1] 56.7
> Si[1:2]
[1] 65.2 73.2
```

As matrizes correspondem a uma coleção de elementos do mesmo tipo definida numa forma retangular, através de linhas e colunas. As matrizes podem ser criadas a partir do comando `matrix`. Este comando tem diversos argumentos que permitem configurar a forma como a matriz é criada. Desses valores devemos salientar os três primeiros argumentos que correspondem aos dados que constituem a matriz, o número de linhas-`nrow` e número de colunas-`ncol`.

```
# gerar uma matriz numérica com 3 linhas e 2 colunas (3 x 2)
y = matrix(1:6, nrow=3, ncol=2)

# Um outro exemplo de criação de uma matriz 2 x 2
elementos <- c(1, 26, 24, 68)
nomesLinhas <- c("R1", "R2")
nomesColunas <- c("C1", "C2")
matrizA <- matrix(elementos, nrow=2, ncol=2, byrow=TRUE,
  dimnames=list(nomesLinhas, nomesColunas))
```

Tal como nos vetores os elementos constituintes das matrizes são referidos pelos seus índices. Na sua criação além destes argumentos no exemplo acima pode-se observar que na criação de uma matriz deve ser indicada a forma como a matriz é preenchida (`byrow`, isto é, por linha quando `byrow = TRUE` ou por coluna, quando `byrow = FALSE`) e os nomes das dimensões - `dimnames` (linhas e colunas).

Vejamos alguns exemplos.

```
> matrizA
  C1 C2
R1  1 26
R2 24 68
> matrizA[1]
[1] 1
> matrizA[2]
[1] 24
> matrizA[1,2]
[1] 26
```

No caso da matriz acima ela pode ser referida quer na sua totalidade quer elemento a elemento pela ordem coluna, linha, quer pelos seus índices de linha e coluna. Podem ainda ser referidos os elementos (linhas ou colunas) pelo nome da linha ou coluna, tal como se entende do exemplo abaixo.

```
> matrizA[, "C2"]
R1 R2
26 68
> matrizA["R1", ]
C1 C2
1 26
```

As arrays apresentam as mesmas características que as matrizes, mas apresentam a possibilidade de terem mais de duas dimensões.

As listas são conjuntos de dados que podem ser de qualquer um dos outros tipos. A referência a elementos de uma lista pode ser pelo seu índice quer por qualquer um dos seus elementos.

```
# Uma lista de três componentes
# um texto, um vetor, e um escalar
telefonos1=c("12345678", "87654321")
pessoal = list(nome="João", numerosT=telefonos1, idade=36)
telefonos2=c("12121212","87878787")
pessoa2 = list(nome="Manuel", numerosT=telefonos2, idade=45)

# Uma lista de listas
listaPessoas <- list(pessoal, pessoa2)
```

Vejamos agora a aplicação destas definições.

```
> pessoal
$nome
[1] "João"
$numerosT
[1] "12345678" "87654321"
$idade
[1] 36
```

Data Frames

Uma data frame corresponde a um conjunto de vetores de igual tamanho. Esses vetores não têm de ser necessariamente do mesmo tipo de dados. As data frames são utilizadas para armazenar tabelas de dados. Vejamos um exemplo de criação de uma data frame - [granitos].

```
> amostras = c("Am1", "Am2","Am3","Am4")
> Si = c(65.2, 73.2, 66.3, 56.7)
> Lamina = c(TRUE, FALSE, FALSE, TRUE)
> granitos = data.frame(amostras,Si,Lamina)
```

As data frames têm uma grande flexibilidade e permitem o tratamento dos dados de diversas formas. O exemplo abaixo mostra a matriz granitos que contém três variáveis (amostras, Si e Lamina). Cada variável tem 4 observações.

```
> granitos
  amostras  Si Lamina
1     Am1 65.2   TRUE
```

```
2      Am2 73.2 FALSE
3      Am3 66.3 FALSE
4      Am4 56.7  TRUE
```

Numa *data frame* as variáveis (colunas) podem ser referidas com recurso ao símbolo \$, como se entende do exemplo abaixo que mostra a lista dos valores de Si na *data frame* [granitos].

```
> granitos$Si
[1] 65.2 73.2 66.3 56.7
```

Quando se pretende aceder a uma determinada variável ela pode ser referida pelo número da coluna ou pelo nome da coluna, veja os exemplos abaixo.

```
> granitos[,3]
[1] TRUE FALSE FALSE TRUE

> granitos["Lamina"]
Lamina
1 TRUE
2 FALSE
3 FALSE
4 TRUE

> granitos[, "Lamina"]
[1] TRUE FALSE FALSE TRUE
```

Fatores

Deve ser ainda tido em conta que em alguns casos as variáveis, quando são criadas, os seus elementos são convertidos para fatores. Quando uma variável é nominal o R cria uma lista de inteiros que corresponde a cada elemento e uma lista de caracteres com os elementos nominais, mapeada num formato de um fator para muitos elementos. Existem funções que permitem a conversão de variáveis fatorizadas nos seus equivalentes numéricos.

Atente nos exemplos apresentados abaixo.

```
> sílica = as.factor(granitos$Si)
> sílica
[1] 65.2 73.2 66.3 56.7
Levels: 56.7 65.2 66.3 73.2
> sílica[1]
[1] 65.2
Levels: 56.7 65.2 66.3 73.2
> as.numeric(sílica)
[1] 2 4 3 1
```

Exercício 4.1.

Explique os resultados de cada um dos comandos executados.

5- PREPARAÇÃO DE DADOS

Os dados que obtemos no mundo real muitas vezes não vem na forma de poder ser tratados pelo software R. Assim é necessário proceder-se à sua transformação e adaptação para poderem ser lidos pelos diversos módulos de importação de dados.

A maioria dos dados para aplicações em geociências deve ter um formato de acordo com o modelo variáveis-observações, o que se adequa perfeitamente ao seu armazenamento em *data frames*. Convém, porém, ter em atenção que muitas das vezes os ficheiros onde estão os dados não estão formatados desta forma. Um exemplo muito comum é aquele que vem da utilização de folhas de cálculo como o *Microsoft Excel* ou o *Google Sheets*.

Muitas vezes esses dados incluem informações fora deste formato o que torna a importação dos dados impossível. Vejamos o exemplo da folha abaixo proveniente de um laboratório de análises químicas:

ACTLABS

TOTAL ORGANIC CARBON, PROGRAMMED PYROLYSIS DATA

ACTIVATION LABORATORIES LTD.

A11-11001

Client ID	Well Name	Sample Type	Sample Prep	Leeco TOC	RE			Tmax (°C)	Ro, %	HI	OI	S2/S3	S1/TOC *100	PI	Notes		Lab ID	
					S1	S2	S3								Checks	Pyrogram		
AQ2	A11-11001	Powder Rock	NOPR	0,25	0,01	0,33	0,52	475	**		134	211	0,6	4	0,03	TOC	f	3,402E+09
AQ4	A11-11001	Powder Rock	NOPR	0,10	0,01	0,21	0,22	547	**		219	229	1,0	11	0,05	TOC RE	f	3,402E+09
GA01B	A11-11001	Powder Rock	NOPR	0,84	0,01	0,42	0,51	554	**		50	61	0,8	1	0,02	TOC	f	3,402E+09
GA37B	A11-11001	Powder Rock	NOPR	0,46	0,01	0,22	0,20	498	**		48	44	1,1	2	0,04	TOC	f	3,402E+09
GA40	A11-11001	Powder Rock	NOPR	0,45	0,00	0,36	0,13	503	**		80	29	2,8			TOC	f	3,402E+09
GA49	A11-11001	Powder Rock	NOPR	0,57	0,03	0,49	0,07	492	**		86	12	7,0	5	0,06	TOC RE	f	3,402E+09
ACRW423	A11-11001	Powder Rock	NOPR	0,34	0,04	0,35	0,16	520	**		104	47	2,2	12	0,11	TOC	f	3,402E+09
ACRH217	A11-11001	Powder Rock	NOPR	0,48	0,02	0,37	0,10	505	**		77	21	3,7	4	0,05	TOC	f	3,402E+09
ACMV430	A11-11001	Powder Rock	NOPR	0,24	0,01	0,31	0,32	506	**		130	134	1,0	4	0,03	TOC RE	f	3,402E+09
ACRH034	A11-11001	Powder Rock	NOPR	0,71	0,01	0,31	0,35	532	**		44	49	0,9	1	0,03	TOC	f	3,402E+09

Notes:
 -f - not measured or invalid value for Tmax
 TOC - Total Organic Carbon, wt %
 S1 - volatile hydrocarbon (HC) content, mg HC/ g rock
 S2 - remaining HC generative potential, mg HC/ g rock
 S3 - carbon dioxide content, mg CO₂ / g rock

Pyrogram:
 f - flat S2 peak
 n - normal
 hs2sh - low temperature S2 shoulder
 hs2sh - high temperature S2 shoulder
 hs2p - low temperature S2 peak
 hs2p - high temperature S2 peak

* - comments regarding contamination
 ** - low S2. Tmax is unreliable
 Meas. %Ro - measured vitrinite reflectance
 HI - Hydrogen index = S2 x 100 / TOC, mg HC/ g TOC
 OI - Oxygen Index = S3 x 100 / TOC, mg CO₂ / g TOC
 PI - Production Index = S1 / (S1+S2)

LECO - TOC on Leco Instrument
 RE - Programmed pyrolysis or
 TOC on Rock-Eval instrument
 SRA - Programmed pyrolysis by SRA
 Instrument
 EXT - Extracted Rock
 NOPR - Normal Preparation

Figura 5.1. Exemplo de uma folha de cálculo com resultados analíticos.

Como se pode observar as linhas 1 a 12 correspondem informação complementar sobre o nome do laboratório, o tipo de análises e o pacote de amostras. Estas linhas não fazem parte da informação relevante para o tratamento de amostras e assim poderão ser eliminadas.

As linhas 26 e seguintes tem uma imagem que corresponde à legenda e corresponde a metadados. Deve também ser eliminada. As linhas 12 e 13 contêm informação do nome das variáveis, mas em alguns casos essas variáveis estão agrupadas. Por exemplo as colunas N, O e P contêm os valores de RE-S1, RE-S2 e RE-S3. Devem, pois, as duas linhas ser convertidas numa única linha com os nomes das respetivas variáveis.

As linhas 14 e 15 estão em branco e escondidas, deve ser evitado ter linhas de observações em branco. As variáveis que não tem valores também devem ser eliminadas.

Repare ainda que existem colunas escondidas (A, D, E, F, G, H e I), elimine-as.

Finalmente, para os nomes das variáveis, escolha designações que não contenham caracteres especiais e sinais (ex. /, °, %) pois isto pode interferir com os nomes das variáveis em R.

Depois de limpa e pronta para importar a tabela de dados fica com o aspeto da Figura 5.2.

Client ID	Well Name	Sample Type	Sample Prep	LecoTOC	RE-S1	RE-S2	RE-S3	Tmax(C)	HI	OI	S2-S3	SI TOC	PI	Checks	Pyrogram	Lab ID
AQ2	A11-11001	Powder Rock	NOPR	0,25	0,01	0,33	0,52	475	134	211	0,6	4	0,03	TOC	r	3402168998
AQ4	A11-11001	Powder Rock	NOPR	0,10	0,01	0,21	0,22	547	219	229	1,0	11	0,05	TOC RE	r	3402169000
GA01B	A11-11001	Powder Rock	NOPR	0,84	0,01	0,42	0,51	554	50	61	0,8	1	0,02	TOC	r	3402169002
GA37B	A11-11001	Powder Rock	NOPR	0,46	0,01	0,22	0,20	498	48	44	1,1	2	0,04	TOC	r	3402169004
GA40	A11-11001	Powder Rock	NOPR	0,45	0,00	0,36	0,13	503	80	29	2,8			TOC	r	3402169006
GA49	A11-11001	Powder Rock	NOPR	0,57	0,03	0,49	0,07	492	86	12	7,0	5	0,06	TOC RE	r	3402169008
ACRW423	A11-11001	Powder Rock	NOPR	0,34	0,04	0,35	0,16	520	104	47	2,2	12	0,11	TOC	r	3402169010
ACRH217	A11-11001	Powder Rock	NOPR	0,48	0,02	0,37	0,10	505	77	21	3,7	4	0,05	TOC	r	3402169012
ACMV430	A11-11001	Powder Rock	NOPR	0,24	0,01	0,31	0,32	506	130	134	1,0	4	0,03	TOC RE	r	3402169014
ACRH034	A11-11001	Powder Rock	NOPR	0,71	0,01	0,31	0,35	532	44	49	0,9	1	0,03	TOC	r	3402169016

Figura 5.2. Exemplo de folha de cálculo depois de preparada para ser importada para o R.

A preparação de dados é uma operação que deve ser avaliada para cada caso e deve ser efetuada com o máximo de cuidado para evitar perda de informação.

O R possui alguns módulos de importação de dados que permitem importar dados diretamente de folhas de cálculo e de outros formatos comerciais, mas é sempre preferível que tenha os dados num formato de texto simples, cujo exemplo máximo é o formato CSV-*Comma Separated Values*, ou seja valores separados por vírgulas.

Para algumas operações os dados podem não estar organizados no formato variável-observação, formato alargado (*wide*, em inglês) descrito anteriormente. Um exemplo disso são algumas funções gráficas do módulo [ggplot] em que a organização dos dados deve ser no formato chave-valor, isto é, num formato alongado (*long*, em inglês).

Um exemplo do formato alargado é por exemplo a tabela das temperaturas por cada hora do dia, como o exemplo seguinte:

Tabela 5.1. A temperatura em Maputo em formato alargado

Dia	12h00	13h00	14h00	15h00	16h00	17h00	18h00
16 Jun	24	24	25	24	23	21	20
17 Jun	24	24	24	24	24	22	20
18 Jun	24	24	24	24	23	21	20

A mesma informação em formato alongado terá o seguinte aspeto:

Tabela 5.2. A temperatura em Maputo em formato alongado:

Dia	Hora	Valor
16 Jun	12h00	24
16 Jun	13h00	24
...		
17 Jun	12h00	24
...		
17 Jun	18h00	20
...		

Estas transformações podem ser efetuadas numa qualquer folha de cálculo, mas o R através do módulo [tidyr] tem funções que permitem a transformação de uma tabela na outra. Esta transformação será apresentada mais à frente neste curso.

Exercício 5.1:

Na pasta Exercicios que acompanha este curso encontra um ficheiro [Exercicio5-1.xls] com um conjunto de dados. Com auxílio de um programa de folha de cálculo faça a limpeza de dados e crie um ficheiro no formato [csv] com a informação relevante.

Discuta as linhas e colunas com os seus colegas e explique as suas opções.

6- IMPORTAÇÃO E EXPORTAÇÃO DE DADOS NUMÉRICOS DE FONTES EXTERNAS (EXCEL E CSV)

A pasta de trabalho

Antes de tratar propriamente de ler/importar ficheiros para o R vamos trazer aqui a noção de pasta/diretório de trabalho. O R necessita que se indique qual a pasta que é definida como de trabalho (em inglês, *working directory*). A definição da pasta de trabalho é fundamental, pois é nela, ou nas suas subpastas que vão ser lidos e gravados todos os dados.

Assim é aconselhável que para cada projeto R se crie uma pasta de trabalho, com diversas subpastas organizadas por assunto. O conteúdo da pasta de trabalho pode ser visto no separador [Files] da subjanela [Ajuda/Gráficos/...]. Pode ainda saber qual a pasta que está definida para trabalho com o comando:

```
> getwd()
[1] "C:/Users/Pedro/Documents"
```

No exemplo acima o RStudio tem como pasta inicial de trabalho a pasta "C:/Users/Pedro/Documents". Para se alterar a pasta de trabalho utilize o comando `setwd`.

```
setwd("caminho")
```

Em que caminho é o nome da pasta e disco que se pretende utilizar para trabalho. Veja o exemplo seguinte:

```
setwd("D:/curso R")
```

No RStudio pode recorrer também ao menu [Session>Set Working Directory>Choose Directory...] para definir qual a pasta de trabalho (Figura 6.1).

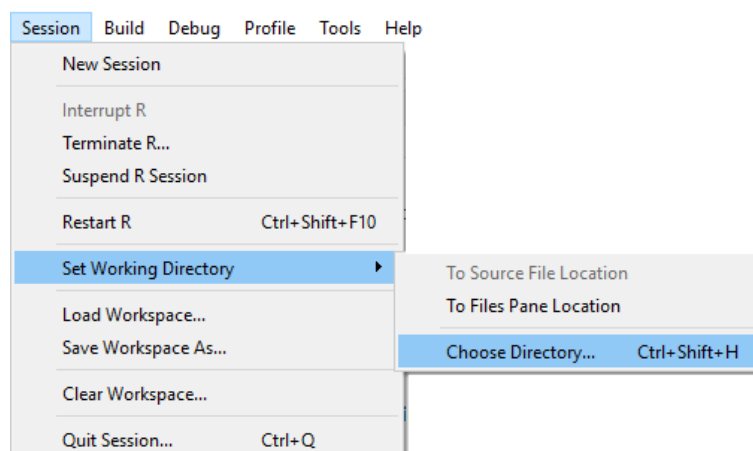


Figura 6.1. A seleção da pasta de trabalho através dos menus do RStudio.

Leitura de ficheiros csv

Depois de ter os dados num formato apropriado e limpos de informação desnecessária deve gravá-los em formato csv ou em formato excel. Estes formatos são devidamente adequados para serem importados e tratados pelo software R.

A função mais elementar para ler ficheiros em R é a função `read.table()` ou nas suas versão para csv:

```
read.csv()
read.csv2()
```

Esta função lê um ficheiro em formato csv e permite definir diversos parâmetros que se prendem com os dados. Devemos lembrar que dependendo dos países e do sistema operativo os ficheiros csv não são todos iguais, por exemplo os valores podem ser separados por uma vírgula (,) ou um ponto e vírgula (;).

Apesar de ser o mais universal, há que ter em atenção algumas convenções que devemos entender acerca dos dados. A primeira questão é o separador – vírgula ou ponto e vírgula – este depende de o separador decimal ser virgula (,) ou ponto (.), isto é, se o seu computador interpreta na forma anglo-saxónica 10.5 como sendo dez e meio, então o seu separador decimal é o ponto e os ficheiros CSV podem usar a vírgula para separar os diferentes campos, se pelo contrário no seu computador dez e meio são representados por 10,5 então os seus ficheiros CSV devem ser separados por ponto e vírgula, para que o separador decimal não seja confundido com o separador de campos.

A tabela 6.1 ilustra exemplos destes dois tipos de formatos.

Tabela 6.1. Exemplos de ficheiros CSV com separador decimal vírgula e ponto.	
X, Y, NOME -9.14, 38.71, Lisboa -8.62, 41.16, Porto	X;Y;NOME -9,14; 38,71; Lisboa -8,62; 41,16; Porto
Ponto	Virgula

É de notar que as tabelas acima possuem uma linha de cabeçalho (*header*, em inglês), que se refere ao nome de cada campo, no caso concreto, X, Y e Nome.

Assim a função `read.csv` permite ler os ficheiros em que o separador decimal é o ponto e os valores estão separados por virgulas. A função `read.csv2` é utilizada no caso em que o separador decimal é a virgula e os valores estão separados por ponto e virgula.

Exercício 6.1.

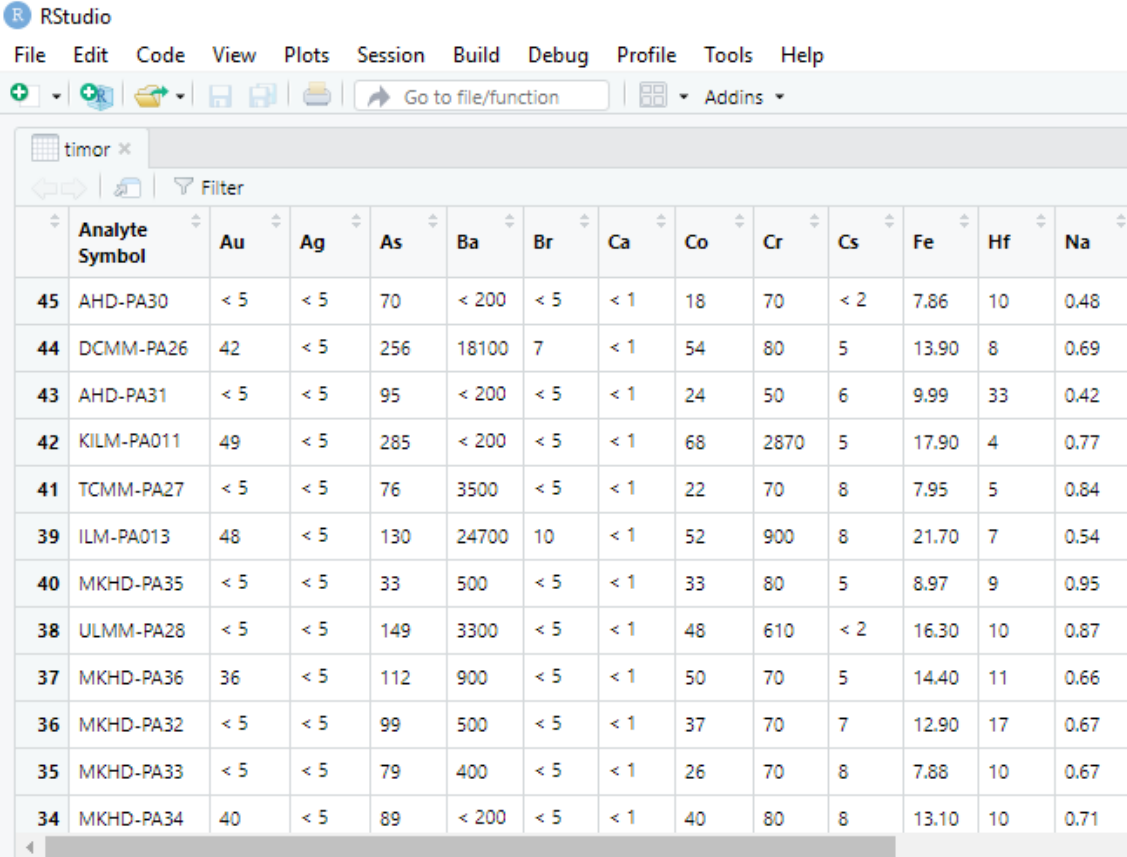
No conjunto de ficheiros de exercícios existem dois ficheiros, um designado `dados6-1.csv` e o outro `dados6-2.csv`.

Experimente utilizar as funções `read.csv` e `read.csv2` em cada um destes ficheiros e explique os resultados obtidos. Não se esqueça porem de acrescentar o argumento `header = TRUE` como no exemplo abaixo.

```
dados1 = read.csv("dados6-1.csv", header = TRUE)
```

Os dados externos devem ser importados para uma *data frame* e pode verificar se os dados foram corretamente importados através da função `View`. A função `head` também permite obter um resumo de alguns dados de uma variável. O RStudio abre um separador com a tabela de dados, como pode ver na figura 6.2.

```
View(dados1)
```



The screenshot shows the RStudio interface with a data frame view. The table has 13 columns: Analyte Symbol, Au, Ag, As, Ba, Br, Ca, Co, Cr, Cs, Fe, Hf, and Na. The rows are numbered 45 down to 34. The data values are as follows:

	Analyte Symbol	Au	Ag	As	Ba	Br	Ca	Co	Cr	Cs	Fe	Hf	Na
45	AHD-PA30	< 5	< 5	70	< 200	< 5	< 1	18	70	< 2	7.86	10	0.48
44	DCMM-PA26	42	< 5	256	18100	7	< 1	54	80	5	13.90	8	0.69
43	AHD-PA31	< 5	< 5	95	< 200	< 5	< 1	24	50	6	9.99	33	0.42
42	KILM-PA011	49	< 5	285	< 200	< 5	< 1	68	2870	5	17.90	4	0.77
41	TCMM-PA27	< 5	< 5	76	3500	< 5	< 1	22	70	8	7.95	5	0.84
39	ILM-PA013	48	< 5	130	24700	10	< 1	52	900	8	21.70	7	0.54
40	MKHD-PA35	< 5	< 5	33	500	< 5	< 1	33	80	5	8.97	9	0.95
38	ULMM-PA28	< 5	< 5	149	3300	< 5	< 1	48	610	< 2	16.30	10	0.87
37	MKHD-PA36	36	< 5	112	900	< 5	< 1	50	70	5	14.40	11	0.66
36	MKHD-PA32	< 5	< 5	99	500	< 5	< 1	37	70	7	12.90	17	0.67
35	MKHD-PA33	< 5	< 5	79	400	< 5	< 1	26	70	8	7.88	10	0.67
34	MKHD-PA34	40	< 5	89	< 200	< 5	< 1	40	80	8	13.10	10	0.71

Figura 6.2. Exemplo do separador com um *data frame* importada.

É de notar que estas duas funções são casos especiais da função `read.table`. Para saber mais pesquise na ajuda do R.

```
?read.table
```

Estas funções além dos parâmetros que já referimos permitem a definição de diferentes opções através de argumentos. Exemplos desses argumentos são:

- `sep`, que define o separador;
- `dec`, que define o símbolo de decimal;
- `col.names`, que permite definir o nome das colunas em alternativa quando os dados não tem cabeçalho;
- `quote`, que permite indicar qual o separador usado para identificar textos (exemplo: “”);
- `header`, que permite indicar se os dados possuem cabeçalho ou não.

Exercício 6.2.

Experimente agora ler os mesmos ficheiros com a função `read.table` com os argumentos corretos.

Além das funções de leitura de ficheiros de texto (csv) existem módulos que permitem ler diretamente os dados de um ficheiro de uma folha de cálculos, como o Microsoft Excel ou o Google Sheets.

O módulo sugerido é o `[readxl]` que é uma parte do pacote `[tidyverse]`.

Se já instalou o `[tidyverse]` pode ativar a biblioteca `[readxl]` através do comando:

```
library(readxl)
```

O comando para ler um ficheiro excel é exemplificado abaixo.

```
timor = read_excel("dados/exercicio6-3.xls")
```

Neste caso é criada a variável `timor` que importa os valores da folha `[timorStream.xls]` que está na subpasta `[Exercicios]` da pasta de trabalho. O resultado da importação é o apresentado na Figura 6.2. e trata-se de uma *data frame*.

Esta função possui diversos argumentos que parametrizam a forma como os dados são importados. De esses argumentos salientamos:

- `sheet`, que permite indicar qual a folha (*sheet*) no caso de ficheiros com mais de uma folha;
- `col_types`, que permite indicar quais os tipos de dados que cada coluna tem;
- `col_names`, valor lógico que indica se a primeira linha da folha contem os nomes das colunas.

7- ORGANIZAÇÃO DE DADOS

Um aspeto fundamental de um projeto R é a forma como os dados são organizados. A regra principal é, independentemente das operações que sejam aplicadas mantemos sempre a integridade dos dados originais. Assim, sempre que fizer uma operação no seu conjunto de dados, crie uma nova variável com o resultado obtido.

Um outro aspeto que deve ter em conta é o de guardar sempre os dados em variáveis com nomes que sejam reconhecidos facilmente. Sugerimos a abordagem com nomes no formato camelo, isto é começar com letra pequena e cada palavra nova começar com maiúscula. Por exemplo quando temos uma variável sobre timor que contém os dados de sedimentos de linha de água pode ser chamada de timorStream ou timorSedimentos.

Um outro aspeto fundamental é conhecer os seus dados. Em primeiro lugar após uma operação de importação de dados deve verificar sempre se estes foram importados corretamente. Um dos erros mais comuns é admitir que um conjunto de valores com casas decimais foram importados e estes foram importados como texto.

Vejamos o seguinte exemplo que utiliza a função `is.numeric(valor)` que retorna um valor lógico (TRUE ou FALSE) dependendo se o valor é numérico ou não.

```
> is.numeric(timor$`Analyte Symbol`)  
[1] FALSE  
> is.numeric(timor$Au)  
[1] FALSE  
> is.numeric(timor$Sm)  
[1] TRUE  
> typeof(timor$Au)  
[1] "character"
```

Exercício 7.1.

Explique os resultados obtidos no exemplo acima.

Relativamente aos dados importados no exemplo da variável [timor] nem todos os valores foram importados como numéricos. O caso de [Analyte Symbol] era expectável pois todos os valores eram de texto. No caso do Au o resultado já é menos claro, mas atendendo a que em algumas células o que tem escrito é "<5" facilmente se compreende que tal não vai corresponder a um valor numérico e, portanto, toda a coluna vai ser importada como texto.

Para calcular o valor da média dos teores de Au na variável correspondente deve-se recorrer a uma função de coerção que force a transformação de um valor de um determinado tipo noutra tipo. No caso de transformar valores texto em numéricos pode ser usada a função `as.numeric(valor)`. O resultado obtido é o seguinte:

```

> as.numeric(timor$Au)
 [1]  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  NA  49  NA  48 1060  165
NA  NA  NA  NA  NA  NA
 [22]  NA  NA  14  NA  42  NA  NA  NA  NA  NA  NA  NA  NA  40  NA  36
NA  24  NA 198  87  NA
 [43]  NA 1160  NA
Warning message:
NAs introduced by coercion

```

Muitos valores aparecem com a designação NA o que significa não tem valor. Para calcular o valor da média deve ser utilizada a função `mean`. O exemplo abaixo ilustra duas formas de aplicar esta função.

```

> mean(as.numeric(timor$Au))
 [1] NA
Warning message:
In mean(as.numeric(timor$Au)) : NAs introduced by coercion
> mean(as.numeric(timor$Au), na.rm = TRUE)
 [1] 243.5833
Warning message:
In mean(as.numeric(timor$Au), na.rm = TRUE) : NAs introduced by coercion

```

Exercício 7.2

Explique os resultados obtidos com os dois exemplos e as diferenças.

Qual a média dos valores de Au nas nossas amostras?

Sugira uma forma de determinar quantos valores foram utilizados no cálculo da média.

Existem funções de coerção entre os diversos tipos de dados, naturalmente a coerção só é possível quando os dados são compatíveis. Vejamos alguns exemplos.

```

> inteiro = c(1,3,5,7,23)
> duplo = c(2.35, 3.54, 7.17, 9.23)
> as.double(inteiro)
 [1]  1  3  5  7 23
> as.integer(duplo)
 [1] 2 3 7 9
i = as.double(inteiro)
d = as.integer(duplo)
typeof(inteiro)
typeof(duplo)
typeof(i)
typeof(d)

```

Exercício 7.3

Explique os resultados.

Mais uma vez é necessário ter cuidado com os resultados da coerção pois nem sempre o senso comum ajuda a entender o resultado. Atenemos agora a alguns exemplos.

```
> texto = c("um", 2, "pedro", 6.3)
> as.integer(texto)
[1] NA  2 NA  6
Warning message:
NAs introduced by coercion
> as.double(texto)
[1] NA 2.0 NA 6.3
Warning message:
NAs introduced by coercion
> as.factor(texto)
[1] um    2    pedro 6.3
Levels: 2 6.3 pedro um
```

Exercício 7.4.

Explique os resultados obtidos.

Qual o resultado que obtém com:

```
as.character(texto)
```

Explique e discuta.

Pode ser criada uma coluna do tipo numérico na *data frame* [timor] com os valores de Au. Se a nova variável for chamada Au_N então o comando será:

```
> timor$Au_N = as.numeric(timor$Au)
Warning message:
NAs introduced by coercion
```

Verifique que há valores NA que são introduzidos por coerção.

Um outro aspeto importante na organização dos dados é acrescentar informação que permita agrupar elementos consoante as suas afinidades. Por exemplo no nome das referências das amostras de [timor], as duas primeiras letras correspondem ao setor de estudo. Assim podemos criar um campo com apenas as duas primeiras letras da referência da amostra para identificar o setor de estudo.

Para isso devemos recorrer-se a uma função de texto (*string*, em inglês) que permita recortar essas letras, e criar um novo campo. A função que seleciona uma parte de um texto é `substr`. Assim, no exemplo abaixo ilustra-se a criação do campo [setor] que é feita através da seguinte expressão:

```
> substr(timor$`Analyte Symbol`,1,2)

 [1] "TB" "TB" "TB" "TB" "TB" "TB" "TB" "TB" "TB" "TB" "KI" "IL" "IL" "KB" "BM" "BM" "BM" "BM" "UM"
 "BM" "SM"

 [22] "UL" "SL" "AC" "DC" "DC" "TC" "UL" "UL" "AH" "AH" "MK" "MK" "MK" "MK" "MK" "MV" "MV" "MC" "MV"
 "MV" "LL"

 [43] "LL" "LC" "LC"

> timor$setor = substr(timor$`Analyte Symbol`,1,2)
```

Note que o primeiro comando apenas ilustra o resultado e o segundo cria o vetor [setor].

8- TRATAMENTO DE DADOS

Os dados depois de lidos, verificados e organizados de acordo com as necessidades do projeto devem ser tratados por forma a se procurar extrair informação dos mesmos. Uma das primeiras abordagens é através da estatística descritiva com recurso a funções elementares. Algumas das funções de estatística descritiva estão sumarizadas nas cabulas do anexo 1.

Funções de tendência central

As medidas de tendência central como a média, a mediana ou o desvio padrão, podem ser calculadas diretamente através das respetivas funções. O exemplo abaixo ilustra, para o caso de [timor], o cálculo dos valores de Sm (Samário) nas diferentes amostras.

```
> mean(timor$Sm)
[1] 7.277778
> median(timor$Sm)
[1] 6.7
> sd(timor$Sm)
[1] 2.264939
```

Além destes valores pode obter os diferentes percentis através da respetiva função, assim como determinar máximos e mínimos de um vetor.

Exercício 8.1.

Recorrendo à função existente nas cábulas, calcule os percentis 25, 50 e 75 do Samário nas amostras [timor].

Função *summary*

Além das funções anteriormente exemplificadas pode utilizar funções agregadoras que permitem ver um *data frame* como um todo. A função mais elementar é `summary`, que permite num relance ver as principais medidas de tendência central.

Um exemplo do resultado desta função é o apresentado abaixo.

```
> summary(timor)
Analyte Symbol      Au          Ag          As          Ba
Length:45          Length:45          Length:45          Length:45
Class :character   Class :character   Class :character   Class :character
Mode  :character   Mode  :character   Mode  :character   Mode  :character

      Br          Ca          Co          Cr          Cs          Fe
Length:45          Length:45          Min.   : 8.00          Min.   : 40          Length:45          Min.   :
3.88
Class :character   Class :character   1st Qu.: 23.00          1st Qu.: 70          Class :character   1st Qu.:
7.02
Mode  :character   Mode  :character   Median : 33.00          Median : 160          Mode  :character   Median
:10.30
```

			Mean : 41.07	Mean : 2473		Mean :13.36
			3rd Qu.: 55.00	3rd Qu.: 870		3rd Qu.:18.50
			Max. :117.00	Max. :35800		Max. :27.70
Hf	Na	Ni	Rb	Sb	Sc	
Min. : 4.00 :10.10	Min. :0.2500	Length:45	Length:45	Length:45	Min.	
1st Qu.: 6.00 Qu.:14.80	1st Qu.:0.6600	Class :character	Class :character	Class :character	1st	
Median : 10.00 :19.60	Median :0.7100	Mode :character	Mode :character	Mode :character	Median	
Mean : 21.24 :22.54	Mean :0.7602				Mean	
3rd Qu.: 22.00 Qu.:25.40	3rd Qu.:0.8600				3rd	
Max. :125.00 :50.40	Max. :1.7500				Max.	
Ta	Th	U	Zn	La	Ce	
Length:45 24.00	Length:45	Length:45	Length:45	Min. : 7	Min. :	
Class :character 66.00	Class :character	Class :character	Class :character	1st Qu.:27	1st Qu.:	
Mode :character 82.00	Mode :character	Mode :character	Mode :character	Median :32	Median :	
84.51				Mean :35	Mean :	
98.00				3rd Qu.:41	3rd Qu.:	
:153.00				Max. :81	Max.	
Nd	Sm	Eu	Yb	Lu	Mass	
Length:45	Min. : 2.600	Length:45	Min. : 2.200	Min. :0.100	Min. : 1.920	
Class :character	1st Qu.: 6.100	Class :character	1st Qu.: 3.600	1st Qu.:0.640	1st Qu.: 2.290	
Mode :character	Median : 6.700	Mode :character	Median : 4.800	Median :0.750	Median : 8.800	
	Mean : 7.278		Mean : 7.293	Mean :1.114	Mean : 7.336	
	3rd Qu.: 8.400		3rd Qu.: 8.900	3rd Qu.:1.160	3rd Qu.:10.600	
	Max. :13.900		Max. :21.400	Max. :3.700	Max. :14.500	
Au_N	setor					
Min. : 14.0	Length:45					
1st Qu.: 39.0	Class :character					
Median : 48.5	Mode :character					
Mean : 243.6						
3rd Qu.: 173.2						
Max. :1160.0						
NA's :33						

Naturalmente esta leitura pode ser um pouco difícil quando se tem muitas variáveis.

Exercício 8.2.

Explique os diversos resultados da função acima referida.

Agregar colunas ou linhas

Por vezes é necessário agregar dados, ou criar novas *data frames* a partir de outras variáveis. O R possui funções que permitem fazer esta manipulação de informação de uma forma rápida e eficiente.

Para criar uma *data frame* com os valores de cada setor de La e Lu do exemplo [timor] deve-se começar por utilizar a função `cbind` (*column bind*, do inglês) que permite agregar colunas de um *data frame*. O exemplo seguinte ilustra a sua utilização através da criação de uma *data frame* designada [timorLaLu].

```
timorLaLu = as.data.frame(cbind(timor$setor,timor$La, timor$Lu))
```

Neste caso a função `cbind` não atribui nomes às diferentes colunas, dando por omissão a designação de V1, V2,... Para atribuir nomes às colunas utiliza-se a função `colnames`, como é descrito no exemplo seguinte.

```
colnames(timorLaLu) = c("Setor", "La (ppm)", "Lu (ppm)")
```

Além da função para agregar variáveis/colunas a uma variável existe a função equivalente para agregar linhas/observações a uma variável. Essa função, naturalmente é, `rbind`.

Se pretender eliminar uma coluna basta simplesmente utilizar a expressão NULL, como pode ver no seguinte exemplo.

```
timorLaLu["Setor"] = NULL
```

Cálculo agregado

Uma das formas de se poder retirar mais informação dos dados é analisá-los por setor (no exemplo de [timor]). Para isso pode-se utilizar a função `aggregate` que permite aplicar uma função (por exemplo `mean`) agrupando os valores por uma variável [setor] no nosso caso.

O resultado obtido é o seguinte.

```
> aggregate(timor , by=list(timor$setor) , FUN=mean, na.rm=TRUE)
```

Group.1	Analyte	Symbol	Au	Ag	As	Ba	Br	Ca	Co	Cr	Cs	Fe	Hf	Na	Ni	Rb	Sb
1	AC		NA	NA	NA	NA	NA	NA	43.00	800.0000	NA	10.30000	7.000000	1.0900	NA	NA	NA
23.30	NA																
2	AH		NA	NA	NA	NA	NA	NA	21.00	60.0000	NA	8.92500	21.500000	0.4500	NA	NA	NA
11.10	NA																
3	BM		NA	NA	NA	NA	NA	NA	68.20	478.0000	NA	25.68000	5.800000	0.3980	NA	NA	NA
47.44	NA																
4	DC		NA	NA	NA	NA	NA	NA	38.50	70.0000	NA	10.04000	6.000000	0.7750	NA	NA	NA
14.15	NA																

5	IL	NA NA NA NA NA NA NA NA	40.00	710.0000	NA	16.20000	6.000000	0.5800	NA NA NA
23.70	NA								
6	KB	NA NA NA NA NA NA NA NA	72.00	120.0000	NA	26.10000	24.000000	0.2500	NA NA NA
24.90	NA								
7	KI	NA NA NA NA NA NA NA NA	68.00	2870.0000	NA	17.90000	4.000000	0.7700	NA NA NA
27.20	NA								
8	LC	NA NA NA NA NA NA NA NA	25.00	1045.0000	NA	9.21500	11.000000	1.1950	NA NA NA
18.15	NA								
9	LL	NA NA NA NA NA NA NA NA	23.50	1110.0000	NA	9.37500	9.500000	1.2800	NA NA NA
20.00	NA								
10	MC	NA NA NA NA NA NA NA NA	55.00	11500.0000	NA	15.90000	45.000000	1.0000	NA NA NA
25.20	NA								
11	MK	NA NA NA NA NA NA NA NA	37.20	74.0000	NA	11.45000	11.400000	0.7320	NA NA NA
22.40	NA								
12	MV	NA NA NA NA NA NA NA NA	82.25	20975.0000	NA	21.57500	110.000000	0.6425	NA NA NA
23.05	NA								
13	SL	NA NA NA NA NA NA NA NA	47.00	1150.0000	NA	11.20000	9.000000	1.1100	NA NA NA
19.60	NA								
14	SM	NA NA NA NA NA NA NA NA	79.00	530.0000	NA	23.90000	6.000000	0.6700	NA NA NA
46.20	NA								
15	TB	NA NA NA NA NA NA NA NA	17.60	56.0000	NA	5.78800	18.800000	0.7400	NA NA NA
13.01	NA								
16	TC	NA NA NA NA NA NA NA NA	22.00	70.0000	NA	7.95000	5.000000	0.8400	NA NA NA
16.30	NA								
17	UL	NA NA NA NA NA NA NA NA	33.00	286.6667	NA	10.72333	8.333333	1.1800	NA NA NA
15.50	NA								
18	UM	NA NA NA NA NA NA NA NA	35.00	160.0000	NA	18.50000	9.000000	0.7600	NA NA NA
39.60	NA								

	Th	U	Zn	La	Ce	Nd	Sm	Eu	Yb	Lu	Mass	Au_N	setor
1	NA	NA	NA	31.00000	66.00000	NA	6.300000	NA	3.30	0.5600000	2.290000	14	NA
2	NA	NA	NA	47.00000	100.50000	NA	8.050000	NA	5.80	0.8450000	5.520000	NaN	NA
3	NA	NA	NA	19.20000	59.40000	NA	6.340000	NA	15.60	2.2440000	12.552000	165	NA
4	NA	NA	NA	29.50000	57.50000	NA	5.950000	NA	2.40	0.4100000	2.105000	42	NA
5	NA	NA	NA	28.50000	63.50000	NA	6.000000	NA	3.60	0.6400000	8.890000	48	NA
6	NA	NA	NA	7.00000	24.00000	NA	2.600000	NA	4.30	0.7800000	12.400000	1060	NA
7	NA	NA	NA	28.00000	70.00000	NA	6.500000	NA	4.80	0.7300000	8.530000	49	NA
8	NA	NA	NA	29.00000	79.00000	NA	6.100000	NA	4.55	0.7250000	11.350000	1160	NA
9	NA	NA	NA	30.50000	67.00000	NA	6.400000	NA	4.85	0.8150000	10.750000	NaN	NA
10	NA	NA	NA	25.00000	98.00000	NA	9.700000	NA	11.90	1.9100000	12.200000	NaN	NA
11	NA	NA	NA	63.20000	122.40000	NA	11.200000	NA	6.12	0.9260000	2.128000	38	NA
12	NA	NA	NA	32.50000	113.00000	NA	10.000000	NA	17.75	3.0525000	9.082500	103	NA
13	NA	NA	NA	29.00000	54.00000	NA	5.300000	NA	2.90	0.4700000	2.350000	NaN	NA
14	NA	NA	NA	19.00000	41.00000	NA	5.700000	NA	13.30	1.2400000	2.430000	NaN	NA
15	NA	NA	NA	37.10000	93.10000	NA	6.100000	NA	4.34	0.6930000	8.479000	NaN	NA
16	NA	NA	NA	31.00000	82.00000	NA	6.500000	NA	2.60	0.4900000	5.950000	NaN	NA
17	NA	NA	NA	38.33333	82.33333	NA	7.166667	NA	3.60	0.6566667	3.276667	NaN	NA
18	NA	NA	NA	48.00000	94.00000	NA	9.700000	NA	8.90	0.1000000	2.380000	NaN	NA

There were 50 or more warnings (use warnings() to see the first 50)

Exercício 8.3.

Explique os diversos resultados da função acima referida.

9- GRÁFICOS: DISPERSÃO, LINHAS BARRAS E HISTOGRAMAS

A melhor forma de representar um conjunto grande de dados é através de um gráfico. Desta forma, podem ser analisadas visualmente as relações entre os elementos representados. Existem gráficos de diferentes tipos consoante os dados que são representados. A escolha do tipo de gráfico deve depender do que se pretende analisar e do tipo de dados que possuímos.

Gráficos de dispersão e de linhas

O software R é especialmente adequado para efetuar representações gráficas de carácter científico. A função `plot` assume um papel fundamental nas representações gráficas em R.

Abaixo está apresentado um exemplo simples desta função.

```
inteiro = c(1, 3, 5, 7, 23)
duplo = c(2.35, 3.54, 7.17, 9.23, 17.88)
plot(x = inteiro, y = duplo)
```

O resultado é apresentado na subjanela [Plots], como está ilustrado na Figura 9.1.

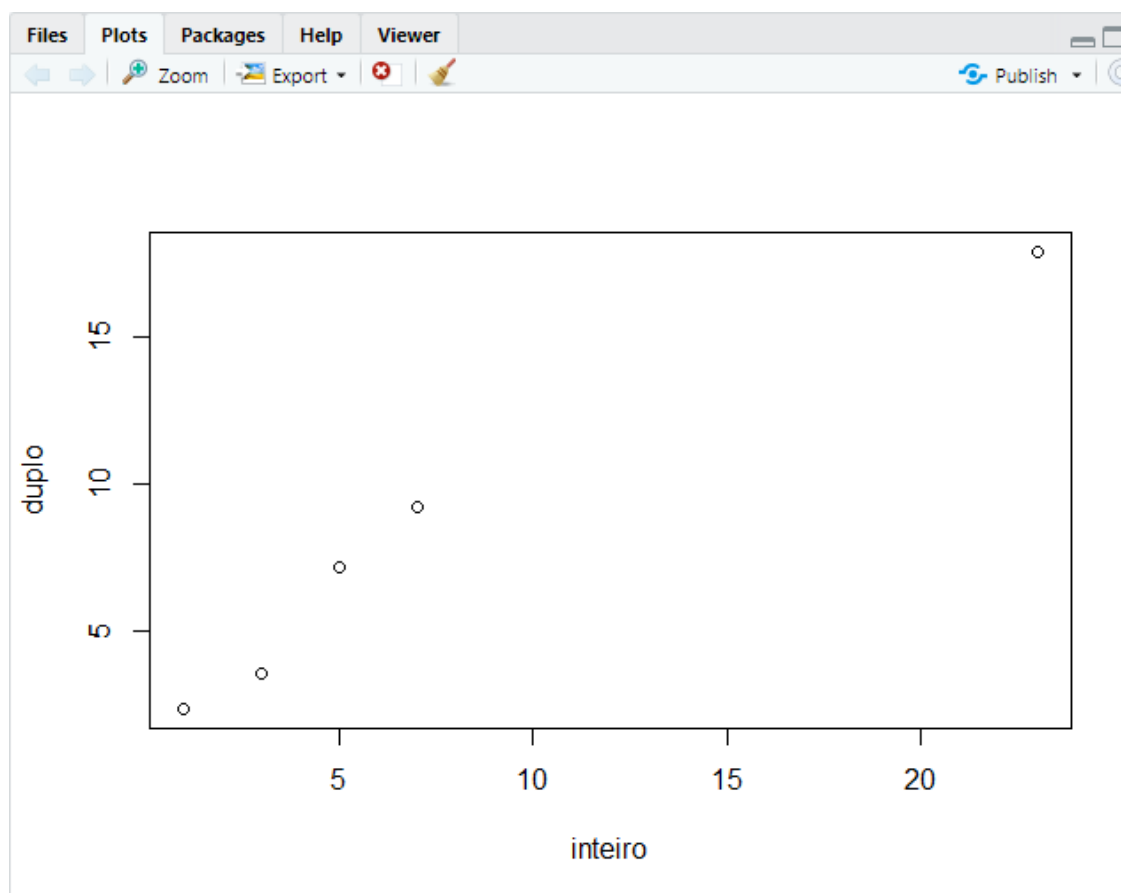


Figura 9.1. Exemplo de um gráfico de dispersão (x, y) simples.

A função `plot` possui um conjunto alargado de argumentos que permitem a configuração completa do gráfico, desde os eixos, a legenda, os símbolos apresentados até ao título do gráfico e as cores.

Alguns destes argumentos e um exemplo mais completo.

- `type`, qual o tipo de gráfico que se pretende:
 - “p” para pontos,
 - “l” para linhas,
 - “b” para ambos,
 - “c” para apenas a parte da linha,
 - “o” para a linha sobreposta aos pontos,
 - “h” para semelhante a histograma ou linhas verticais,
 - “s” para degraus terminando no último valor,
 - “S” para degraus terminando no penúltimo valor
 - “n” for no plotting.

- `main`, o título do gráfico;
- `sub`, o subtítulo do gráfico;
- `xlab`, o título do eixo dos XX;
- `ylab`, o título do eixo dos YY;
- `pch`, para o símbolo dos pontos;
- `lty`, para o tipo de linha.

Um exemplo mais completo da utilização de alguns dos argumentos referidos.

```
plot(x = inteiro, y = duplo, type = "o", lty = "dashed", pch = 15,  
main = "Exemplo de gráfico")
```

O resultado obtido é o apresentado na figura 9.2.

Exercício 9.1.

Discuta os argumentos utilizados e o resultado obtido.

Exemplo de gráfico

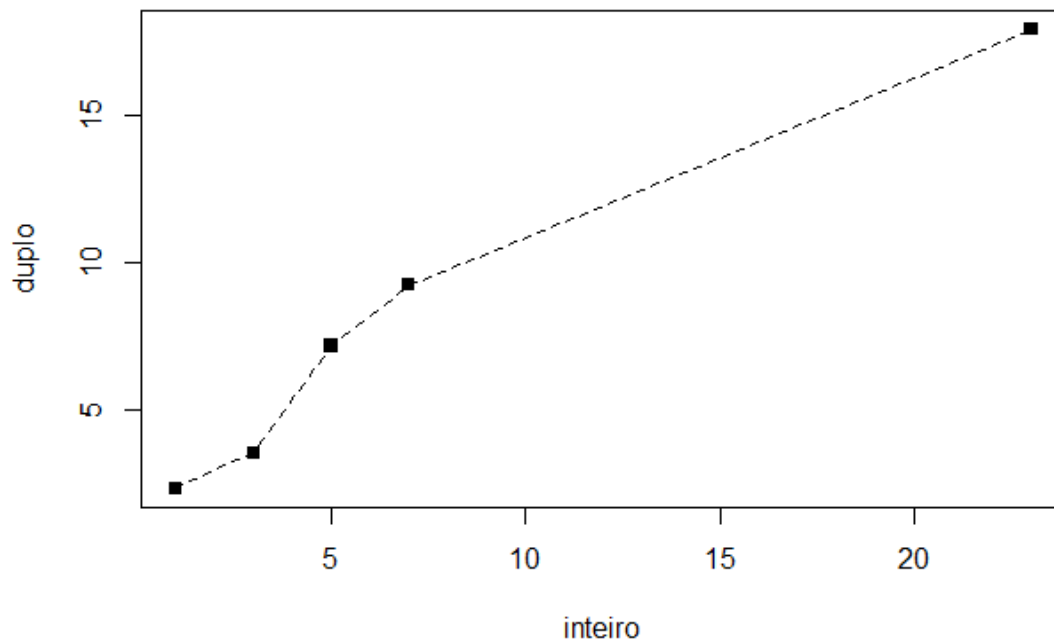


Figura 9.2. Exemplo de gráfico XY com algumas configurações.

Muitas outras configurações são possíveis para os gráficos de dispersão em R. Pode saber mais em:

https://www.tutorialspoint.com/r/r_scatterplots.htm

<https://www.r-bloggers.com/how-to-plot-a-graph-in-r/>

Gráficos de funções

Pode também utilizar a função `plot` para fazer gráficos de uma função. Abaixo encontra o exemplo de um gráfico de uma recta e de uma função exponencial.

```
# Define os valores de x
x1 = seq(-2,2,.1)
# Linha reta
y1 = x1
# função exponencial
y2 = x1^2

# Desenha o gráfico da primeira linha
plot(x1,y1,type="l", col="darkgreen")
# Desenha a segunda linha
lines(x1,y2,col="red")
# desenha uma grelha
```

```
grid( col = "lightgray", lty = "dotted")
```

Este exemplo resulta no gráfico da figura 9.3.

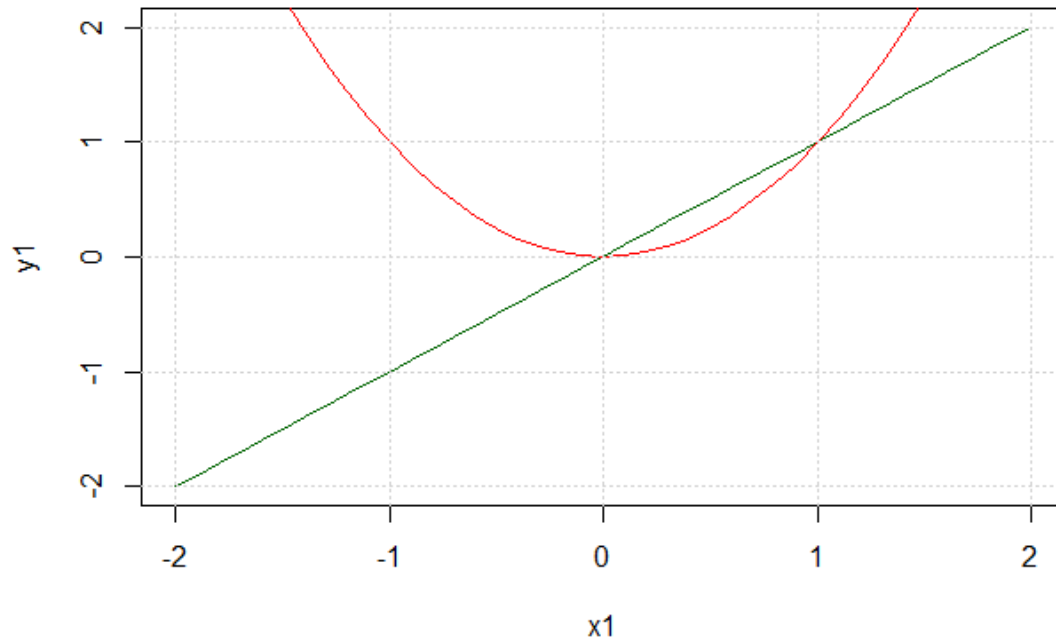


Figura 9.3. O exemplo do gráfico de duas funções.

Exercício 9.2.

Explique os resultados obtidos.

Faça o gráfico das funções $\sin(x)$ e $\sin(x)/x$

Gráficos de barras

Além dos gráficos de dispersão e de linhas o R permite a construção de gráficos de barras simples através da função `barplot`, que funciona, em quase tudo, igual à função `plot`. Vejamos um exemplo do seu funcionamento.

```
# Gráfico de barras
t <- c(7, 12, 28, 32, 41)
mes <- c("Mar", "Abr", "Mai", "Jun", "Jul")
barplot(t, names.arg= mes )
```

O resultado obtido é o apresentado na figura 9.4.

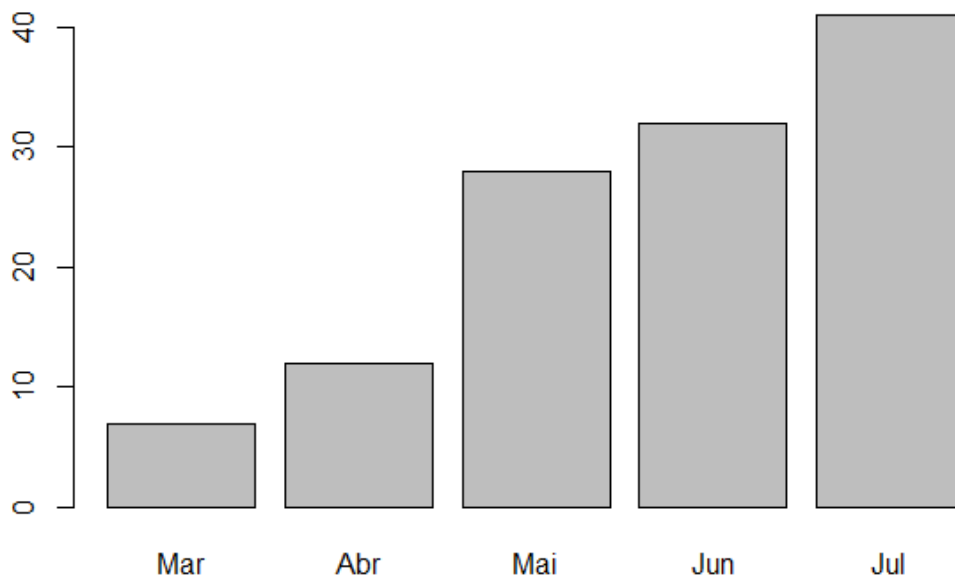


Figura 9.4. Exemplo de um gráfico de barras simples.

Muitos dos argumentos que são os mesmos da função `plot`. No caso do `barplot` pode ainda decidir a orientação das barras (horizontal ou vertical) utilizando o argumento abaixo.

```
barplot(t, names.arg= mes, horiz= TRUE)
```

O resultado é semelhante ao anterior, mas desta vez com as colunas na horizontal.

Existe ainda a possibilidade de se criar gráficos de barras empilhados (*stacked*, em inglês). Para tal é necessária alguma preparação dos dados. Veja o exemplo seguinte.

```
# Gráfico de barras
homens = c(14.2, 12.4, 13.5, 13.9, 14.3)
mulheres = c(15.3, 13.2, 14.7, 15.4, 16.2)
ano = c("2010", "2011", "2012", "2013", "2014")

todos=data.frame(Homens=homens, Mulheres=mulheres, row.names = ano)
barplot(t(as.matrix(todos)), beside = TRUE, names.arg=ano, col=c("blue", "pink")
)
```

O resultado é o que está apresentado na Figura 9.5.

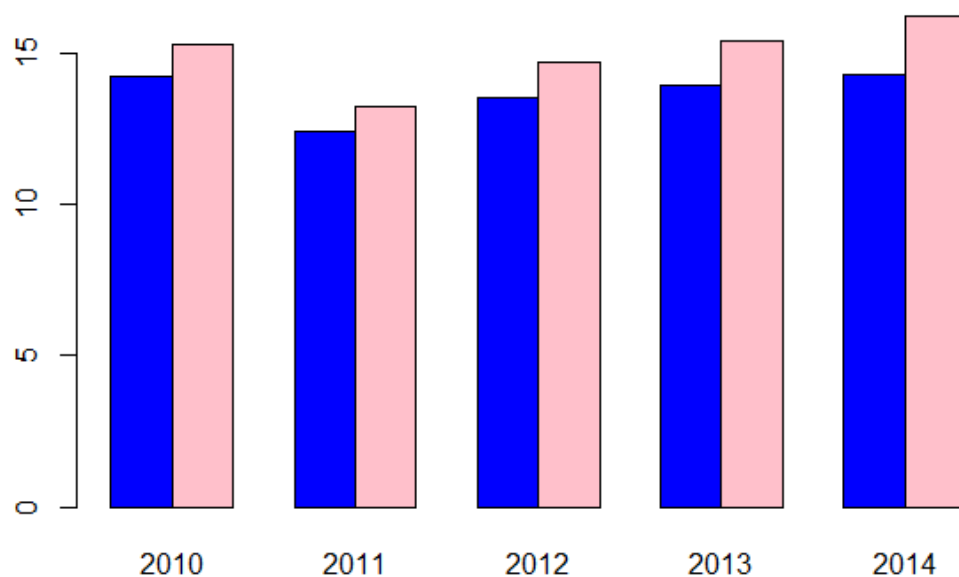


Figura 9.5. Exemplo de gráfico de barras empilhado.

Exercício 9.3.

Comente os resultados obtidos.

Explique o significado da seguinte operação:

```
todos=data.frame(Homens=homens,Mulheres=mulheres, row.names = ano)
```

Explique o significado dos argumentos:

```
t(as.matrix(todos))
col=c("blue","pink")
```

Histogramas

Os histogramas são uma categoria especial de gráficos de barras que tratam de representar a frequência de uma ou mais variáveis. O R tem uma função especial para lidar com os histogramas, a função `hist`.

Regressando ao exemplo de [timor] suponha que pretende representar em histograma os valores do Lantânio (La). Para tal utilize o seguinte código.

```
hist(timor$La)
```

O resultado é o da Figura 9.6.

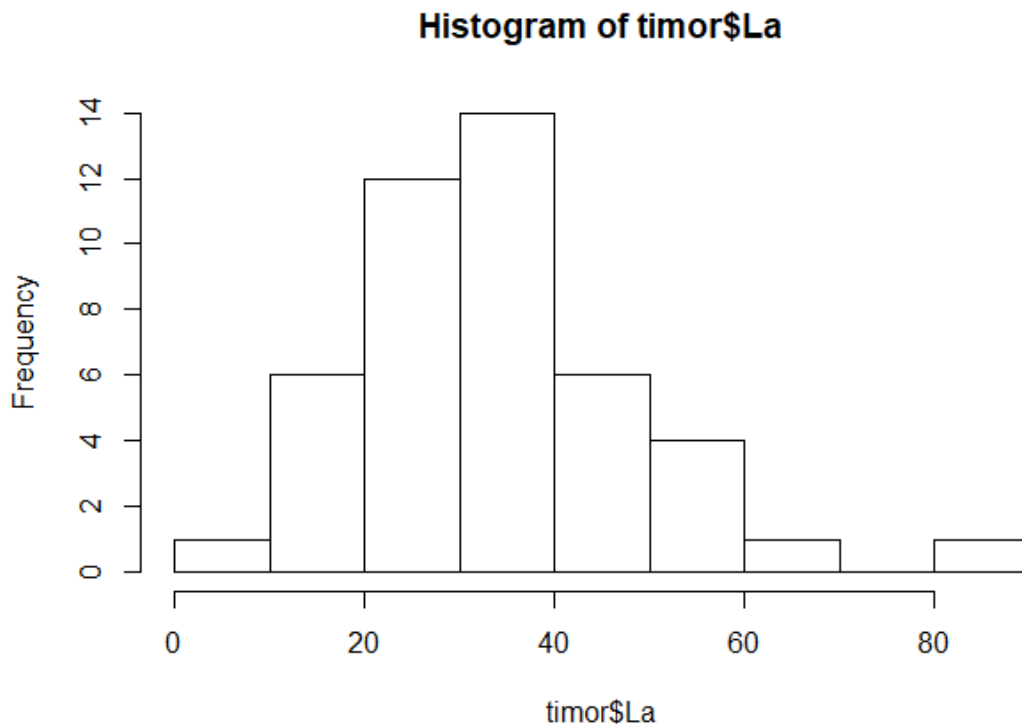


Figura 9.6. Exemplo de histograma com a frequência dos valores de La nos dados [timor]

O gráfico representa a frequência com que cada grupo de valores de La ocorrem. No caso dividido em classes de 10 em 10 ppm.

A função `hist` permite alterar os parâmetros e tratar a informação do histograma de forma a que este possa ser ajustado às análises a realizar. Além disso permite que em vez da frequência (contagem de valores por classe) as probabilidades associadas a cada classe. Com base nesta abordagem é possível traçar as linhas com as funções de densidade.

Atente no seguinte Código exemplo.

```
hist(timor$La, prob=TRUE, col="grey")
lines(density(timor$La), col="blue", lwd=2)
lines(density(timor$La, adjust=2), lty="dotted", col="darkgreen", lwd=2)
lines(density(timor$La, adjust=3), lty="dashed", col="red", lwd=2)
```

O resultado obtido é o apresentado na Figura 9.7.

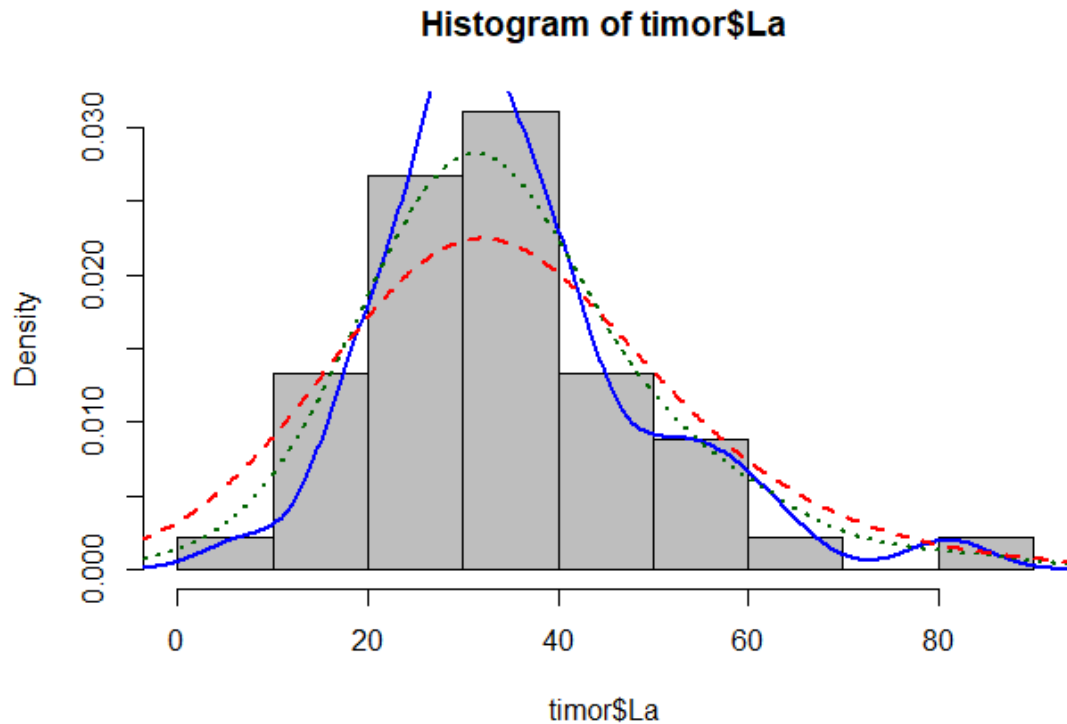


Figura 9.7. Exemplo de histograma onde foram acrescentadas as funções densidade com diferentes parâmetros.

Exercício 9.3.

Comente e explique os parâmetros utilizados na função `hist` e `density`.

A função `hist` pode ser utilizada para criar uma variável com parâmetros ajustáveis. O exemplo abaixo mostra alguns parâmetros de configuração de um histograma, permitindo extrair as frequências, as classes, etc.

```
> LaHist = hist(timor$La, breaks= seq(0,100,5), col = "khaki3", prob = TRUE )
> LaHist$breaks
 [1]  0  5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90
95 100
> LaHist$counts
 [1] 0 1 0 6 2 10 8 6 4 2 2 2 1 0 0 0 1 0 0 0
> LaHist$density
 [1] 0.000000000 0.004444444 0.000000000 0.026666667 0.008888889 0.044444444
0.035555556 0.026666667
 [9] 0.017777778 0.008888889 0.008888889 0.008888889 0.004444444 0.000000000
0.000000000 0.000000000
[17] 0.004444444 0.000000000 0.000000000 0.000000000
> LaHist$mids
 [1]  2.5  7.5 12.5 17.5 22.5 27.5 32.5 37.5 42.5 47.5 52.5 57.5 62.5 67.5 72.5
77.5 82.5 87.5 92.5 97.5
```

A função `hist` cria uma *data frame* com um conjunto de variáveis que correspondem aos parâmetros para desenhar o histograma.

Exercício 9.4.

Adapte a função `barplot` para desenhar o histograma para a variável [LaHist].

```
>barplot(LaHist$counts, names.arg= LaHist$breaks[-1], col="khaki4")
```

10- ETIQUETAS, GRELHAS, LEGENDAS, LINHAS E TEXTOS

Os gráficos em R podem ser configurados até ao mais ínfimo pormenor incluindo todas as definições que um gráfico deve ter. Procurando seguir uma ordem lógica vamos passo por passo definir todos os elementos para criar um gráfico de qualidade para publicação.

Eixos, nomes dos eixos e título

Para demonstrar esta parametrização vamos projetar as variáveis La e Lu (Lantânio e Lutécio) da *data frame* [timor].

A função básica é:

```
plot(timor$La, timor$Lu)
```

O resultado obtido é o da Figura 10.1.

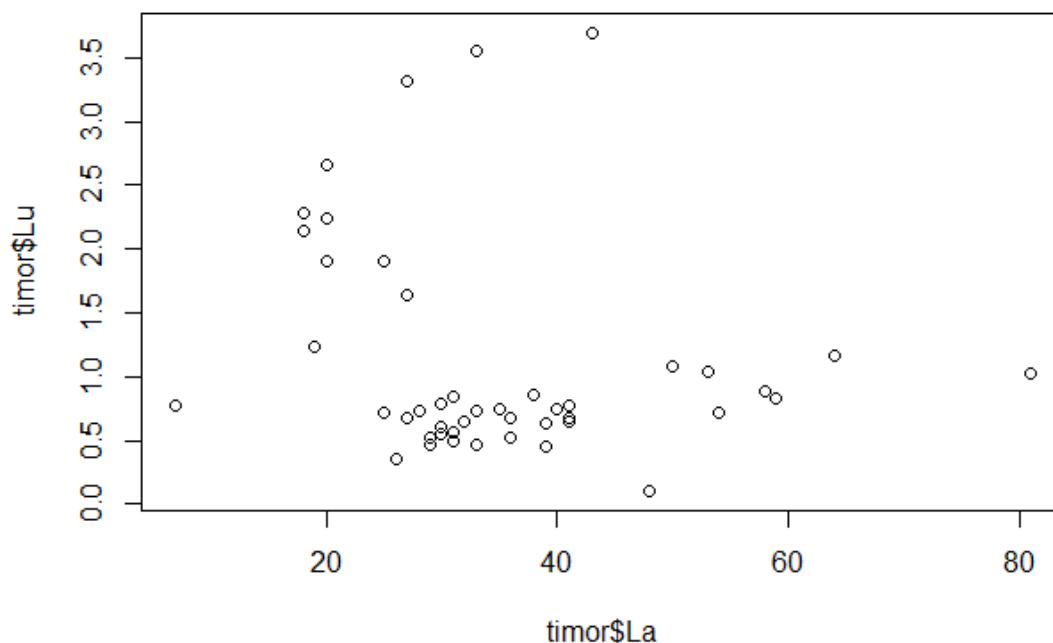


Figura 10.1. Gráfico de dispersão La vs Lu.

O R automaticamente dá o nome dos eixos igual ao nome das variáveis e calcula os valores dos eixos XX e YY de acordo com os máximos e mínimos de cada uma das variáveis.

Estas definições por omissão podem ser alteradas com os seguintes argumentos.

```
plot(timor$La, timor$Lu,  
     xlab="La (ppm)", ylab= "Lu (ppm)",  
     xlim=c(0,100), ylim=c(0,5),  
     main= "La vs Lu")
```


O resultado obtido agora é o da Figura 10.2.

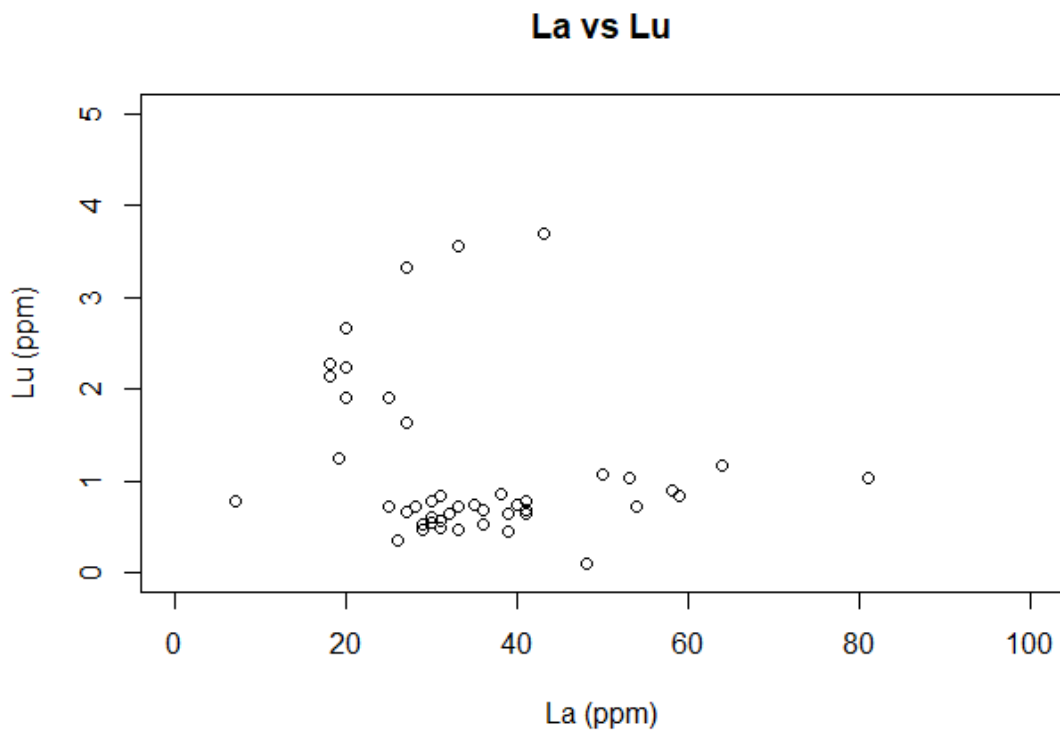


Figura 10.2. Configuração dos eixos, etiquetas de eixos e título.

Exercício 10.1.

Explique os argumentos da função acima.

Grelhas

As grelhas ajudam a tornar um gráfico mais legível. A função `grid` permite acrescentar uma grelha a um gráfico (`plot`). O exemplo seguinte mostra o uso da função `grid`.

```
grid(lty = "dotted", col="grey")
```

A Figura 10.3. mostra o resultado desta função. Note que a linha é do tipo ponteadado (*dotted*) e a cor utilizada é cinzento.

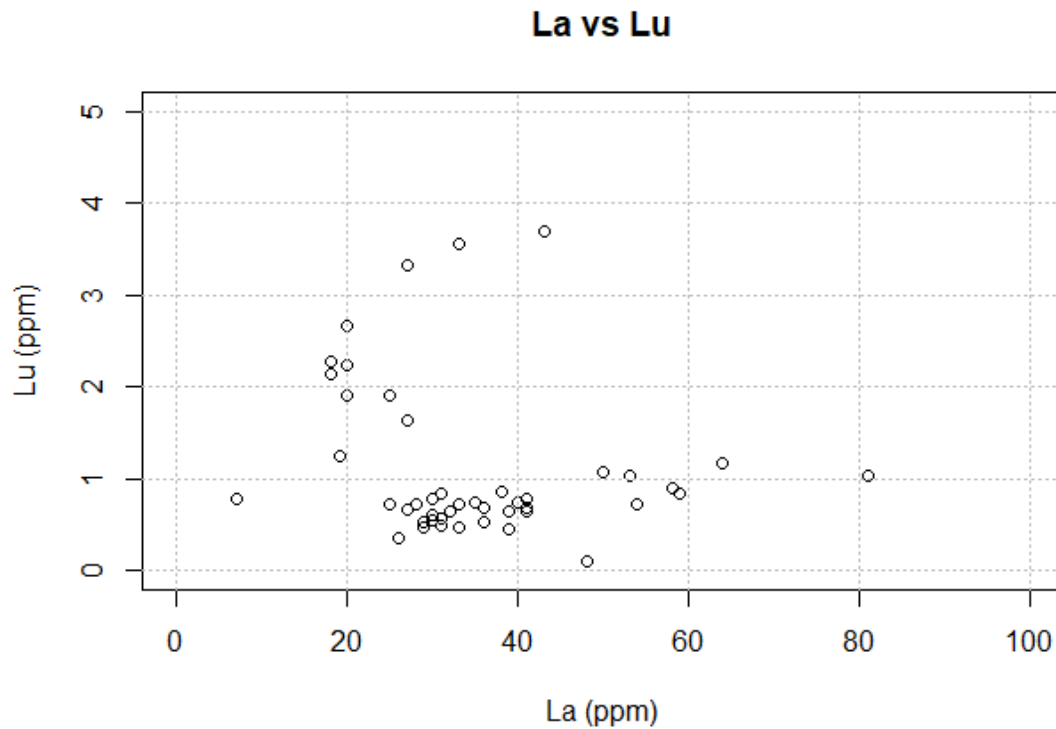


Figura 10.3. O resultado da função `grid` com parâmetros por omissão.

Exercício 10.2.

Crie uma grelha apenas para o eixo dos XX.

Legendas

As legendas permitem em gráficos com diferentes símbolos ilustrar o que corresponde a cada símbolo.

O exemplo seguinte projeta dois grupos de elementos, o Lutécio e o Itérbio contra o Lantânio.

```
plot(timor$La,timor$Yb,
      xlab="La (ppm)", ylab= "ppm",
      xlim=c(0,100), main="La vs Yb e Lu", pch=1)
points(timor$La, timor$Lu, pch=15)
legend(70, 20, legend=c("Yb (ppm)","Lu (ppm)"), pch=c(1,15), title="Elementos")
```

O resultado desta projeção é apresentado na Figura 10.4.

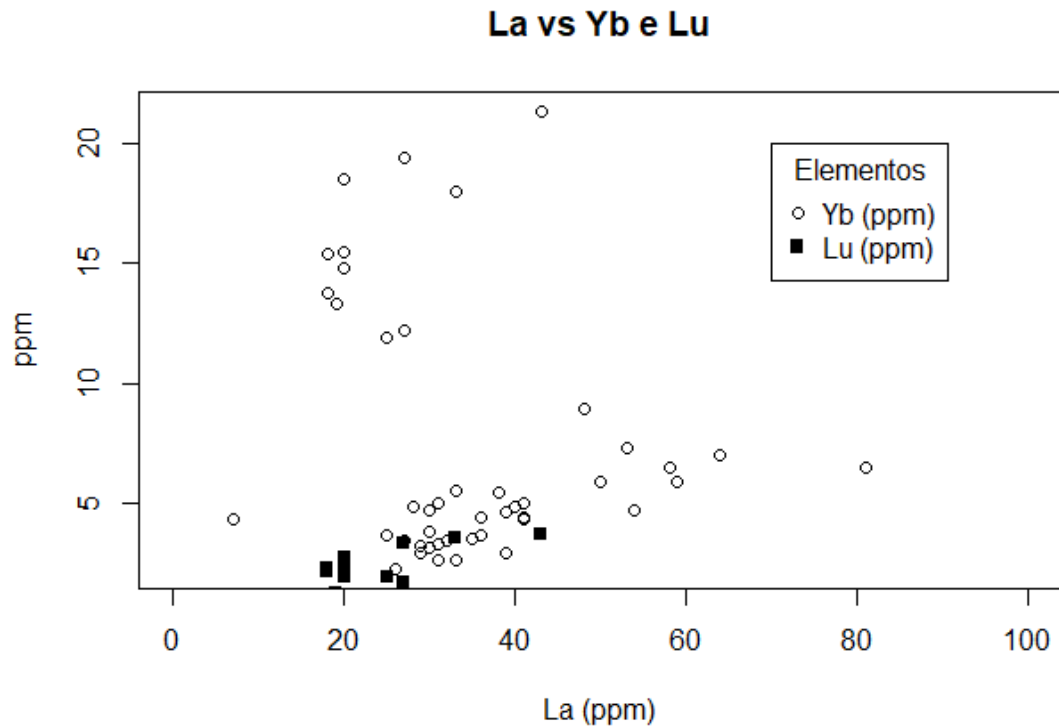


Figura 10.4. Gráfico de dispersão do La vs o Yb e o Lu. Acrescentou-se a legenda.

Note-se que para se ter duas séries de valores recorreu-se à função `points` que acrescenta uma nova série de pontos ao gráfico anteriormente criado. Os argumentos são semelhantes aos utilizados na função `plot`.

É necessário ter atenção que os novos pontos são projetados no gráfico já desenhado. Os limites do eixo dos XX e do eixo dos YY não são alterados pela nova função e desta forma alguns pontos ou até mesmo a totalidade destes pode não ser projetada no gráfico. Para o exemplo acima foi necessário projetar primeiro os valores de Yb e em segundo momento os de Lu, caso contrário a maioria dos valores de Yb cairiam fora do gráfico.

A legenda é colocada numa determinada posição. Essa posição depende das coordenadas do gráfico, no caso $x=70$ e $y=20$. Este valor é configurável e deve ser avaliado para que o gráfico resultante seja o mais legível possível.

Linhas

A linhas num gráfico podem servir para diversos fins, para desenhar grelhas, para separar domínios de um gráfico, para assinalar uma correlação linear, ou mesmo para assinalar a posição da média num histograma.

O exemplo seguinte desenha uma linha correspondente à regressão linear. No título escreve a equação dessa mesma correlação. Para se poder utilizar algumas funções de regressão devemos recorrer à biblioteca `[stats]`. A função `require` ativa essa biblioteca, desde que ela já esteja instalada no vosso sistema.

Neste exemplo é também usada a função `paste0` que permite juntar fragmentos de texto com valores numéricos.

A função `lm` corresponde à criação de um modelo de ajuste linear de um conjunto de dados em função de uma variável dependente, no exemplo abaixo `timor$Yb`.

A função `abline` é utilizada para desenhar a reta definida pela regressão. O resultado de `lm` é complexo e contém a interceção em $x=0$ e o declive.

```
# necessário para fazer os cálculos da regressão linear
require(stats)

# A regressão linear
reg = lm(timor$Yb~timor$La)

# Os coeficientes
coef=coefficients(reg)

# A equação da linha
eq = paste0("y = ",round(coef[1],1)," + ", round(coef[2],3), "*x ")

# O gráfico
plot(timor$La, timor$Yb, main=eq)

# A linha
abline(reg, col="blue")

segments(20, 5, 80, 20, col="red")
```

O resultado é o apresentado na Figura 10.5.

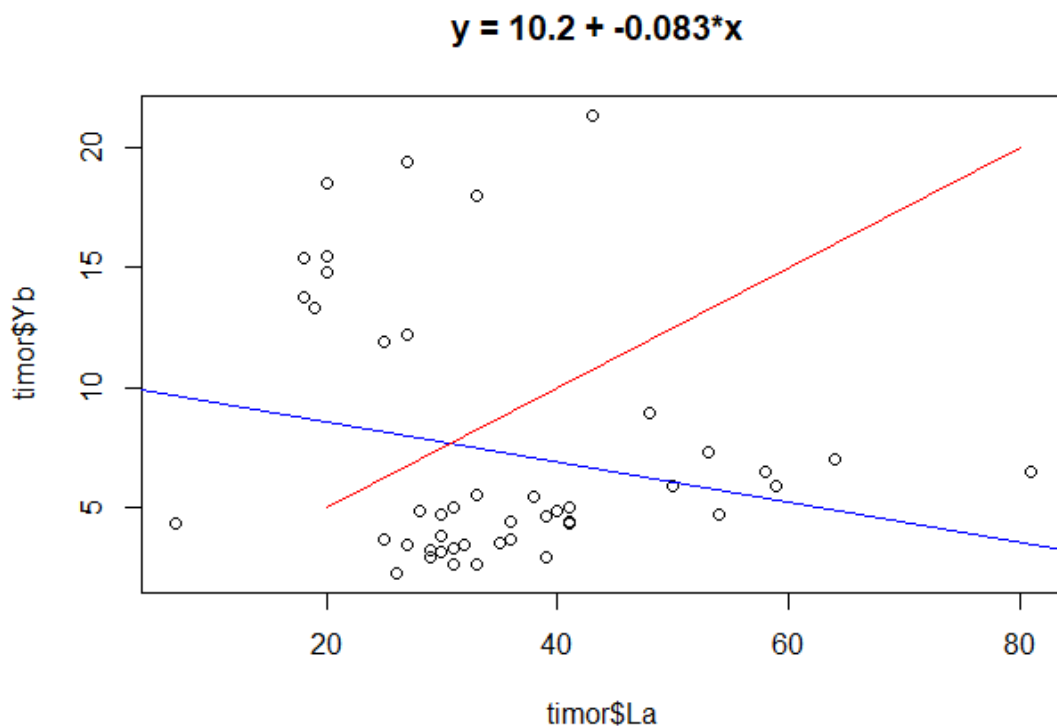


Figura 10.5. Gráfico com a regressão linear entre as variáveis La e Lu. A equação da regressão está no título do gráfico. Foi também desenhado um segmento de reta para ilustrar esta função.

Exercício 10.3.

Este exemplo apresenta algumas funções novas.

Começa por ativar uma biblioteca de tratamento estatístico [stats].

Cria uma função de regressão linear entre duas variáveis [lm].

Determina os coeficientes dessa regressão [coefficients].

Cria um texto complexo, composto por diferentes textos parciais [paste0].

Desenha uma linha com base numa função [abline] que utiliza a interceção na origem e o declive.

Desenha um segmento de reta entre os pontos (20,5) e (80,20).

Para determinar o valor de R^2 [`r.squared`] que fornece uma indicação do erro da regressão linear pode proceder da seguinte forma.

```
# calculo de R2
v1 = summary(reg)
v1$r.squared
```

A função `summary` permite calcular alguns dos parâmetros de uma regressão incluindo o R^2 .

Textos em gráficos

Em alguns casos é necessário adicionar informação extra ao gráfico. Caso essa informação seja sob a forma de texto pode ser utilizada a função `text`.

Esta função, além da localização do texto (ponto central) permite definir os argumentos habituais, como a cor (`col`), o tamanho do texto (`cex`) e a posição do texto (1-inferior, 2- lado esquerdo, 3-topo e 4-lado direito).

O exemplo seguinte ilustra a criação das coordenadas (`v1`, `v2`) de um conjunto de 10 pontos gerados aleatoriamente (`rnorm`) com média de 5 e desvio padrão de 4.

```
# Texto complexo e linhas verticais e horizontais
v1 = rnorm(10,5,4)
v2 = rnorm(10,5,3)
plot(v1,v2)
text(v1, v2,
     paste0("(" , round(v1,0) , ", " , round(v2,0) , ")"),
     cex=.6, pos=3, col="red")
abline(h=5, col="red", lty= "dotted")
abline(v=5, col="red", lty= "dotted")
```

O resultado é o da figura 10.6.

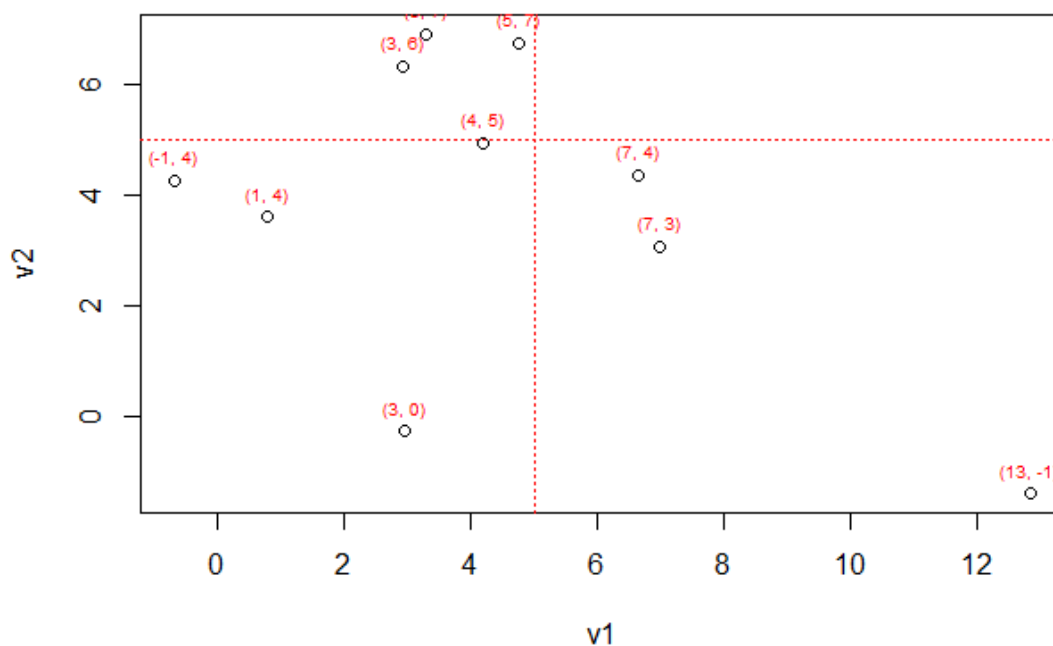


Figura 10.6. Gráfico de dispersão em que os pontos são legendados e são desenhadas linhas horizontais e verticais no valor da média.

Neste exemplo é utilizada a função `rnorm` para criar números aleatórios em torno de um valor médio (10) e de um desvio padrão (5 e 4 respetivamente).

A função `abline` é também aqui utilizada de forma ligeiramente diferente da apresentada anteriormente, em que os argumentos (`v=`) e (`h=`) são utilizados para desenharm linhas horizontais e verticais.

11- COMPOSIÇÃO DE MÚLTIPLOS GRÁFICOS E EXPORTAÇÃO DOS GRÁFICOS

Por vezes é necessário combinar vários gráficos numa imagem por motivos de clareza dos resultados ou para separar informação relevante. O R permite combinar vários gráficos numa única imagem.

Gráficos múltiplos

Para organizar os gráficos o R utiliza a função `par` (significa definição de parâmetros de um gráfico) associada aos parâmetros `mfrow` ou `mfc col`.

O exemplo seguinte projeta dois histogramas de 100 valores aleatórios na mesma imagem.

```
# 1 linhas com dois gráficos
v1 = rnorm(100, 5, 4)
v2 = rnorm(100, 5, 3)
par(mfrow= c(1, 2))
hist(v1, col= "blue")
hist(v2, col= "red")
```

A Figura 11.1 apresenta o resultado obtido.

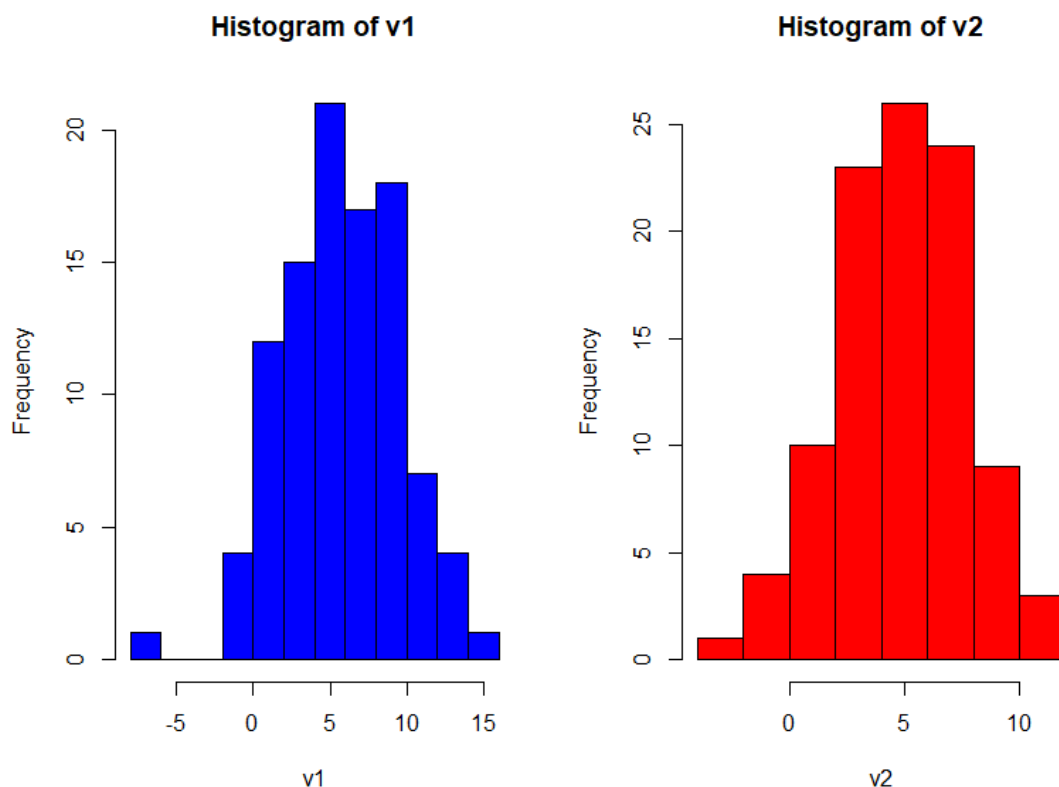


Figura 11. Composição de dois histogramas na mesma imagem.

Pode-se incluir tantas imagens como as que se pretender. Para isso basta modificar os parâmetros referidos anteriormente.

O exemplo seguinte apresenta os mesmos dois histogramas e dois diagramas de bigodes para as referidas variáveis.

```
# 1- linhas para dois gráficos
v1 = rnorm(100,5,4)
v2 = rnorm(100,5,3)
# 2- definição da configuração ( 2 x 2 )
par(mfcol=c(2,2))
# 3- gráficos
hist(v1,col="blue")
boxplot(v1, horizontal=TRUE, col="cyan")
hist(v2,col="red")
boxplot(v2, horizontal=TRUE, col="orange")
```

A figura 11.2 apresenta os resultados obtidos. Note que a seqüência dos gráficos é construída pela ordem coluna, linha ao contrário de quando utilizamos `mfrow`.

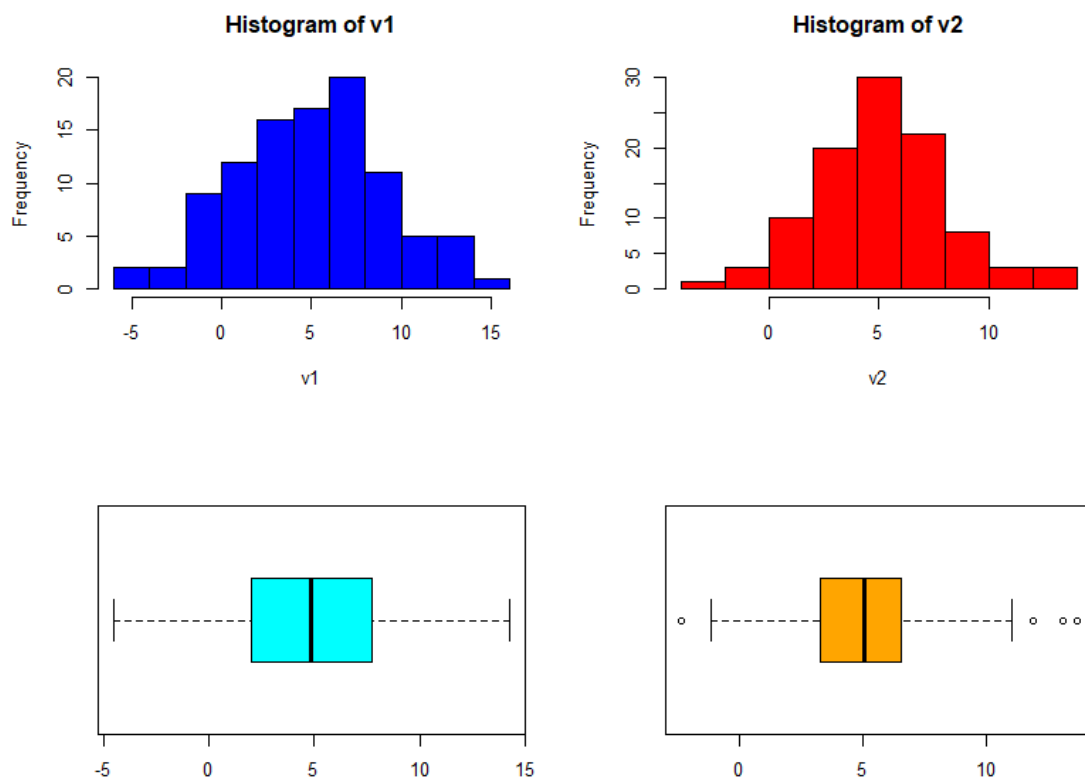


Figura 11.2 Composição de 4 gráficos com variáveis aleatórias.

Exercício 11.1.

No exemplo anterior foi introduzido um novo tipo de gráfico. Os gráficos de bigodes (*whisker plot*, em inglês).

Utilize a ajuda do R para entender os diferentes parâmetros deste tipo de gráfico.

Exportar gráficos

De uma forma simples no RStudio um gráfico pode ser exportado utilizando a respetiva função na subjanela [Plots]. A figura 11.3 ilustra esta opção.

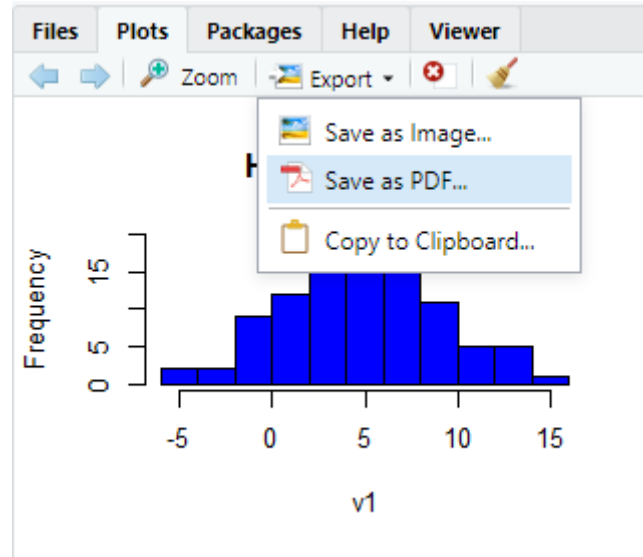


Figura 11.3. O menu [Export] da subjanela [Plots].

Além desta possibilidade, em R, pode utilizar a linha de comandos para criar um gráfico e exportá-lo. Para isso existem funções distintas consoante o formato da imagem que se pretende exportar. Vamos exemplificar com os formatos jpg, png e pdf que são bastantes comuns.

Deve começar por abrir e criar o ficheiro, definindo o nome e caminho se for o caso e as suas dimensões. Naturalmente estas dimensões vão condicionar a forma como a imagem é criada.

Vejamos o exemplo seguinte.

```
# Exportação de imagens
# 1- Abrir o ficheiro jpeg
jpeg("LaLu.jpg", width = 800, height = 600)
# 2- Definir o gráfico
par(mfcol=c(1,2))
boxplot(timor$La, col="red", main="La (ppm)")
boxplot(timor$Lu, col="blue", main="Lu (ppm)")
# 3- Gravar o ficheiro
dev.off()
```

A função `dev.off()` serve para indicar ao programa que o gráfico está terminado e pode gravar. Todas as operações feitas posteriormente a esta instrução não se refletem no ficheiro gravado.

O resultado do exemplo acima encontra-se na Figura 11.4.

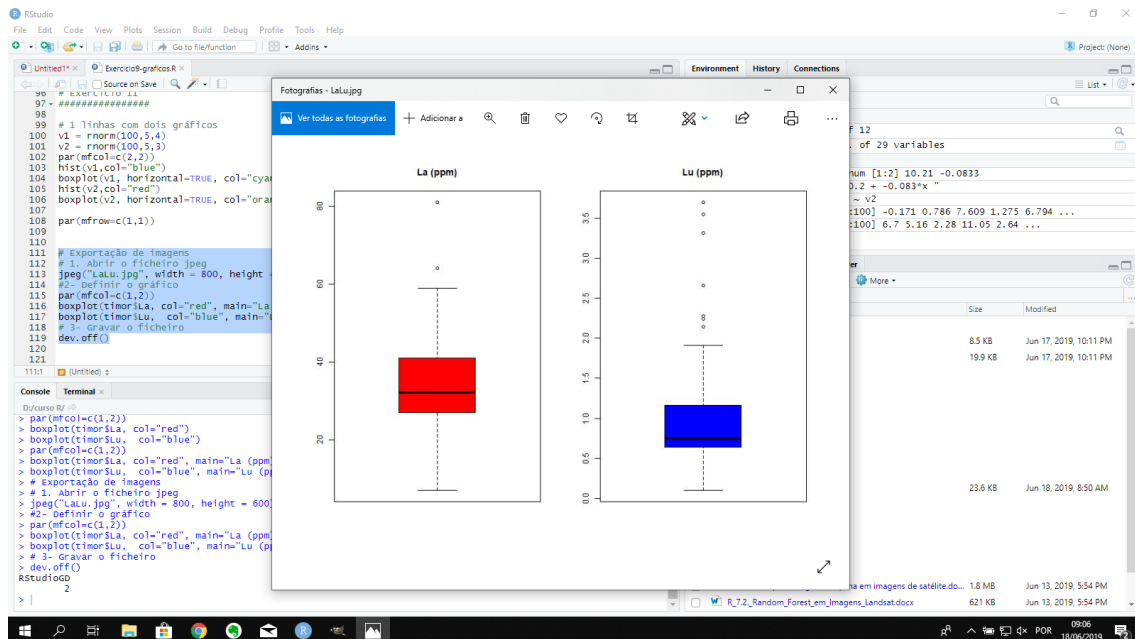


Figura 11.4. Imagem jpg criada em R.

Além desta função podem ser usadas as funções:

- pdf("rplot.pdf"): para ficheiro pdf
- png("rplot.png"): para ficheiro png
- jpeg("rplot.jpeg"): para ficheiro jpeg
- postscript("rplot.ps"): para ficheiro postscript
- bmp("rplot.bmp"): para ficheiro bmp
- win.metafile("rplot.wmf"): para ficheiro windows metafile

Em todos os casos deve sempre terminar com a função `dev.off()`.

12- GRÁFICOS AVANÇADOS: O GGPLOT

Muito embora as funções de gráficos existentes por omissão em R sejam já por si bastante boas e produzam resultados de grande qualidade, existe um módulo específico que permite criar gráficos com carácter profissional e com grande flexibilidade na sua parametrização.

A filosofia dos gráficos com `ggplot` é um pouco diferente da que é utilizada nos gráficos normais.

O primeiro passo trata de criar uma variável que vai ser o gráfico. A função `ggplot` permite definir os parâmetros iniciais do gráfico. No exemplo é criada a variável `g1` que vai conter o gráfico; é utilizado o argumento `data` para definir a *data frame* que contem os dados e o argumento `aes()` para definir a estética desse mesmo gráfico, no caso os valores de `x` e de `y`. Na segunda linha é apresentado o gráfico (`g1`) ao qual se acrescentam os pontos com recurso à função `geom_point()`.

```
g1 = ggplot(data= timor, aes(x= La, y= Lu))  
g1 + geom_point()
```

O resultado deste exemplo está ilustrado na Figura 12.1.

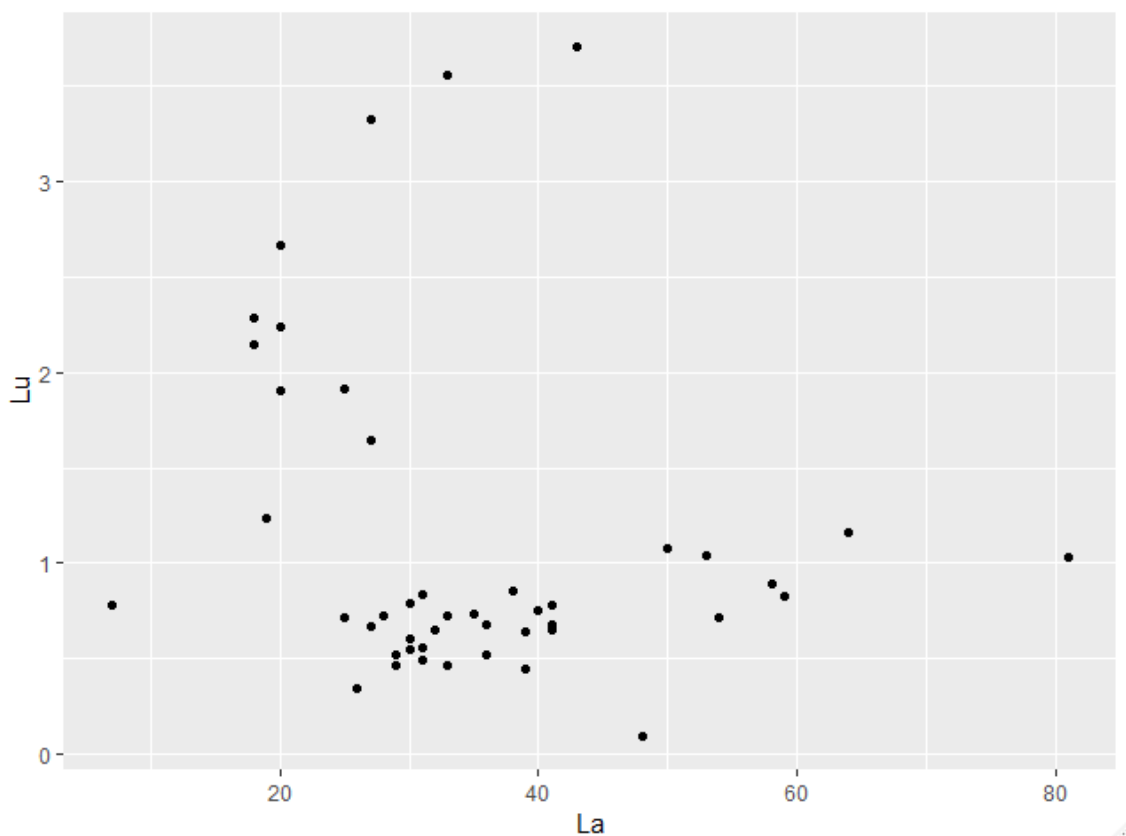


Figura 12.1. Gráfico de dispersão La vs Lu criado em ggplot.

A estética de um gráfico corresponde a tudo aquilo que pode ser visível no gráfico. Essa estética inclui os seguintes parâmetros:

- posição (i.e., nos eixos x e y)
- color (cor exterior)
- fill (cor do interior)
- shape (forma dos pontos)
- linetype (tipo de linha)
- size (tamanho dos elementos)

Geometrias

Os dados a projetar num gráfico devem possuir uma geometria. Essa geometria pode ser de pontos (ver exemplo anterior) ou de um qualquer outro tipo de representação que o ggplot possua. Todas as funções de geometria em ggplot começam com `geom_` e devem ser acrescentadas ao gráfico criado.

Nas pastas do curso existe um documento pdf intitulado [ggplot2- cheatsheet] que contém uma síntese das diferentes possibilidades do ggplot. A Figura 12.2 apresenta um pequeno resumo.

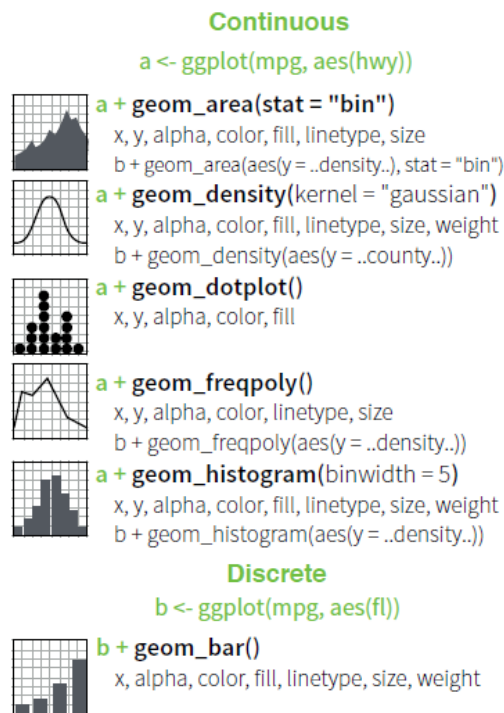


Figura 12.2. Resumo das geometrias em ggplot para uma única variável.

Assim, o exemplo anterior pode ser adaptado para criar um histograma com os valores de `La`, com a função `geom_histogram()`.

```
g1 = ggplot(data= timor, aes(x= La))
g1 + geom_histogram()
```

O resultado é o da figura 12.3.

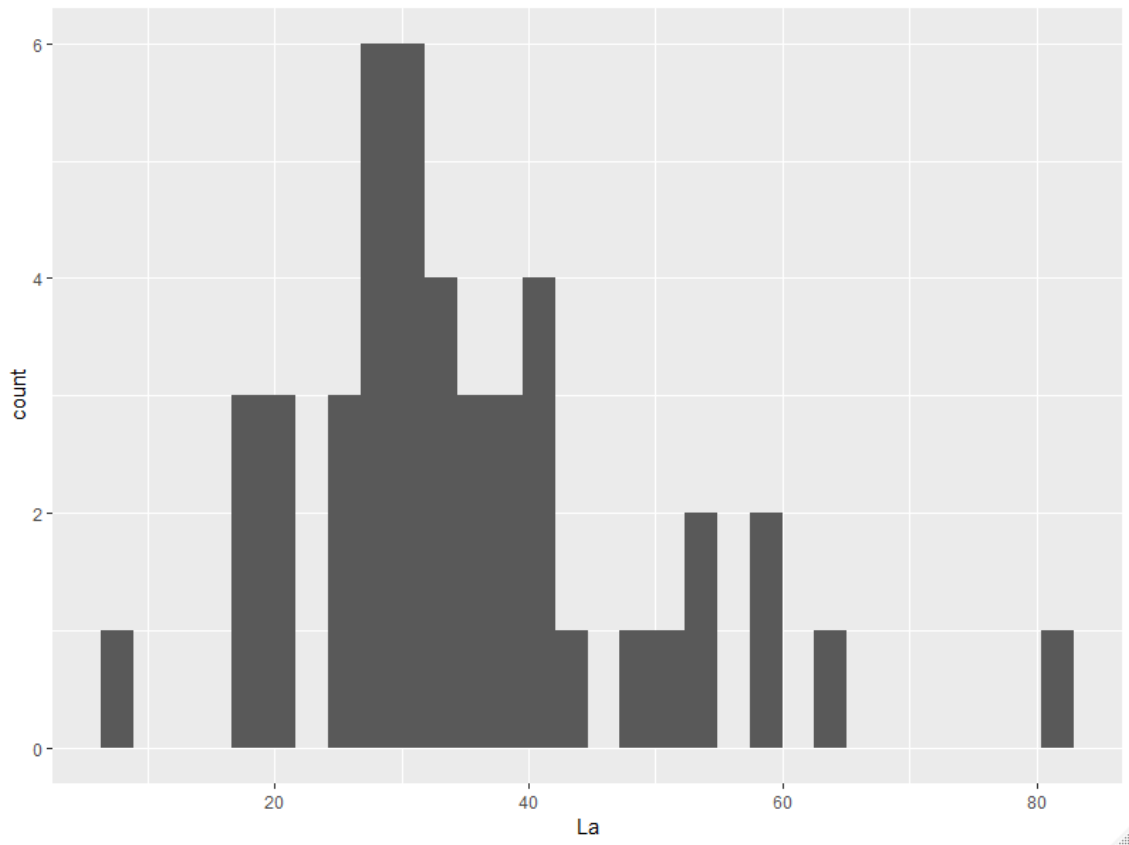


Figura 12.3. Gráfico com o histograma para o La criado em ggplot.

Para os exemplos seguinte vai ser utilizado o conjunto de análises geoquímicas de sedimentos de linha de água da região de Manica, Moçambique, colhidos em três zonas diferentes.

O exemplo seguinte ilustra como se produz o histograma dos valores de La separado por diferentes zonas. Note que a variável Zona é uma coluna dos nossos dados.

O código em R é o seguinte:

```
# 1- ler os dados de Manica
manica= read.csv2("dados/exercicio12-1.csv", header=TRUE)
#2- criar o gráfico em que as cores vão depender da variável Zona
g2 = ggplot(data= manica, aes(x=La, fill=Zona))
# 3- projetar o histograma
g2 + geom_histogram(position= "dodge", binwidth= 20)
```

O resultado é apresentado na Figura 12.4.

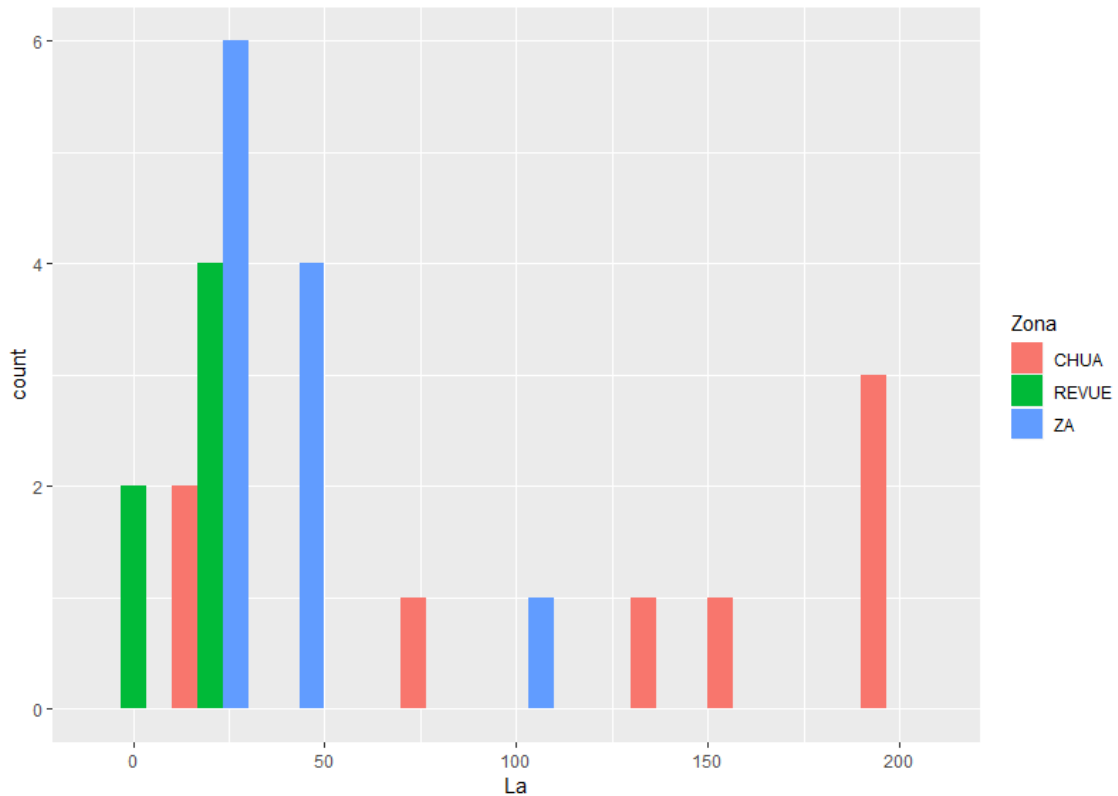


Figura 12.4. Histograma dos valores de La na região de Manica agrupados por Zonas.

Estes resultados permitem rapidamente ver que o La apresenta comportamentos diferentes consoante cada uma das zonas amostradas.

Agrupamento de gráficos em ggplot

Os dados podem ser agrupados de diversas formas. Uma forma simples é utilizar as cores para distinguir os diferentes tipos de dados. Vejamos o exemplo de um gráfico La vs Lu por zonas em ggplot.

```
# 2- criar o gráfico em que as cores vão depender da variável Zona
g2 = ggplot(data= manica, aes(color= Zona))
# 3- projetar o histograma
g2 + geom_point(aes(x= La, y= Lu))
```

Note os seguintes aspetos o gráfico é criado sem na estética inicial ser indicado qual o valor de x e y, apenas é indicado para agrupar por cores as diferentes zonas. No momento de implementar a geometria é indicado quais os valores a projetar (x e y).

O resultado é o gráfico da figura 12.5.

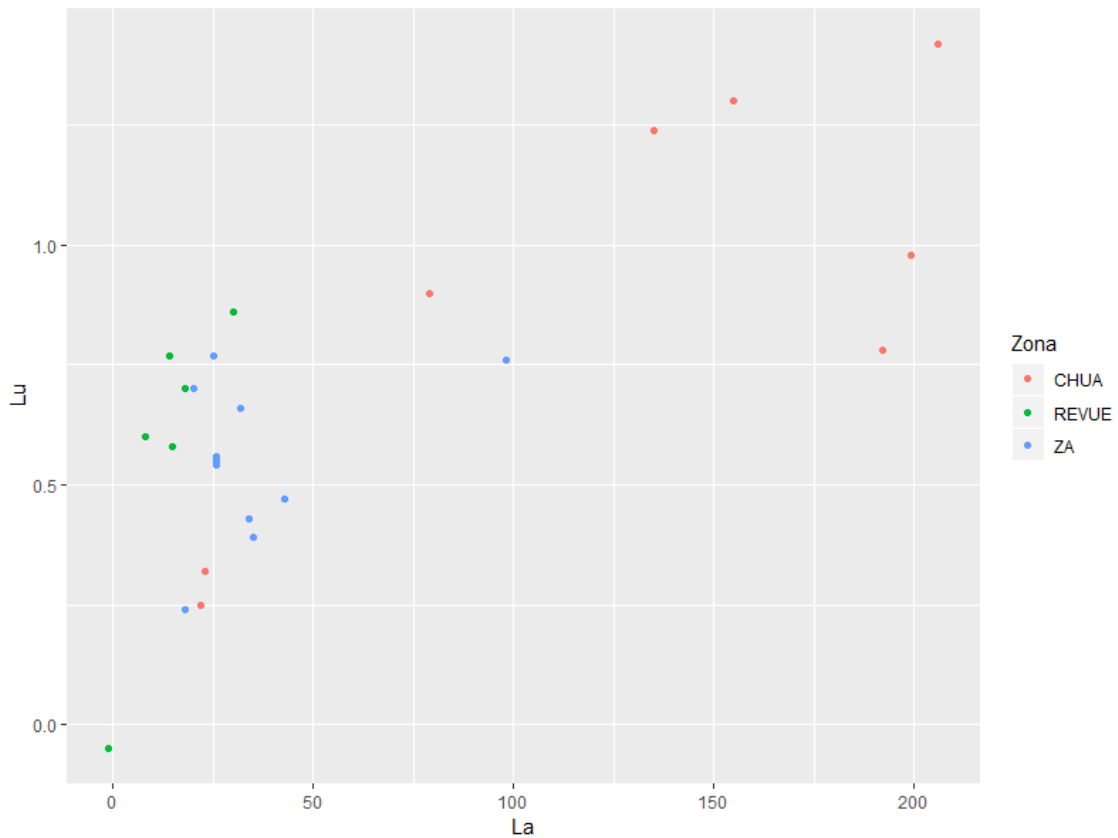


Figura 12.5. Gráfico de dispersão com as cores a indicar as diferentes zonas de colheita de amostras.

Quando a quantidade de dados for demasiado grande ou se justifique por algum motivo pode-se separar os gráficos baseados numa variável. O exemplo abaixo ilustra esta situação com o recurso à função `facet_grid()`.

```
# 4- criação de gráficos separados
g2 + geom_point(aes(x= La, y= Lu)) + facet_grid(~Zona)
```

A figura 12.6. apresenta o resultado da função `facet_grid` aplicada aos dados do exemplo.

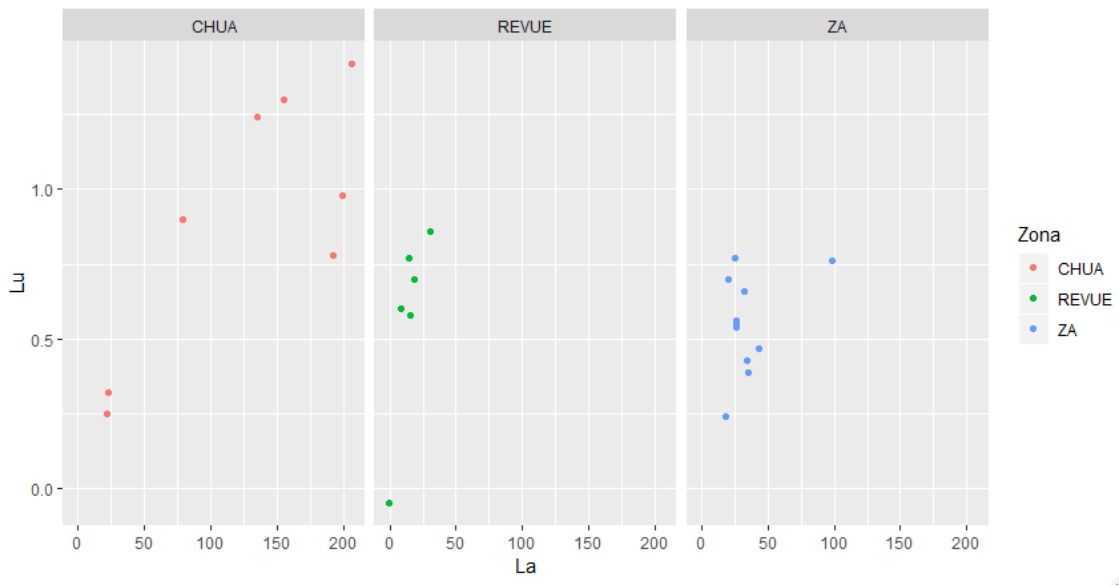


Figura 12.6. Gráfico separado com `facet_grid`.

Quando o número de gráficos é demasiado grande pode utilizar a função `facet_wrap` em substituição da `facet_grid`.

Diagramas normalizados

Em alguns trabalhos de geoquímica é necessário normalizar um conjunto de valores com base num padrão. Essa normalização permite a comparação com o padrão e avaliar as diferenças com esse padrão.

Um exemplo desse tipo de abordagem são os diagramas de terras raras que permitem avaliar o padrão das terras raras (leves e pesadas) e como estas se comportam em relação ao valor padrão, que muitas vezes é o condrito ou um valor de MORB- *mid ocean ridge basalt*.

Esta abordagem necessita de um tratamento prévio dos dados e a determinação do padrão.

Neste exemplo as nossas amostras são de sedimentos de linha de água, o que implica que o padrão a utilizar deve ser uma rocha deste tipo, no caso o NAS- *North American Shale*. O primeiro aspeto é a criação de um conjunto de variáveis com os padrões referidos.

```
#####
# NORMALIZAÇÃO
#####
# 1- criação dos valores de NAS
elementos=c("La", "Ce", "Nd", "Sm", "Eu", "Tb", "Yb", "Lu")
NAS = c(31.1, 66.7, 27.4, 5.59, 1.18, .85, 3.06, .456)
# 2- criação de data frame com os valores das amostras
Amostras = data.frame(manica$La, manica$Ce, manica$Nd, manica$Sm, manica$Eu,
manica$Tb, manica$Yb, manica$Lu)
```



```
# 3- normalização dos valores das amostras
normalizado = amostras/NAS
# 4- criação da data frame final
# Ajusta-se a coluna com as classes que vão ser trabalhadas
reeManica = cbind(normalizado, Zona= manica$Zona)
colnames(reeManica) = c(elementos, "Zona")
```

Para se poder criar o gráfico de terras raras é necessário converter os dados para um formato estendido, e para tal utiliza-se a função `melt` da biblioteca [reshape2].

```
# Diagramas do terras raras
library(reshape2)
# Conversão de dados para o gráfico
reeM = melt(reeManica)
```

O resultado da transformação é o seguinte:

```
> head(reeManica)
      La      Ce      Nd      Sm      Eu      Tb      Yb      Lu Zona
1  0.5787781  1.019490 -0.3649635  0.3577818 -0.169491525 -2.35294118  0.68627451  0.526315789  ZA
2  0.3898051  2.956204 -1.7889088  1.5254237 -0.235294118 -0.65359477  5.70175439  0.018006431  ZA
3  1.2773723  17.352415 -8.4745763  3.0588235  0.294117647 -4.38596491  0.09324759  0.005847076  ZA
4  7.6923077  150.000000 -11.7647059  1.2418301 -0.438596491 -0.06430868  0.04347826  0.017153285  ZA
5  22.0338983  103.529412 -3.2679739  4.3859649  0.028938907 -0.02998501  0.08029197  0.096601073  ZA
6  23.5294118  26.143791  100.8771930  0.1028939 -0.002998501 -0.07299270  0.60822898  0.593220339  ZA

> head(reeM)
  Zona variable      value
1   ZA      La  0.5787781
2   ZA      La  0.3898051
3   ZA      La  1.2773723
4   ZA      La  7.6923077
5   ZA      La 22.0338983
6   ZA      La 23.5294118
```

Nota: A função `head` mostra apenas alguns elementos de uma variável longa.

Pode também em alternativa utilizar a função `gather` já referida atrás, com o exemplo seguinte.

```
reeM2 = gather(reeManica, key="Elemento", value="Valor", -Zona)

> head(reeM2)
  Zona Elemento      Valor
1     ZA      La  0.578778135
```

2	ZA	La	0.389805097
3	ZA	La	1.277372263
4	ZA	La	7.692307692

Com os dados neste formato o ggplot permite que os agrupemos por cores e os dividamos por zonas. O código seguinte ilustra o pretendido.

```
g3 = ggplot(data= reeM, aes(x= variable, y=log(value), group=Zona, color=Zona))
g3 + geom_point(show.legend= FALSE) + geom_smooth() + facet_grid(~Zona)
```

A função `geom_smooth()` permite criar linhas e zonas de tendência num gráfico. O resultado é o ilustrado na figura 12.7. Note que para o eixo dos y as variáveis são projetadas em valores logarítmicos para facilitar a leitura.

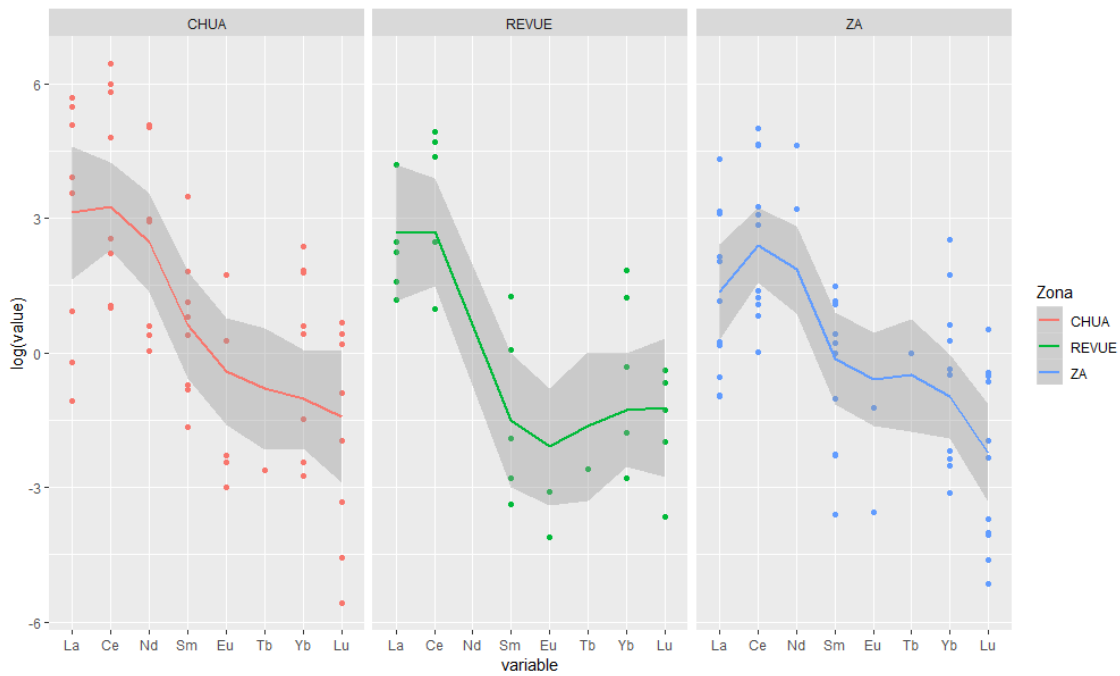


Figura 12.7. Gráficos de terras raras por zona para a região de Manica.

Diagramas triangulares

Para a realização de diagramas triangulares é necessário utilizar um *package* especial designado de [Ternary] que permite desenhar este tipo de gráficos.

Para tal comece por instalar este módulo e de seguida ative as suas bibliotecas.

```
install.packages("Ternary")
library(Ternary)
```

Os dados devem estar organizados em três colunas que permitem classificar os três vértices. Não têm de estar normalizados para 100, uma vez que o programa trata de fazer essa normalização.

O exemplo abaixo mostra a organização dos dados. Para este exemplo vai ser criado um diagrama ternário de La-Th-Sc.

```
# 1- organizar os dados
La = manica$La
Th = manica$Th
Sc = manica$Sc

# 2- criar a data frame com as três variáveis
dataManica = cbind(La, Th, Sc )
```

De seguida há que criar o gráfico e adicionar os pontos ao gráfico, como no exemplo abaixo.

```
# 3- criar o gráfico
TernaryPlot(point="up", atip= 'La', btip= 'Th', ctip= 'Sc')

# 4- acrescentar os pontos
AddToTernary(points, dataManica, pch= 15, col= "blue")

# 5- exemplo para etiquetar os pontos
TernaryText(dataManica, col= "red", font= 2)
```

No exemplo foi ainda adicionado a identificação de cada ponto com a função `TernaryText()`.

A Figura 12.8 ilustra o resultado obtido.

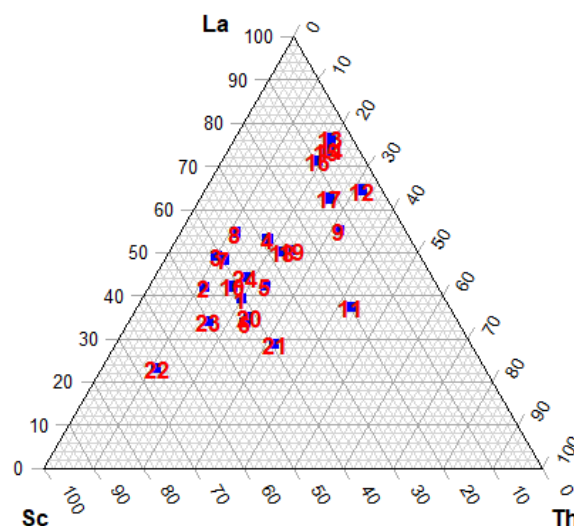


Figura 12.8. Diagrama La-Th-Sc para as amostras de [manica].

Também é possível incluir as linhas de contorno de isovalores dentro do gráfico ou um gradiente de cores consoante a densidade do gráfico, como ilustram os dois exemplos abaixo.

```
TernaryPlot(point= "up", atip= 'La', btip= 'Th', ctip= 'Sc')
TernaryPoints(dataManica, col= 'red', pch= 15, cex= .5)
```

```
TernaryDensityContour(dataManica, resolution= 10L, col= "blue")
```

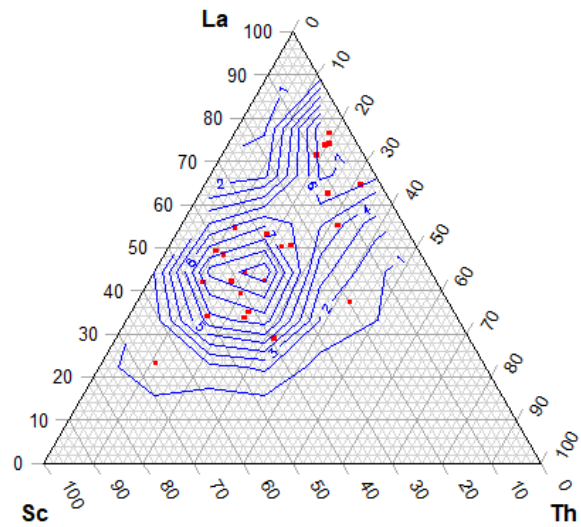


Figura 12.9. Diagrama La-Th-Sc com contornos.

```
TernaryPlot(point= "up", atip= 'La', btip= 'Th', ctip= 'Sc')  
ColourTernary(TernaryDensity(dataManica, resolution= 10L))  
TernaryPoints(dataManica, col= "red", pch= 15, cex= .5)
```

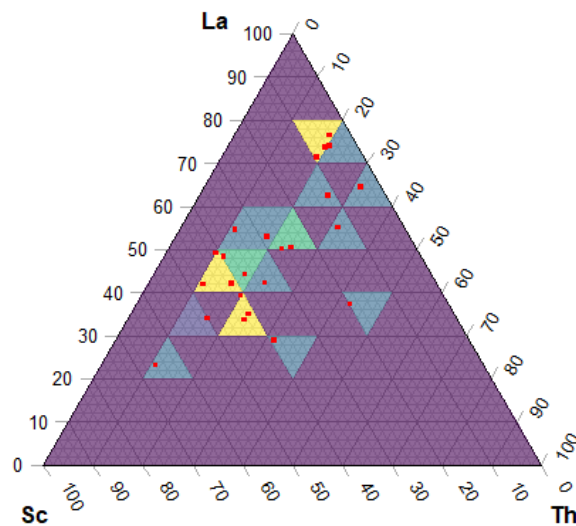


Figura 12.10. O mesmo diagrama com gradiente de cores a ilustrar a densidade de pontos.

Como dominar o ggplot?

Com o pacote (ggplot2) as opções de formatação de gráficos são inúmeras, de seguida são mostrados alguns exemplos de como dominar os seus dados e dominar as principais potencialidades deste pacote. Desde a inserção de dados à sua exportação.

Funções

Tal como já mencionado o `ggplot` permite o uso de funções específicas tendo em conta o tipo de gráfico que queremos construir, sejam eles histogramas, linhas, pontos etc.

Utilizemos para os seguintes exercícios o documento [exercício12-2.csv], com o intuito de realizarmos um gráfico de pontos.

```
#1 - Abrir o ficheiro - Vamos chamá-lo de "Temp"
Temp = read.csv2(file="exercício12-2.csv", header=TRUE, dec=",")

#2 - criar o gráfico
a = ggplot(ThvsSal, aes(x=ThvsSal$Th, y=ThvsSal$NaCl, group=ThvsSal$Type))
a
```

Q: Porque é que não vemos os dados no gráfico?

```
#3 - Adicionar função referente ao gráfico que queremos fazer, neste caso pontos.
a1 = a + geom_point(aes(shape=Type, color=Type))
```

Q: Acha o gráfico apelativo e de fácil interpretação?

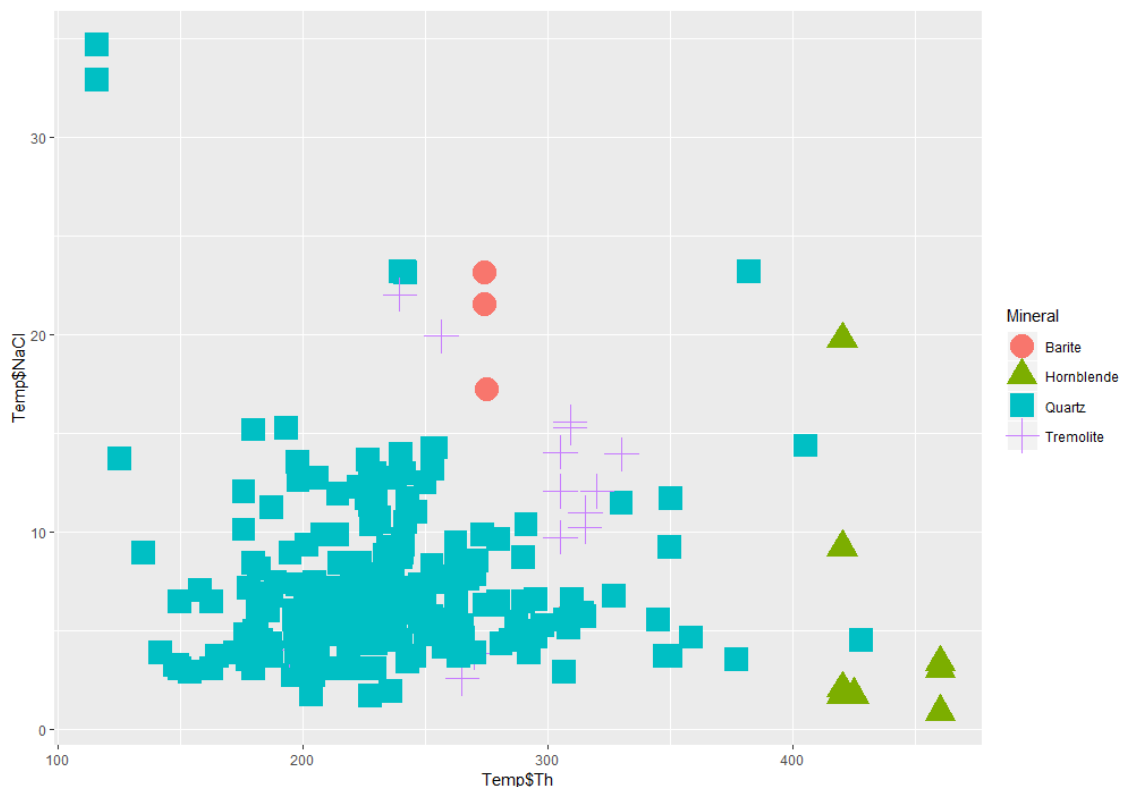


Figura 12.11. Projção do gráfico contruído na função #3

Legendas e simbologias

A legenda dos eixos pode ser colocada automaticamente tendo em conta a legenda das colunas da folha de dados (Excel). No entanto, é possível modificar a legenda utilizando para isso a função `xlab(expression(paste("")))` e `ylab(expression(paste("")))`.

A função `scale_shape_manual()` permite definir quais os símbolos que o gráfico irá possuir.

```
#4 - Adicionar legendas no eixo dos x e no eixo dos y
a2 = a1 + xlab(expression(paste("T" [h], "(°C)"))) +
ylab(expression(paste("Salinity (wt. % NaCl Equiv.)")))
#5 - Alterar a simbologia
a3= a2 + scale_shape_manual(values=c(15, 18, 16, 17 ))
```

Para isto teremos de adicionar à função `geom_point()` o parâmetro `size`.

```
a1 = a + geom_point (aes(shape=Mineral, color=Mineral), size=7)
```

De seguida corra o código todo novamente. Os símbolos estão demasiado grandes? Reduza o número do `size`. No exemplo é utilizada a função `scale_color_manual()` que à semelhança da `scale_shape_manual()` permite definir as cores a serem apresentadas para cada símbolo.

```
#7 - Alterar a cor dos simbolos
a4 = a3 + scale_color_manual(values=c("goldenrod2", "darkorange2", "royalblue",
"darkgreen"))
```

Utilize a cábula de códigos de cor do R apresentada nos Anexos.

Podemos inserir anotações no nosso gráfico, como por exemplo número de amostras/análises. Para isso teremos de utilizar a função `annotate()`.

```
#8 - Anotações no gráfico
a5 = a4 + annotate("text", x=275, y=35, label= "n=265", size = 10, fontface=
"italic")
a5
```

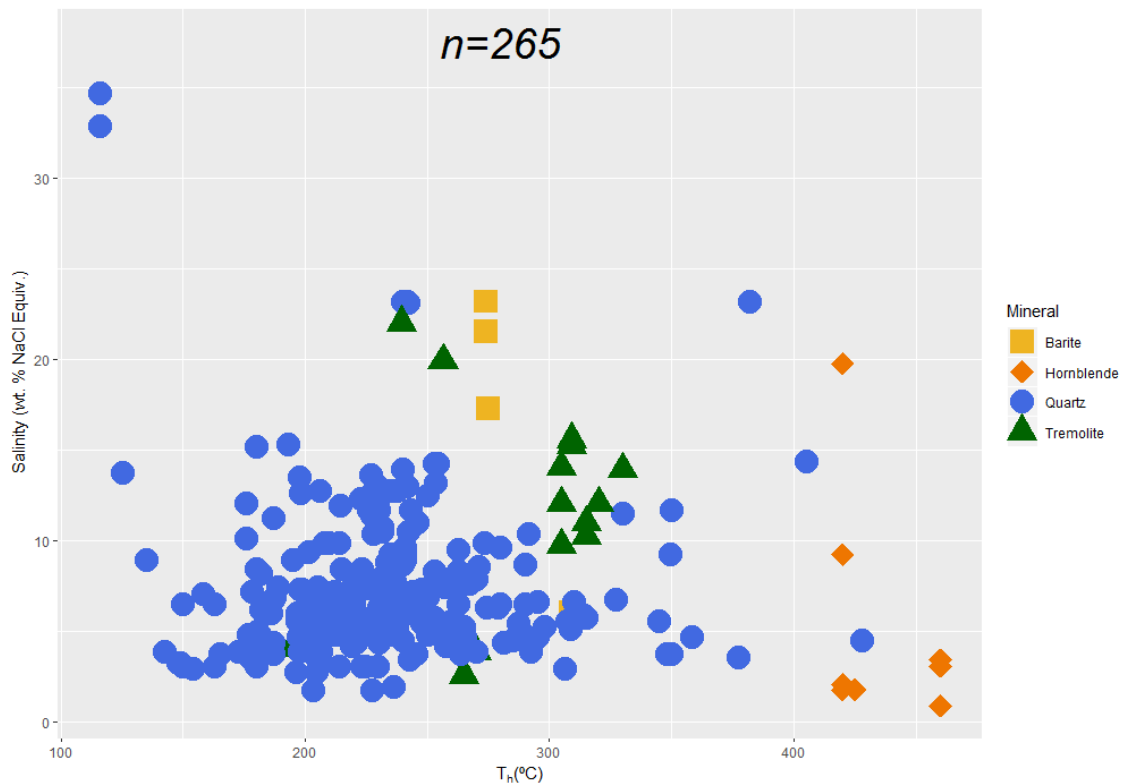


Figura 12.12. Projeção do gráfico contruído na função #8

Eixos (Intervalos)

Quando é feito o *plot* de dados em R este automaticamente atribui uma legenda de eixos ao gráfico tendo em conta os valores máximos e mínimos dos dados.

Este automatismo nem sempre apresenta a gama de valores que pretendemos, e por isto é possível manipular os eixos x e y para apresentar os valores e espaçamentos que pretendemos.

O *ggplot* permite utilizar funções de valores contínuos, discretos ou manual:

```
scale_*_continuous
scale_*_discrete
scale_*_manual
```

Estas funções irão permitir escolher o valor máximo, mínimo e intervalos de valores para os eixos do gráfico.

- Apliquemos então aos nossos dados usando a função `scale_x_continuous`

Q: Analise os dados (usando a função `view()`, `min(data$data)`, `max(data$data)`) e escolha a gama de valores mais adequado substituindo os *

```
#9 - Ajustar escala dos eixos x e y
a6 = a5 + scale_x_continuous(limits=c(*, *), breaks=seq(*, *, *)) +
scale_y_continuous(limits=c(*, *), breaks=seq(*, *, *))
a6
```

Nota: Para garantirmos uma correta interseção dos eixos no valor que pretendemos, podemos utilizar a função `expand_limits(x=*, y=*)`

```
#10 - Introduzir ponto de origem (abscissa e ordenada)
a6 + expand_limits(x=100, y=0)
```

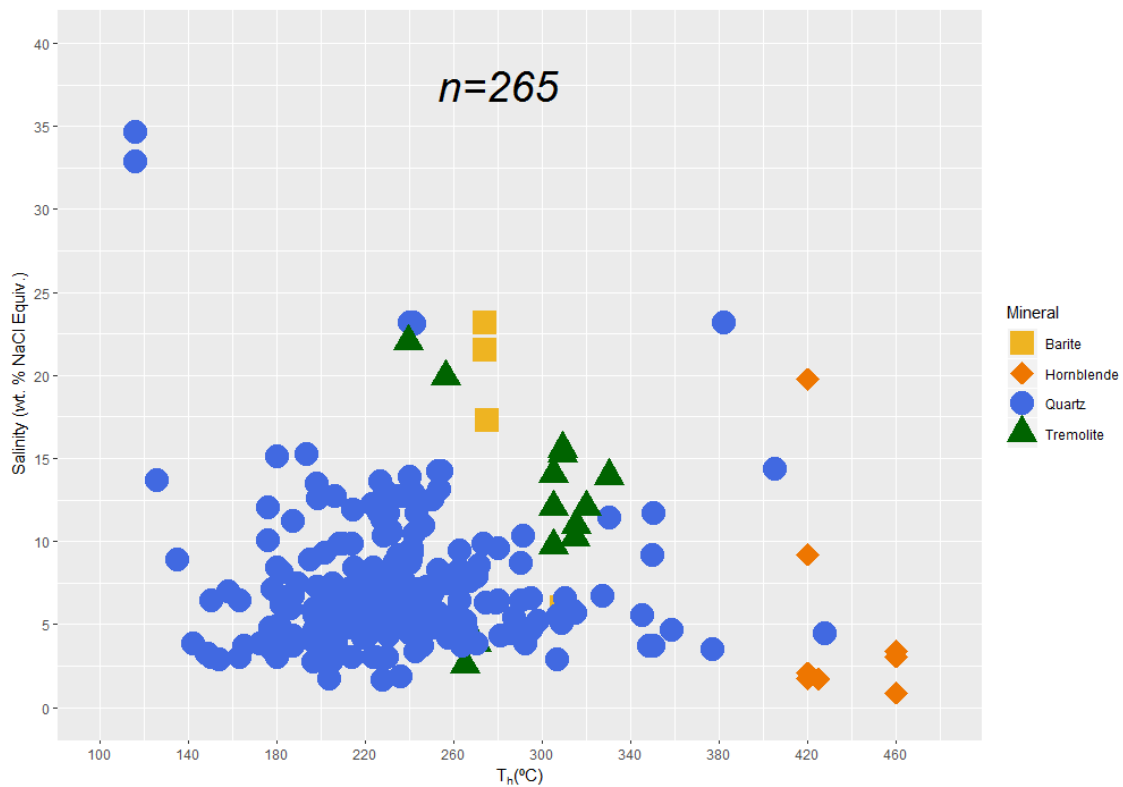


Figura 12.13. Projção do gráfico contruído na função #10

Layout

Apesar de a descrição e tratamento dos dados ser a componente mais importante de qualquer análise de dados geoquímicos, é também importante olhar para a estética do gráfico. Avaliando se é de simples compreensão, se as cores são as corretas, se com base no gráfico podemos acompanhar o texto explicativo dos dados.

Por isto importa utilizar funções de layout de gráficos para melhorar a aparência dos mesmos.

De seguida é apresentado um exemplo de layout que poderão aplicar ao gráfico em construção, no entanto existem várias opções (ver sites: [Stackoverflow](#) e [sthda](#)).

```
#11 - Mudar tamanho e posição da legenda
a7 = a6 + theme(legend.title = element_text(colour="black", size=15,
face="bold")) +
theme(legend.text = element_text(colour="black", size=15, face="bold"))+
theme(legend.position="bottom")
a7

#12 - Mudar tamanhos eixo, linhas, cor de fundo
a8 = a7 + theme(axis.text = element_text(colour = "black", size=20),axis.title.x
= element_text(colour = "black", size=25),axis.title.y = element_text(colour =
```



```
"black", size=25), panel.background =
element_rect(fill="grey100"), panel.grid.minor.y
element_line(size=3), panel.grid.major = element_line(colour = "gray33"),
plot.background = element_rect(fill="white"))
a8
```

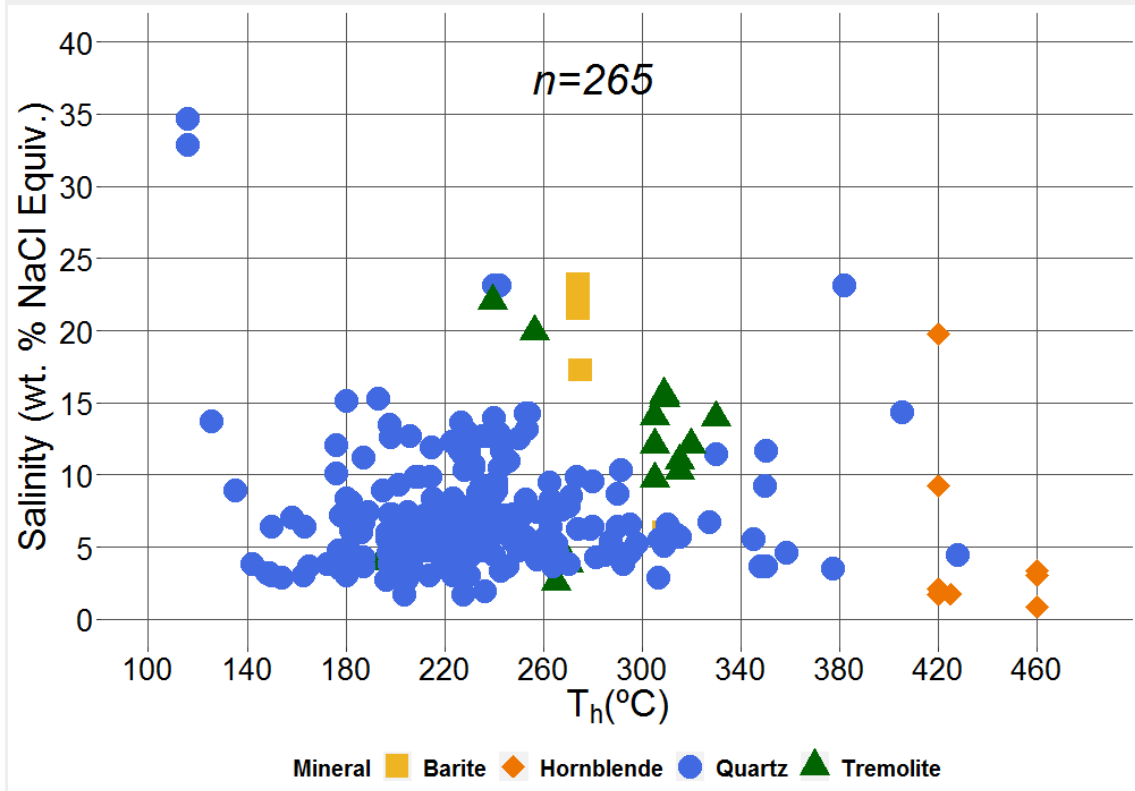


Figura 12.14. Projeção do gráfico contruído na função #12

Gráficos Mistos (Pontos + histogramas)

O ggplot permite a construção de vários gráficos em simultâneo, como é o caso seguinte em que podemos adicionar histogramas ao nosso gráfico de pontos.

Para isto é necessária a instalação de um novo pacote, o `ggExtra`.

```
#13 - Instalação do ggExtra
install.packages("ggExtra")
library(ggExtra)

#14 - Adicionar Histogramas ao gráfico de pontos
a9 = ggExtra::ggMarginal(a8, type = "histogram", color="gray1", fill="gray50")
a9
```

Nota: A projeção de dados, especialmente na ciência, deve ser clara e intuitiva. Em suma, quanto mais simples melhor. Por isto devemos ter atenção para não sobrecarregarmos os nossos dados com informação desnecessária.

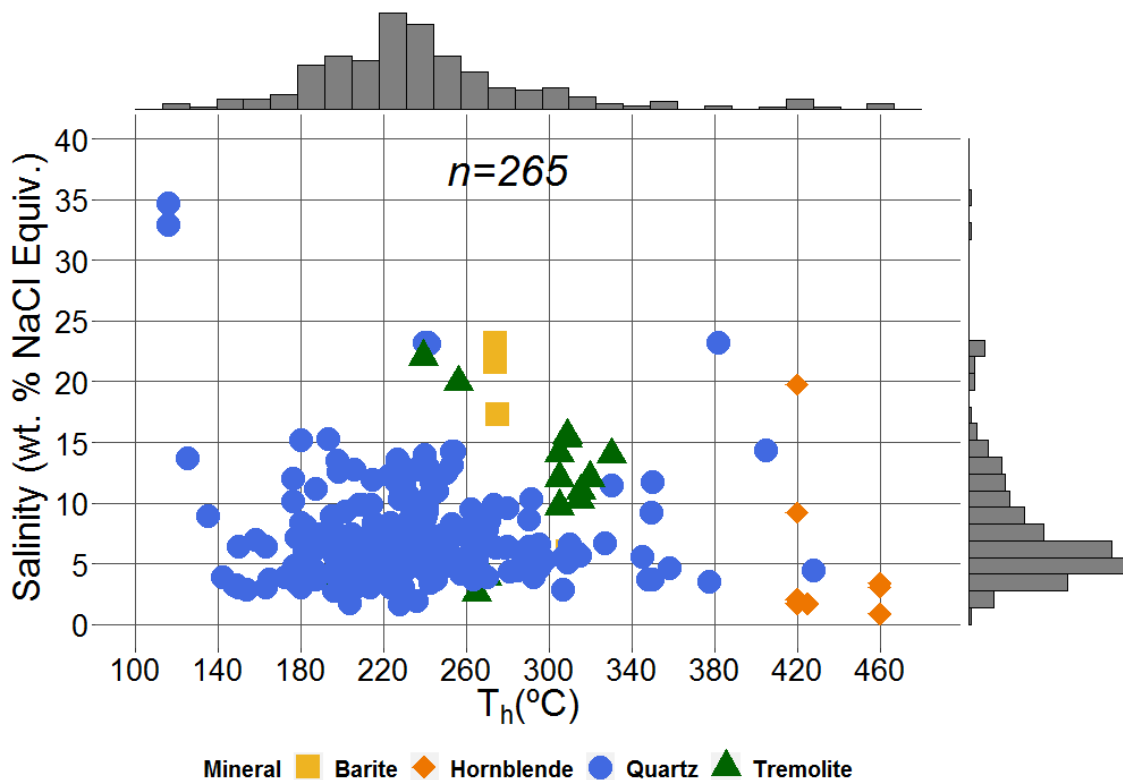


Figura 12.15. Projeção do gráfico contruído na função #14

Retas de maior declive e cálculo de R^2

A utilização de retas de maior declive e o cálculo de R^2 é um tratamento estatístico que pode ser bastante útil em avaliar a correlação dos nossos dados relativamente a uma variável.

Para este ponto iremos utilizar um *script* de código de R para, primeiro, calcularmos a reta de maior declive e de seguida o valor de R^2 . Utilizando para isso os dados e gráfico do exercício que temos vindo a desenvolver nos últimos pontos.

```
#15 - Reta de Maior Declive
X=ggplot(Temp)
require(stats)
reg=lm(NaCl ~ Th, data = Temp)
reg
```

Nota: A função `lm()` calcula a reta de maior declive, fornecendo os valores da interseção da reta e do declive, estes valores deverão ser colocados na função `geom_abline` em `intercept`, `slope`.

```
coeff=coefficients(reg)
eq = paste0("y = ", round(coeff[2],1), "*x + ", round(coeff[1],1))
a10 = a8 + geom_abline(intercept = 7.7970727, slope = -0.0003278, size=2)
a10

#16 - R2
eq = substitute(italic(NaCl) == a + b * italic(Th), "~italic(r)^2~"=~r2,
list(a = format(coef(reg)[1], digits = 2), b = format(coef(reg)[2], digits = 2),
r2 = format(summary(reg)$r.squared, digits = 3))
a11 = a8 + geom_text(x = 100, y = 55, label=Temp$Type.I)+ ggtitle(eq)
a11
```

O cálculo da reta de maior declive e R2 remove os histogramas do gráfico, pelo que se quisermos mantê-los anexados deveremos repetir novamente a expressão do ponto #14.

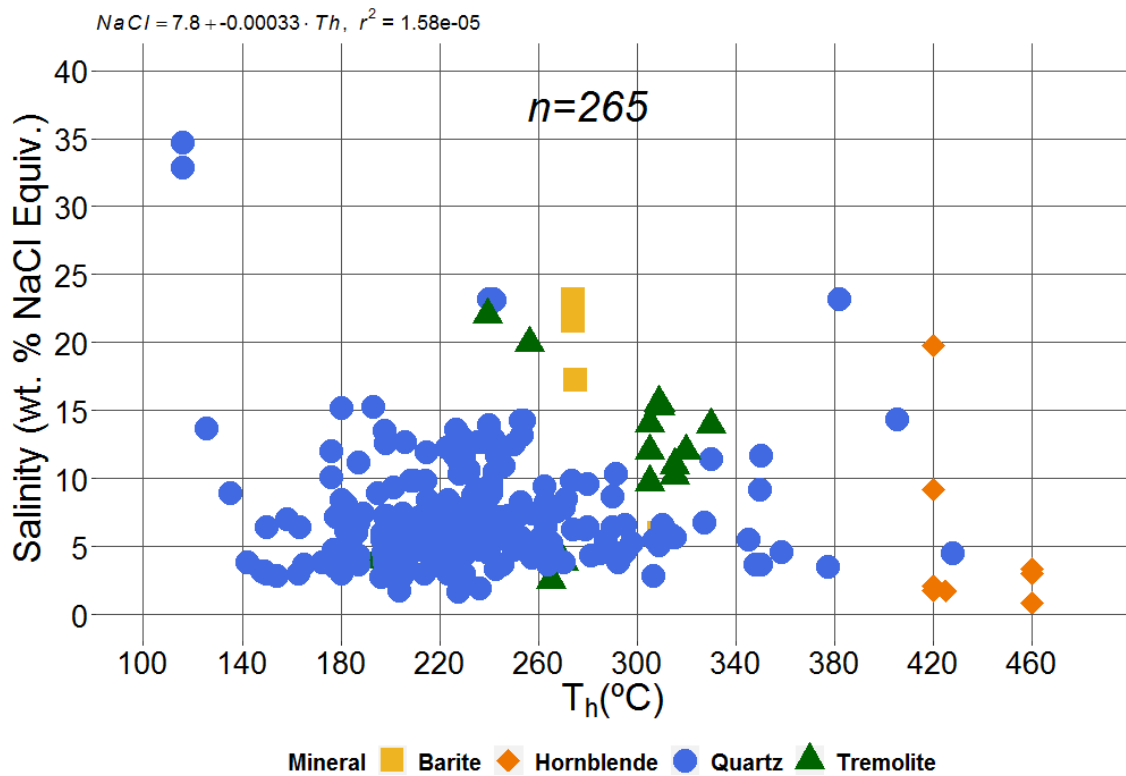


Figura 12.16. Projeção do gráfico contruído na função #16

```
#17 - Reanexar Histogramas
```

```
a12 = ggExtra::ggMarginal(a11, type = "histogram", color="gray1", fill="gray50")  
a12
```

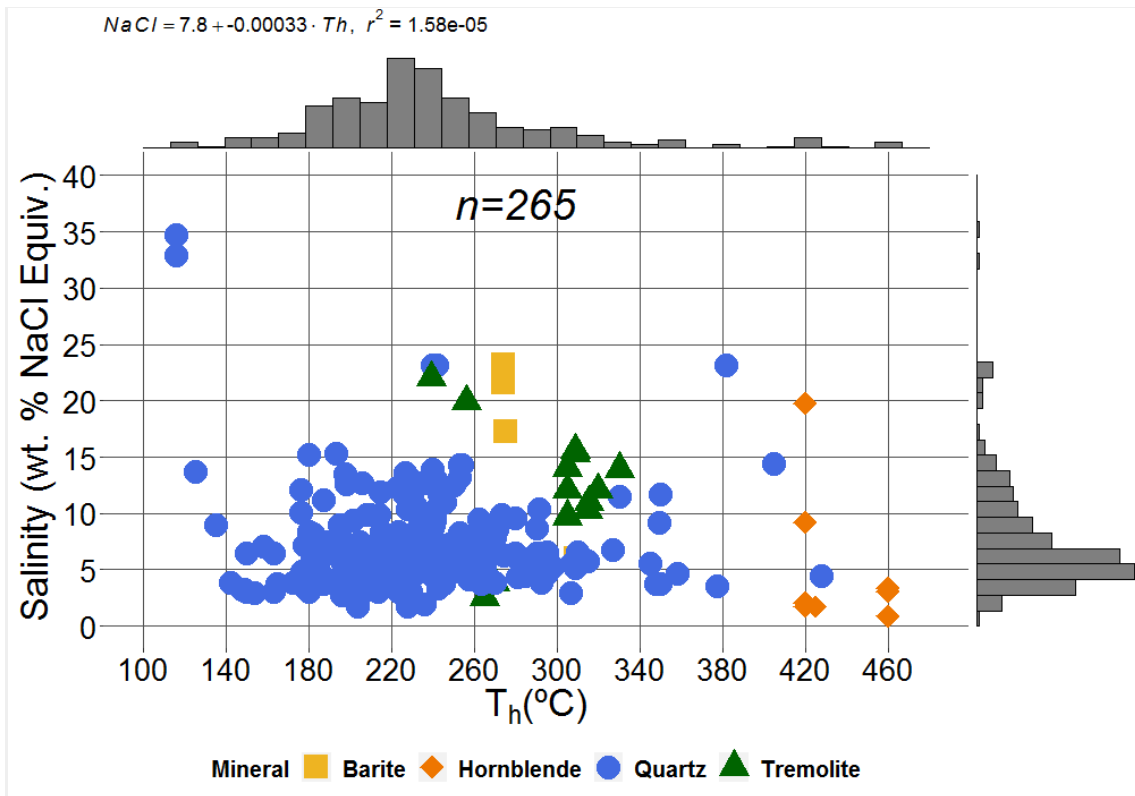


Figura 12.17. Projecção do gráfico contruído na função #17

Exercício Extra

Realize este exercício extra para compreender melhor o funcionamento dos pontos #1 a #17. Para este exercício utilize o ficheiro **exercício12-2.csv**, utilizado nos pontos supracitados.

```
#1 - Abrir o ficheiro - Vamos chamá-lo de "Temp"
Temp = read.csv2(file="dados/exercício12-2.csv",header=TRUE, dec=",")
```

Neste exercício o objetivo é, em primeiro lugar realizar gráficos de pontos com as colunas a + b, c + d, e + f, g + h e i + j. Projetando-os relativamente à condicionante Type.I, Type.II, Type. III, Type.IV e Type.V respetivamente.

Uma vez que são 5 gráficos em separado iremos denominar as funções de "a" a "e" respetivamente, em que:

```
#2 - Plot dos gráficos separadamente
a = ggplot(Temp, aes(x=Temp$a, y=Temp$b, group=Temp$Type.I)) +
  geom_point(aes(shape=Type.I, color=Type.I), size=7) +
  scale_shape_manual(values=c(19,19)) + xlab(expression(paste("T" [h], "(°C)"))) +
  ylab(expression(paste("Salinity (wt. % NaCl Equiv.)"))) +
  scale_color_manual(values=c("gray25","gray25"))+
  scale_x_continuous(limits=c(100, 480), breaks=seq(100, 480, 40)) +
  scale_y_continuous(limits=c(0, 40), breaks=seq(0,40, 5))+ expand_limits(x=100,
  y=10) +
  theme(legend.title = element_text(colour="black", size=15, face="bold")) +
  theme(legend.text = element_text(colour="black", size=12,
  face="bold"))+theme(legend.position="left") +
```

```

theme(axis.text = element_text(colour = "black", size=15),axis.title.x =
element_text(colour = "black", size=15),axis.title.y = element_text(colour =
"black", size=15),panel.background
element_rect(fill="grey100"),panel.grid.minor.y
element_line(size=3),panel.grid.major = element_line(colour =
"gray33"),plot.background = element_rect(fill="white")) +
annotate("text", x=360, y=32.5, label= "n=35", size = 5,fontface="italic")
a

b = ggplot(Temp, aes(x=Temp$c, y=Temp$d, group=Temp$Type.II)) +
geom_point(aes(shape=Type.II, color=Type.II), size=7) +
scale_shape_manual(values=c(19,19)) + xlab(expression(paste("T" [h], "(°C)")))
+
ylab(expression(paste("Salinity (wt. % NaCl Equiv.)"))) +
scale_color_manual(values=c("gray25","gray25"))+
scale_x_continuous(limits=c(100, 480), breaks=seq(100, 480, 40)) +
scale_y_continuous(limits=c(0, 40), breaks=seq(0,40, 5))+ expand_limits(x=100,
y=10) +
theme(legend.title = element_text(colour="black", size=15, face="bold")) +
theme(legend.text = element_text(colour="black", size=12,
face="bold"))+theme(legend.position="left") +
theme(axis.text = element_text(colour = "black", size=15),axis.title.x =
element_text(colour = "black", size=15),axis.title.y = element_text(colour =
"black", size=15),panel.background
element_rect(fill="grey100"),panel.grid.minor.y
element_line(size=3),panel.grid.major = element_line(colour =
"gray33"),plot.background = element_rect(fill="white")) +
annotate("text", x=360, y=32.5, label= "n=24", size = 5,fontface="italic")
b

c = ggplot(Temp, aes(x=Temp$e, y=Temp$f, group=Temp$Type.III)) +
geom_point(aes(shape=Type.III, color=Type.III), size=7) +
scale_shape_manual(values=c(19,19)) + xlab(expression(paste("T" [h], "(°C)")))
+
ylab(expression(paste("Salinity (wt. % NaCl Equiv.)"))) +
scale_color_manual(values=c("gray25","gray25"))
scale_x_continuous(limits=c(100, 480), breaks=seq(100, 480, 40)) +
scale_y_continuous(limits=c(0, 40), breaks=seq(0,40, 5))+ expand_limits(x=100,
y=10)+
theme(legend.title = element_text(colour="black", size=15, face="bold")) +
theme(legend.text = element_text(colour="black", size=12,
face="bold"))+theme(legend.position="left") +
theme(axis.text = element_text(colour = "black", size=15),axis.title.x =
element_text(colour = "black", size=15),axis.title.y = element_text(colour =
"black", size=15),panel.background
element_rect(fill="grey100"),panel.grid.minor.y
element_line(size=3),panel.grid.major = element_line(colour =
"gray33"),plot.background = element_rect(fill="white")) +
annotate("text", x=360, y=32.5, label= "n=84", size = 5,fontface="italic")
c

d = ggplot(Temp, aes(x=Temp$g, y=Temp$h, group=Temp$Type.IV)) +
geom_point(aes(shape=Type.IV, color=Type.IV), size=7) +
scale_shape_manual(values=c(19,19)) + xlab(expression(paste("T" [h], "(°C)")))
+
ylab(expression(paste("Salinity (wt. % NaCl Equiv.)"))) +
scale_color_manual(values=c("gray25","gray25"))+
scale_x_continuous(limits=c(100, 480), breaks=seq(100, 480, 40)) +
scale_y_continuous(limits=c(0, 40), breaks=seq(0,40, 5))+ expand_limits(x=100,
y=10) + theme(legend.title = element_text(colour="black", size=15,
face="bold")) +
theme(legend.text = element_text(colour="black", size=12, face="bold")) +
theme(legend.position="left") +
theme(axis.text = element_text(colour = "black", size=15),axis.title.x =
element_text(colour = "black", size=15),axis.title.y = element_text(colour =
"black", size=15),panel.background
element_rect(fill="grey100"),panel.grid.minor.y

```

```

element_line(size=3),panel.grid.major = element_line(colour =
"gray33"),plot.background = element_rect(fill="white")) +
annotate ("text", x=360, y=32.5, label= "n=56", size = 5,fontface="italic")
d

e = ggplot(Temp, aes(x=Temp$i, y=Temp$j, group=Temp$Type.V)) +
geom_point(aes(shape=Type.V, color=Type.V), size=7) +
scale_shape_manual(values=c(19,19)) + xlab(expression(paste("T" [h], "(°C)")))
+
ylab(expression(paste("Salinity (wt. % NaCl Equiv.)"))) +
scale_color_manual(values=c("gray25","gray25"))+
scale_x_continuous(limits=c(100, 480), breaks=seq(100, 480, 40)) +
scale_y_continuous(limits=c(0, 40), breaks=seq(0,40, 5))+ expand_limits(x=100,
y=10) +
theme(legend.title = element_text(colour="black", size=15, face="bold")) +
theme(legend.text = element_text(colour="black", size=12,
face="bold"))+theme(legend.position="left") +
theme(axis.text = element_text(colour = "black", size=15),axis.title.x =
element_text(colour = "black", size=15),axis.title.y = element_text(colour =
"black", size=15),panel.background
=
element_rect(fill="grey100"),panel.grid.minor.y
=
element_line(size=3),panel.grid.major = element_line(colour =
"gray33"),plot.background = element_rect(fill="white")) +
annotate("text", x=400, y=32.5, label= "n=66", size = 5,fontface="italic")
e

```

Nota: Repare que a função utilizada contém todas as funções para definição de escalas, cores, legendas dos eixos, tamanho dos pontos, e layout. Se achar mais conveniente realize o exercício passo por passo.

```

#3 - Colocar reta de maior declive e calcular R2 alternadamente para cada gráfico

##Reta para a
X=ggplot(Temp)
require(stats)
reg=lm(b ~ a, data = Temp)
reg
coeff=coefficients(reg)
eq = paste0("y = ", round(coeff[2],1), "*x + ", round(coeff[1],1))
a1=a + geom_abline(intercept = 7.327426, slope = -0.002401, size=2)
a1

##R2 para a
eq=substitute(italic(b) == a + b %.% italic(a)*", "~italic(r)^2~"=~r2, list(a
= format(coef(reg)[1], digits = 2), b = format(coef(reg)[2], digits = 2), r2 =
format(summary(reg)$r.squared, digits = 3)))
a2= a1 + geom_text(x = 100, y = 55, label=Temp$Type.I)+ ggtitle(eq)
a2

##Reta para b
X=ggplot(Temp)
require(stats)
reg=lm(d ~ c, data = Temp)
reg
coeff=coefficients(reg)
eq = paste0("y = ", round(coeff[2],1), "*x + ", round(coeff[1],1))
b1=b + geom_abline(intercept = 4.81345, slope = 0.01892, size=2)
b1

##R2 para b

```

```

eq=substitute(italic(d) == a + b %.% italic(c)*", "~italic(r)^2~"=~r2, list(a
= format(coef(reg)[1], digits = 2), b = format(coef(reg)[2], digits = 2), r2 =
format(summary(reg)$r.squared, digits = 3))

b2= b1 + geom_text(x = 100, y = 55, label=Temp$Type.II)+ ggtitle(eq)
b2

##Reta para c

X=ggplot(Temp)
require(stats)
reg=lm(f ~ e, data = Temp)
reg
coeff=coefficients(reg)
eq = paste0("y = ", round(coeff[2],1), "*x + ", round(coeff[1],1))
c1=c + geom_abline(intercept = -2.54810, slope = 0.04658, size=2)
c1

##R2 para c

eq=substitute(italic(f) == a + b %.% italic(e)*", "~italic(r)^2~"=~r2, list(a
= format(coef(reg)[1], digits = 2), b = format(coef(reg)[2], digits = 2), r2 =
format(summary(reg)$r.squared, digits = 3))

c2= c1 + geom_text(x = 100, y = 55, label=Temp$Type.II)+ ggtitle(eq)
c2

##Reta para d

X=ggplot(Temp)
require(stats)
reg=lm(h ~ g, data = Temp)
reg
coeff=coefficients(reg)
eq = paste0("y = ", round(coeff[2],1), "*x + ", round(coeff[1],1))
d1= d + geom_abline(intercept = 7.239449, slope = -0.006252, size=2)
d1

##R2 para d

eq=substitute(italic(h) == a + b %.% italic(g)*", "~italic(r)^2~"=~r2, list(a
= format(coef(reg)[1], digits = 2), b = format(coef(reg)[2], digits = 2), r2 =
format(summary(reg)$r.squared, digits = 3))

d2= d1 + geom_text(x = 100, y = 55, label=Temp$Type.II)+ ggtitle(eq)
d2

##Reta para e

X=ggplot(Temp)
require(stats)
reg=lm(j ~ i, data = Temp)
reg
coeff=coefficients(reg)
eq = paste0("y = ", round(coeff[2],1), "*x + ", round(coeff[1],1))
e1=e + geom_abline(intercept = 16.5081, slope = -0.0356, size=2)
e1

##R2 para e

italic(i)*", "~italic(r)^2~"=~r2, list(a = format(coef(reg)[1], digits = 2), b
= format(coef(reg)[2], digits = 2), r2 = format(summary(reg)$r.squared, digits
= 3))

e2= e1 + geom_text(x = 100, y = 55, label=Temp$Type.II)+ ggtitle(eq)
e2

```

Quando temos vários gráficos poderá ser uma boa opção tentar agrupá-los numa só imagem. O R Studio permite esta opção através do pacote `gridExtra`.

```
#4 - Agrupar os 5 gráficos numa só imagem
## Instalar o gridExtra

install.packages("gridExtra")
library(gridExtra)

## Agrupar os gráficos

grid.arrange(e2, arrangeGrob(d2, b2, c2, a2, ncol=2), nrow=2)
```

- Uma vez termos 5 gráficos e estarmos a escolher duas colunas (`ncol = 2`) e duas filas (`nrow = 2`) teremos de dizer à nossa função que um dos gráficos terá de ocupar a posição equilateral a dois gráficos. Por isto `e2` é colocado fora da função `arrangeGrob()`.

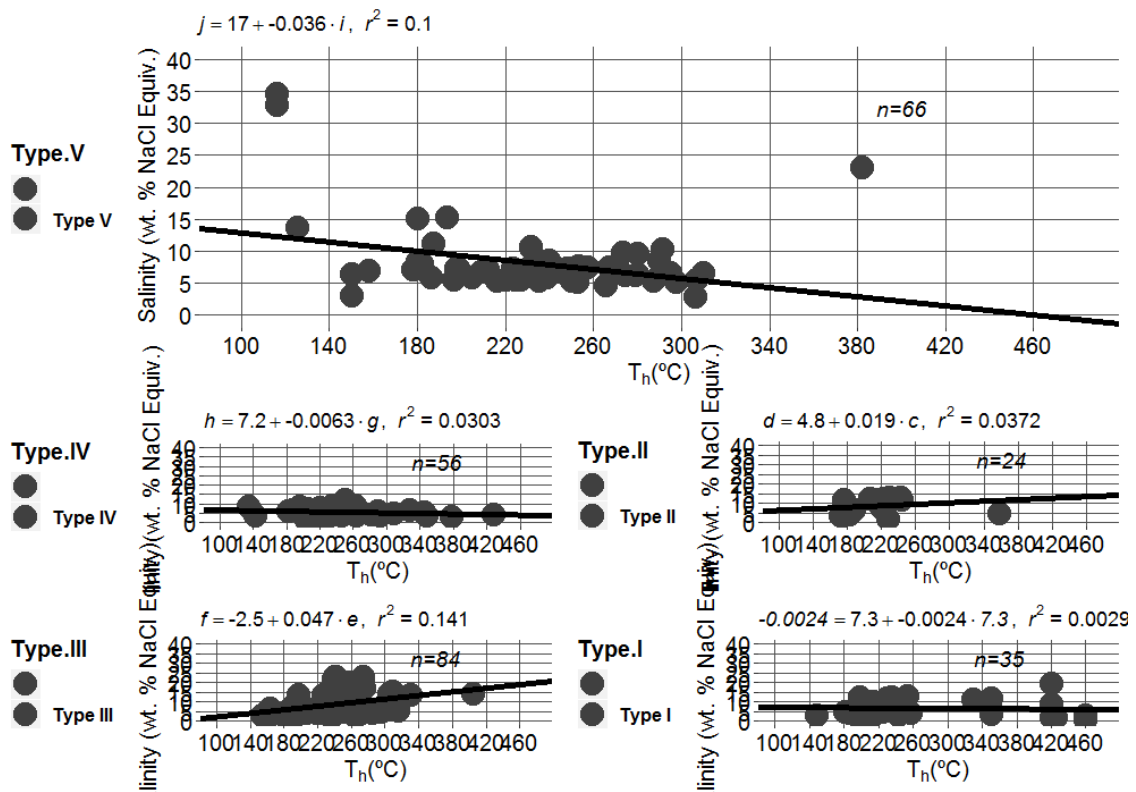


Figura 12.18. Projção do gráfico contruído na função #4

Gráficos de Linhas com aplicação (Espectroscopia Raman, DRX, FTIR, SEM-EDS, LA-ICP-MS)

A aplicação do ggplot é vasta, e ainda mais vasta é a aplicação do *Software* R e RStudio no que concerne o tratamento de dados de geoquímica, espectrometria, espectroscopia etc.

De seguida serão mostrados alguns exemplos de código R que poderá usar para projetar os seus dados lineares de p.e. Espectroscopia Raman ou LA-ICP-MS

Relembre-se que o R é versátil e poderá combinar novas funções para tornar os dados mais objetivos.

Projeção de gráficos de linhas

Comecemos por um exercício simples, projetar um único ficheiro num gráfico de linhas.

```
#1 - Abrir ficheiro 1.1.csv
a = read.csv2(file="dados/1.1.csv",header=TRUE,dec=",")

#2 - Inspeccionar os dados
View(a)

#3 - Projetar o nosso gráfico
a1=ggplot() +geom_line(data=a,aes(x=Wave, y=Counts, size=Spectra,
color=Spectra))
a1
```

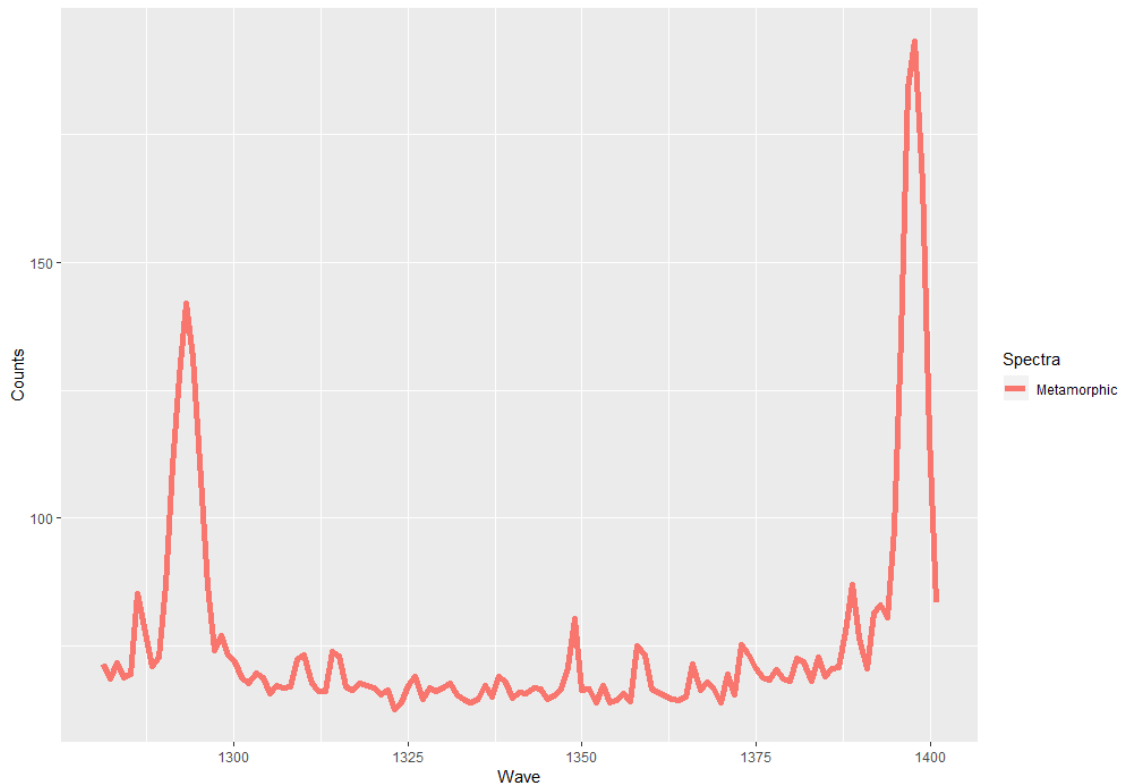


Figura 12.19. Projeção do gráfico contruído na função #3

Parece simples demais? Não gosta do limite dos eixos e cores?

Não se esqueça que no exercício anterior inspecionamos todas as funções necessárias para alterarmos o nosso gráfico. Vejamos como fica!

```
#4 - Colocar limites, alterar cores e formatar layout
a2=
  a1
  + scale_color_manual(values=c("gray1"))+
  scale_size_manual(values=c(1.5)) + xlab( expression(paste("Raman Shift","/",
  cm^{-1}))) + ylab ("Counts") + theme(axis.text = element_text(colour = "black",
  size=20),axis.title.x = element_text(colour = "black", size=20),axis.title.y =
  element_text(colour = "black", size=20),legend.text=element_text(size=17),
  legend.title= element_text(size=25),panel.background
  = element_rect(fill="grey100"),panel.grid.minor.y
  = element_line(size=3),panel.grid.major
  = element_line(colour
  = "gray33"),plot.background
  = element_rect(fill="white"))+
  scale_x_continuous(limits=c(1280, 1400), breaks=seq(1280, 1400, 40)) +
  scale_y_continuous(limits=c(50, 400), breaks=seq(50,400, 100))
a2
```

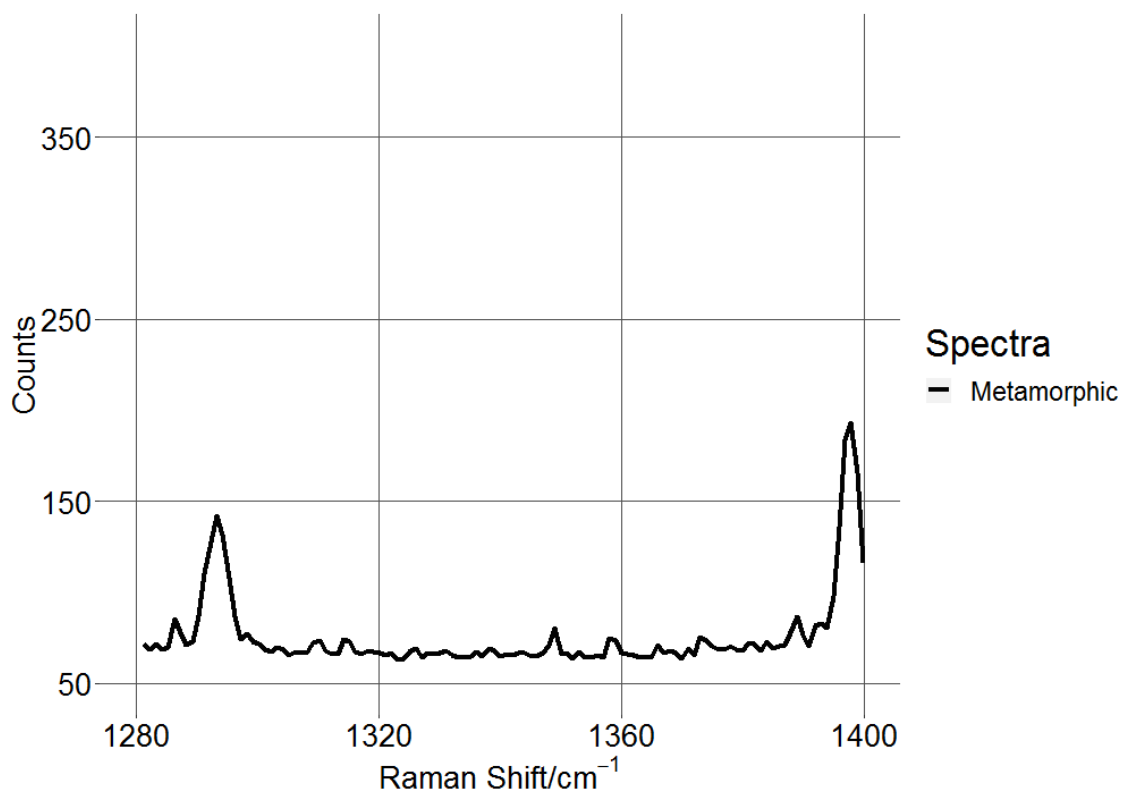


Figura 12.20. Projeção do gráfico contruído na função #4

Exercício: Repita a operação para os ficheiros seguintes (2.1 a 6.1). Tenha em atenção que deverá utilizar escalas para os eixos ajustadas aos seus dados. Conseguiu? Excelente!

Juntar gráficos num só

Se completou o exercício pedido deverá ter chegado ao final com 6 gráficos diferentes referentes aos dados de cada um dos ficheiros. Respetivamente do gráfico a2 ao gráfico a6.

Assim sendo seria uma excelente opção poder projetar os 6 gráficos juntos e obter uma mesma imagem ao invés de 6 gráficos em separado.

O R Studio permite esta opção através do pacote `gridExtra`.

```
#6 - Instalar o gridExtra

install.packages("gridExtra")
library(gridExtra)

#7 - Projetar os 6 gráficos juntos

library(gridExtra)
grid.arrange(arrangeGrob(a2, b2, c2, d2, e2, f2, ncol=2), nrow=1)
```

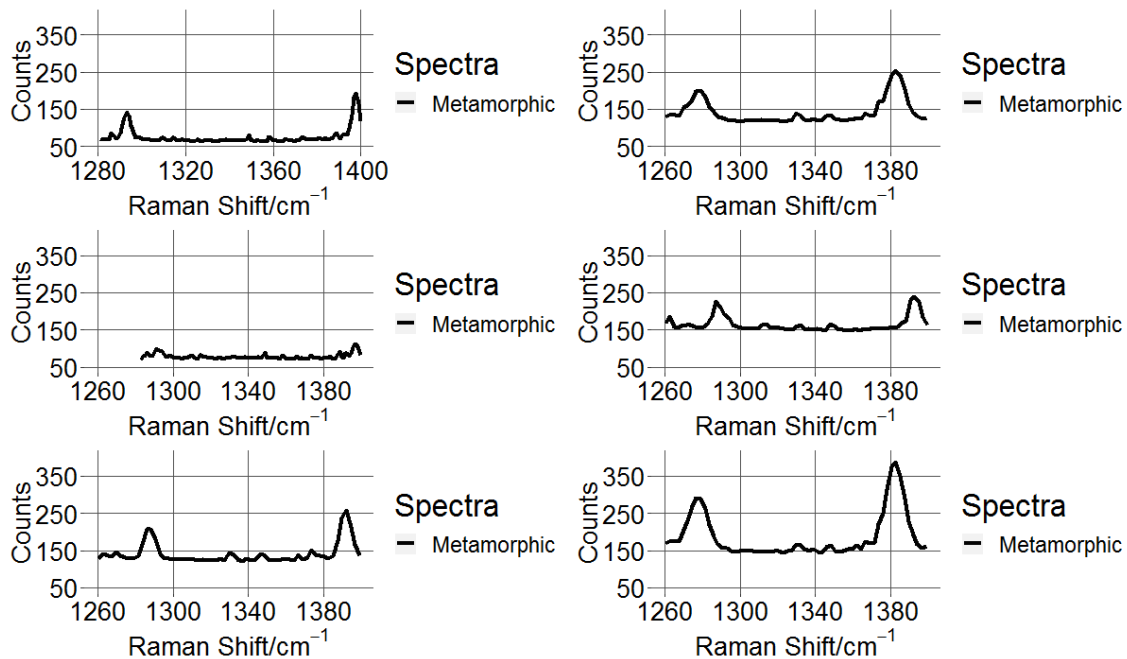


Figura 12.21. Projeção do gráfico contruído na função #7

Projetar várias linhas num mesmo gráfico

E se em vez de projetarmos os 6 gráficos anteriores, quisermos projetar os nossos dados num mesmo gráfico, sobrepostos por forma a comparar melhor os resultados?

Podemos para isso projetá-los e conjunto. Como? Voltemos à função inicial e chamemos-lhe agora "X" em vez de `a1`.

Mas primeiro vamos importar os ficheiros cujos dados queremos projetar.

```
#8 - Importar dados de ficheiros 1.1 a 6.1

a=read.csv2(file="dados/1.1.csv",header=TRUE,dec=",")
b=read.csv2(file=" dados/2.1.csv",header=TRUE,dec=",")
c=read.csv2(file=" dados/3.1.csv",header=TRUE,dec=",")
d=read.csv2(file=" dados/4.1.csv",header=TRUE,dec=",")
e=read.csv2(file=" dados/5.1.csv",header=TRUE,dec=",")
f=read.csv2(file=" dados/6.1.csv",header=TRUE,dec=",")

#9 - Projetar vários ficheiros num mesmo gráfico

X = ggplot() + geom_line(data=a,aes(x=Wave, y=Counts, size=Spectra,
color=Spectra))
```

- Como pode ver apenas temos representado os dados referentes ao ficheiro "a", ou seja ao ficheiro 1.1.csv, representado pela função `geom_line (data=a)` .
Precisamos então de adicionar os restantes ficheiros "b", "c", "d", "e" e "f" referentes aos ficheiros 2.1, 3.1, 4.1, 5.1 e 6.1.

Para isso temos apenas de adicionar as restantes funções `geom_line (data=)` para esses ficheiros.

```
X = ggplot() +geom_line (data=a, aes (x=Wave, y=Counts, size=Spectra,
color=Spectra))+geom_line (data=b, aes (x=Wave, y=Counts, size=Spectra,
color=Spectra)) + geom_line (data=c, aes (x=Wave, y=Counts, size=Spectra,
color=Spectra)) + geom_line (data=d, aes (x=Wave, y=Counts, size=Spectra,
color=Spectra)) + geom_line (data=e, aes (x=Wave, y=Counts, size=Spectra,
color=Spectra))+ geom_line (data=f, aes (x=Wave, y=Counts, size=Spectra,
color=Spectra))
X
```

Falta agora escolher limites dos eixos, cores, tamanho da linha.

```
X = X + scale_color_manual( values = c("gray1", "gray3", "gray5", "gray7",
"gray10", "gray13")) + scale_size_manual( values = c(1.5, 1.5, 1.5, 1.5, 1.5,
1.5)) + xlab( expression(paste("Raman Shift","/", cm^{-1}))) + ylab
("Counts")+theme(axis.text = element_text(colour = "black",
size=20),axis.title.x = element_text(colour = "black", size=20),axis.title.y =
element_text(colour = "black", size=20),legend.text=element_text(size=17),
legend.title= element_text(size=25),panel.background =
element_rect(fill="grey100"),panel.grid.minor.y =
element_line(size=3),panel.grid.major = element_line(colour =
"gray33"),plot.background = element_rect(fill="white"))+
scale_x_continuous(limits=c(1260, 1400), breaks=seq(1260, 1400, 40)) +
scale_y_continuous(limits=c(50, 400), breaks=seq(50,400, 100))+
annotate("text", x=1320, y=300, label= "n=10", size = 8,fontface="italic")
X
```

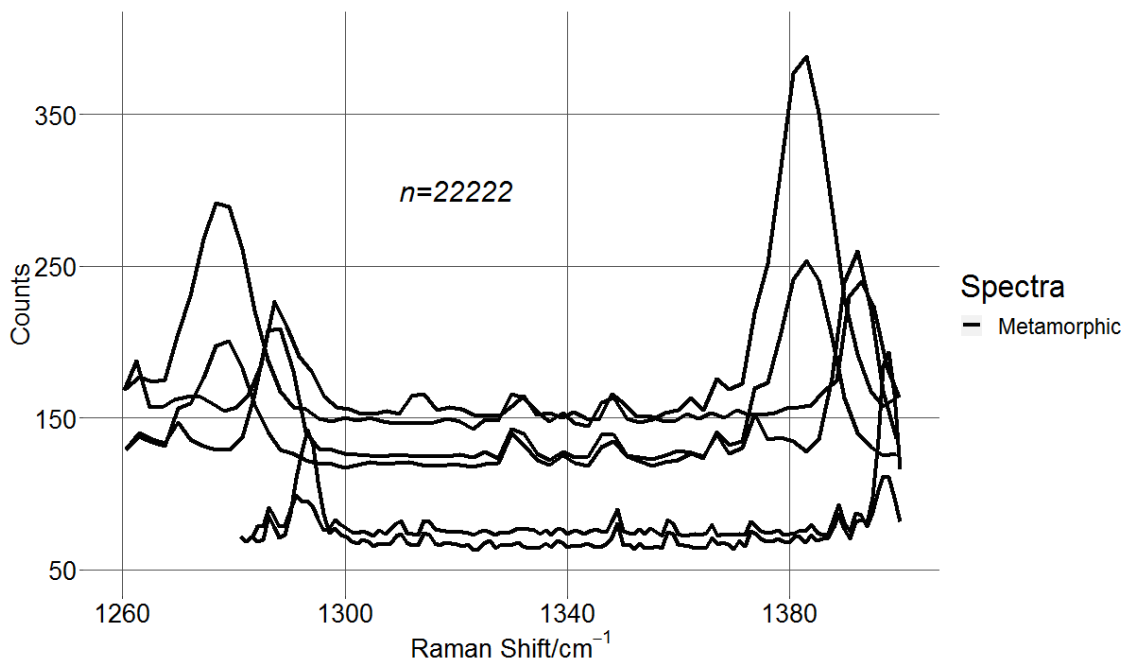


Figura 12.22. Projção do gráfico contruído na função #9

Exportar gráficos

Já vimos anteriormente de que forma podemos exportar os gráficos do R para o formato PNG ou PDF. No entanto as imagens são exportadas em fundo branco que por vezes não se adapta ao tipo de apresentação pretendida.

Deixamos-vos um pequeno e simples código que permite exportar gráficos em formato PNG com fundo transparente.

```
#10 - Exportar dados c/ fundo transparente  
  
png('Exercício 13.3.png',width=1300,height=900,units="px",bg = "transparent")  
print(X)  
dev.off()
```

Dentro da função `print()` devem colocar o nome dos vossos dados. Neste caso "X" referente ao ponto #9.

13- CRIAÇÃO DE FUNÇÕES EM R

Uma das vantagens de se utilizar uma linguagem de programação é que se não existirem funções para os cálculos que se pretendem realizar podemos criar as nossas próprias funções. Se pretende criar uma função para calcular a média ponderada (já existe uma, este é apenas um exemplo), para isso podemos criar a nossa própria função.

Vamos designar a função de mediaPond. Para se calcular a média ponderada é necessário ter dois vetores, um com os valores e outro com as ponderações. Os dois vetores têm de ter o mesmo número de elementos.

Para criar uma função use a função `function`, como se apresenta no exemplo abaixo.

```
mediaPond <- function(valores, pesos) {  
  if(length(valores)!=length(pesos)) {  
    stop("Os valores e os pesos não tem o mesmo comprimento")  
  }  
  
  if(sum(pesos) != 1) {  
    stop("O total dos pesos não soma 1")  
  }  
  
  resultado = sum(valores*pesos)  
  return(resultado)  
}
```

Uma função tem um conjunto de argumentos separados por vírgulas. O resultado da função deve ser devolvido com a função `return`. Esta função pode devolver uma variável de qualquer tipo. Pode utilizar as condições `if(condição) { ação }` para testar os argumentos de entrada e a função `stop` ou `warning` para transmitir uma mensagem de erro.

Depois de criada uma função pode utilizá-la como qualquer outra função em R. Veja o exemplo abaixo.

```
> a= c(1, 2, 3, 4, 10)  
> p= c(.1, .1, .1, .1, .1)  
> mediaPond(a, p)  
Error in mediaPond(a, p) : O total dos pesos não soma 1  
>  
> a= c(1, 2, 3, 4)  
> p= c(.1, .1, .1, .1, .1)  
> mediaPond(a, p)  
Error in mediaPond(a, p) :  
  Os valores e os pesos não tem o mesmo comprimento  
>
```

```
> a= c(1, 2, 3, 4, 5)
> p= c(.2, .2, .2, .2, .2)
> mediaPond(a, p)
[1] 3
```

Exercício 13.1

Comente os resultados obtidos.

Exercício 13.2.

Crie uma função para normalizar os valores de um vetor com base num padrão.

14- MODELAÇÃO SIMPLES: CORRELAÇÕES

As correlações são uma das formas mais diretas para analisar o comportamento de duas variáveis. Uma correlação é tanto mais forte quanto mais próximo da unidade é o seu valor.

Correlação e variância

O R possui uma função para avaliar a correlação entre duas variáveis. Vejamos dois exemplos simples.

```
# Duas sequencias, uma de 1 até 10 e outra de 23 a 32
x = seq(1,10,1)
y = seq(23,32,1)
cor(x, y)

# duas variáveis aleatórias
x = rnorm(10, 5, 5)
y = rnorm(10, 5 ,5)
cor(x, y)
```

O resultado é o expresso abaixo. Note que para o caso do segundo teste este foi executado duas vezes com dois conjuntos de valores aleatórios.

```
> x = seq(1,10,1)
> y = seq(23,32,1)
> cor(x, y)
[1] 1
> x = rnorm(10, 5, 5)
> y = rnorm(10, 5 ,5)
> cor(x, y)
[1] -0.2202422
> x = rnorm(10, 5, 5)
> y = rnorm(10, 5 ,5)
> cor(x, y)
[1] 0.1904389
```

Exercício 14.1.

Explique os resultados obtidos.

Explique a diferença entre o segundo e terceiro valor das correlações.

Modelação linear

Para modelar o comportamento de uma variável em função de outra o R possui um conjunto de ferramentas relativamente vasto. Aqui é apresentada uma abordagem simples para modelar uma regressão linear.

O exemplo abaixo cria dois conjuntos de pontos com uma variação aleatória. Essa variação aumenta com o valor de x (isto é, para a direita do gráfico), na razão $x/5$ e com a equação $y = m*x + b$.

```
x = seq(1, 100, 1)
x = x+ runif(100)
m = .5
b = 20
y = m* (x + runif(100) * (x/5)) + b
plot(x, y)
```

A função `runif` cria um conjunto de pontos (n) com uma distribuição uniforme.

O resultado é o apresentado na Figura 14.1.

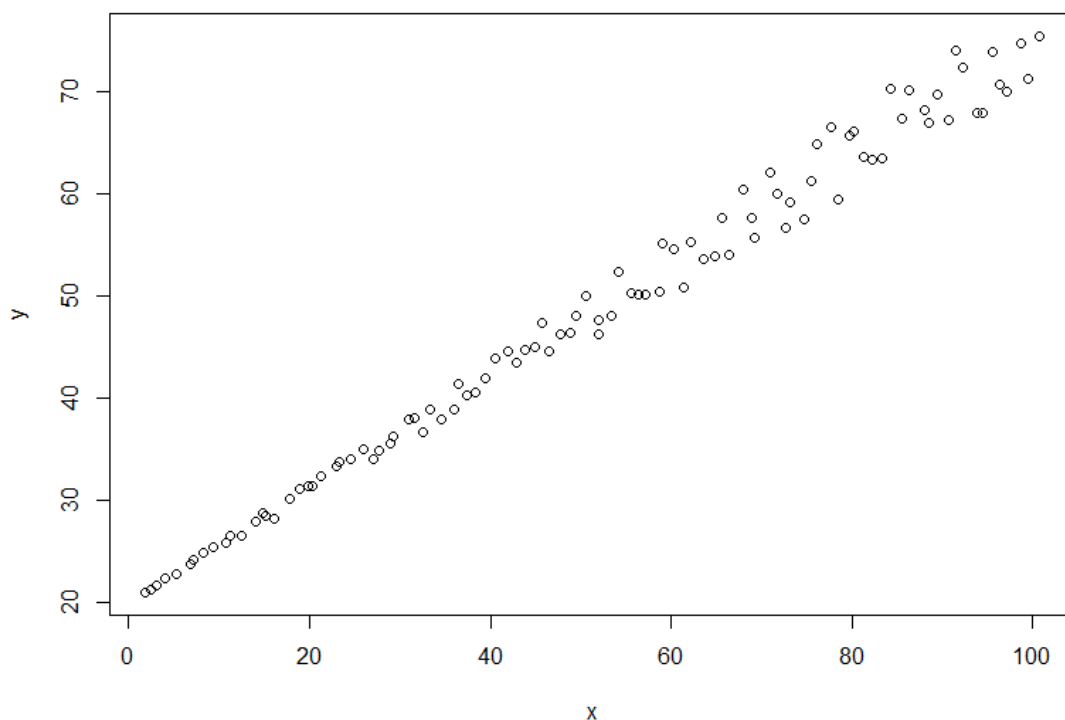


Figura 14.1. Gráfico que representa a relação entre dois valores segundo uma equação $y=m*x+b$.

Para se proceder à modelação dos dados deve ser proposta qual a função representativa da mesma. Neste caso concreto trata-se de uma função em que y é função de x (em R, $y \sim x$). O exemplo abaixo ilustra essa função e os resultados.

```

modeloExp= lm(y ~ x)

summary(modeloExp)

Call:
lm(formula = y ~ x)

Residuals:
    Min       1Q   Median       3Q      Max
-3.6489 -0.9695 -0.0380  0.7403  4.2198

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 20.377009   0.350859   58.08 <2e-16 ***
x            0.541997   0.005985   90.56 <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.727 on 98 degrees of freedom
Multiple R-squared:  0.9882, Adjusted R-squared:  0.9881
F-statistic: 8201 on 1 and 98 DF, p-value: < 2.2e-16

```

Os coeficientes previstos no `summary` são `[Intercept] = 20.377009` e `[x] = 0.541997`, valores bastante aproximados a aqueles que escolhemos originalmente.

Para se utilizar-se a função `predict` para determinar os parâmetros da referida função. O exemplo abaixo ilustra este procedimento.

```

Z = seq(1, 100, 1)

valoresExp= predict(modeloExp, list(z))

lines(valoresExp, lwd= 2, col= "blue", xlab= "x", ylab= "Counts")

```

O resultado é o conjunto de pontos previstos com base no modelo `[modeloExp]` para os pontos da lista `[z]`. O gráfico 14.2. ilustra os resultados obtidos.

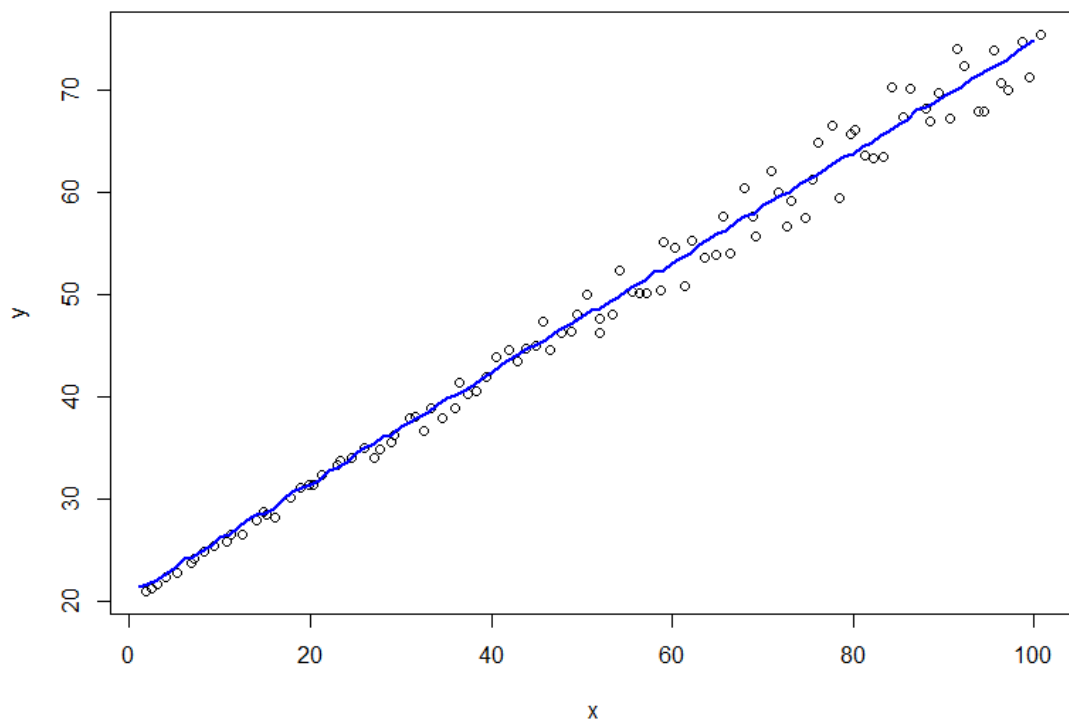


Figura 14.2. Linha com o ajuste obtido com o modelo de predição.

Exercício 14.2.

Crie o modelo para a equação $y \sim \log(x)$.

Solução:

```
#####  
# Modelo logaritmico  
#####  
x = seq(1,100,1)  
x = x + runif(100)  
y2 = log( x + runif(100)*(x/5))  
plot(x, y2)  
modeloExp = lm(y2 ~ log(x))  
summary(modeloExp)  
valoresExp <- predict(modeloExp, list(x))  
lines(valoresExp,lwd=2, col = "blue", xlab = "x", ylab = "Counts")
```

Diagramas de correlação

O R possui um *package* [corrgram] que permite criar gráficos de interpretação da correlação entre múltiplas variáveis. Este módulo é especialmente útil em análises geoquímicas quando se pretende analisar o comportamento de muitas variáveis.

O exemplo abaixo ilustra a utilização deste módulo na interpretação dos dados de terras raras de [manica].

```
library(corrgram)  
corrgram(reeManica,  
         lower.panel= panel.shade, upper.panel= panel.pie,  
         main="Terras Raras em Manica", na.rm=TRUE)
```

A Figura 14.1. apresenta o diagrama criado. Neste exemplo o canto inferior apresenta as linhas a indicar correlações positivas ou negativas. As cores têm o mesmo significado, isto é, o azul representa correlações positivas e o vermelho correlações negativas. O painel superior apresenta a força das correlações em forma de *pie chart*.

Terras Raras em Manica

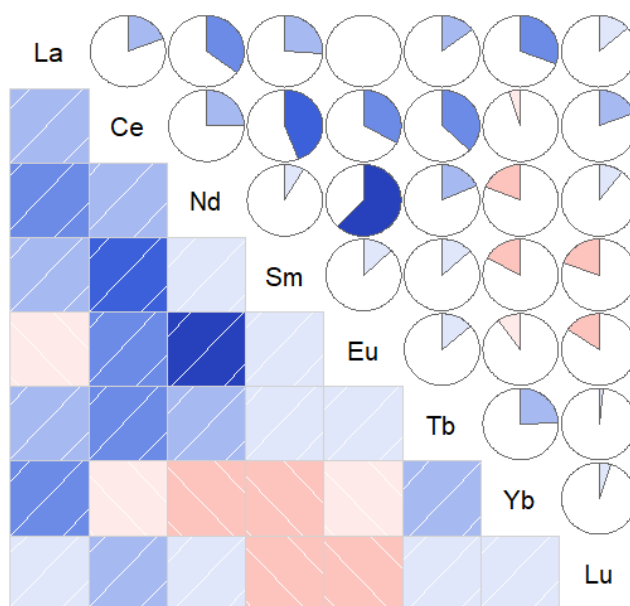


Figura 14.1. Diagrama de correlação para as terras raras de [manica].

Exercício 14.1.

Crie um diagrama de correlação para os elementos que não são terras raras da variável [manica].

15- ANÁLISE DE AGRUPAMENTOS- *CLUSTERS* (HCLUST)

Quando o conjunto de variáveis disponíveis é bastante grande deve-se procurar efetuar análises que permitam entender o comportamento dessas variáveis no seu conjunto. A utilização de correlações (capítulo 14) é desde logo uma primeira aproximação, mas uma abordagem com estatística multivariada permite retirar mais informação dos dados disponíveis efetuando uma análise do comportamento conjunto das variáveis.

Existem diversas abordagens que permitem estudar o comportamento das variáveis; no caso de estudos de geoquímica vai permitir separar comportamentos semelhantes ou distintos de certos elementos. Como exemplo de uma aplicação é aqui proposta uma análise por agrupamento hierárquico que utiliza a distância entre os diferentes elementos para a definição dos diferentes grupos. Naturalmente quanto menos distantes estiverem os elementos mais semelhantes eles vão ser.

Para se realizar este tipo de análise os dados devem ser exclusivamente numéricos e não devem conter *NAs*. A variância também deve ser superior a zero.

O exemplo aqui apresentado tem como origem os dados de [manica]. Para se efetuar a análise com agrupamento hierárquico foi necessário efetuar alguma preparação dos dados para se criar uma *data frame* nas condições de ser analisada.

Os valores de NA foram substituídos pela média dos valores dessa variável e quando existem apenas NA ou apenas um valor medido eliminou-se essa coluna. Decidiu-se ainda fazer a análise de dois grupos diferentes, por um lado as terras raras [reeManica] e por outro os restantes elementos [manicaOutros], essencialmente metais. As terras raras também foram normalizadas ao NASC, muito embora isso não altere o agrupamento dos elementos; a variável criada foi designada [reeManicaNAS].

```
#####  
# Limpar os valores abaixo  
# do limite de deteção  
# Neste caso os valores estão  
# representados com  
# valores negativos  
#####  
data=manica  
  
# Se o valor é negativo converte para NA  
for(i in 2:35){  
  data[data[,i]< 0, i] = NA  
}  
  
# Se o valor é NA converte para a média  
for(i in 2:35){  
  # Calcula a media se a coluna tem mais de um elemento  
  if(length(which(!is.na(data[,i])))> 1) {  
    data[is.na(data[,i]), i] <- mean(data[,i], na.rm= TRUE)  
  }  
}  
  
# Se os valores não tem media  
# ficam todos NA e a media fica NAN  
# Apaga colunas com NAs
```

```

data = data[colSums(is.na(data)) == 0]
manica=as.data.frame(data)

reeManica= data.frame(manica$Sample.ID, manica$La, manica$Ce, manica$Nd,
manica$Sm, manica$Eu, manica$Tb, manica$Yb, manica$Lu, manica$Zona)

#####
# Cria a lista de
# Terras raras Normalizada
#      NASC
#####
elementos=c("La","Ce","Nd","Sm","Eu","Tb","Yb","Lu")

NAS = c(31.1, 66.7, 27.4, 5.59, 1.18, .85, 3.06, .456)

normalizado = reeManica[,2:9]/NAS

reeManicaNAS = cbind(manica$Sample.ID, normalizado, manica$Zona)

colnames(reeManicaNAS) = c("Amostra", elementos, "Zona")

summary(reeManicaNAS)

#####
# Cria a Lista dos outros elementos
# apagando os que são Terras Raras
#####
manicaOutros=manica

manicaOutros[elementos] = NULL

summary(manicaOutros)

```

Em primeiro lugar é criada uma variável intermédia [data] para realizar as operações de transformações dos dados.

Os ciclos `for() {}` são utilizados para percorrer todas as colunas numéricas e alterar os valores menores que zero para NA. De seguida eliminam-se as colunas em que todos os valores são NA. No terceiro passo substituem-se os valores de NA ainda restantes pelo valor da média da respetiva coluna. Finalmente apaga as colunas que ainda mantêm NA. O valor final da variável é passado novamente para [manica]. É também criada a variável [reeManica] com os valores das terras raras.

O passo seguinte trata de normalizar o valor de [reeManica] dividindo os valores dos diferentes elementos pelo valor normalizado do NASC. Depois de normalizado acrescenta as colunas [Amostras] e [Zona] para se poder classificar as diferentes amostras.

O último passo desta primeira parte trata de criar a variável [manicaOutros] que utiliza todas as variáveis de [manica] com exceção dos elementos de terras raras [elementos].

Neste momento as variáveis [manicaOutros] e [reeManica] estão preparadas para se efetuar a análise de agrupamentos.

A análise inicia-se pela criação de uma matriz com a correlação entre todas as variáveis, seguida do cálculo da distância entre os diferentes elementos. A função `hclust` vai depois utilizar a classificação hierárquica e determinar os parâmetros da classificação. Por fim o `plot` permite criar o dendrograma ilustrativo das relações entre as diferentes variáveis. O exemplo abaixo ilustra os passos aqui referidos.

```
#####
# Calculo do hclust
#####
# Esta versão utiliza o valor da correlação
# mas as correlações fortes negativas
# são consideradas como fracas
#####
dissimilaridade = 1 - cor(manicaOutros[,2:18])
distancia = as.dist(dissimilaridade)
grupos = hclust(distancia)
plot(grupos, main="Dissimilaridade = 1 - Correlação", xlab="")
```

O resultado do gráfico está apresentado na Figura 15.1.

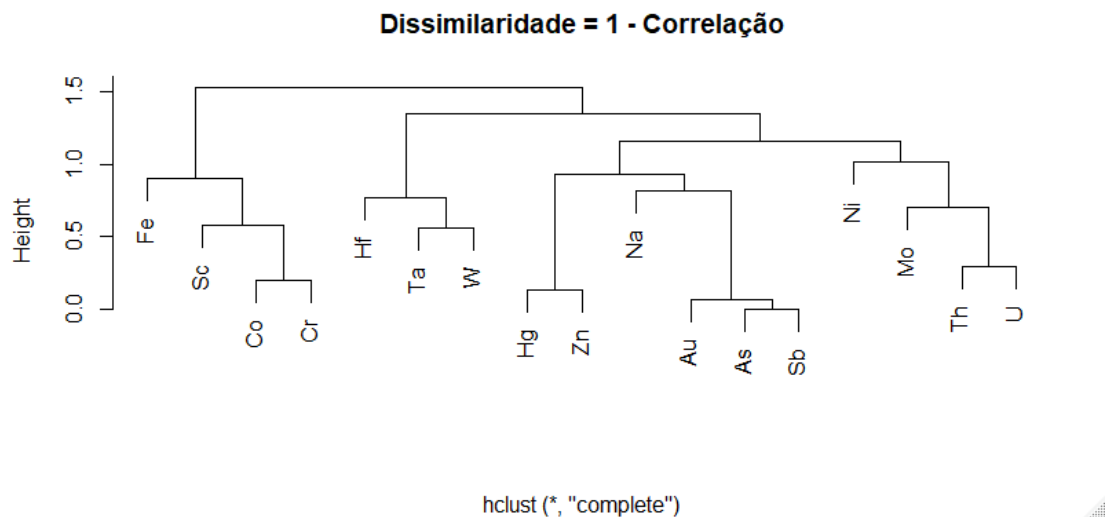


Figura 15.1. Dendrograma com agrupamento dos elementos metálicos de [manica].

Esta abordagem permite desde logo avaliar os agrupamentos definidos pela correlação entre os diversos elementos. É de notar que a operação `1 - cor(valor)` é utilizada para que os valores mais próximos sejam representados próximos da base.

Deve ter em consideração que ao utilizar a correlação entre os elementos as correlações negativas vão corresponder a uma grande distância entre os elementos. Atente no diagrama da figura 15.2.

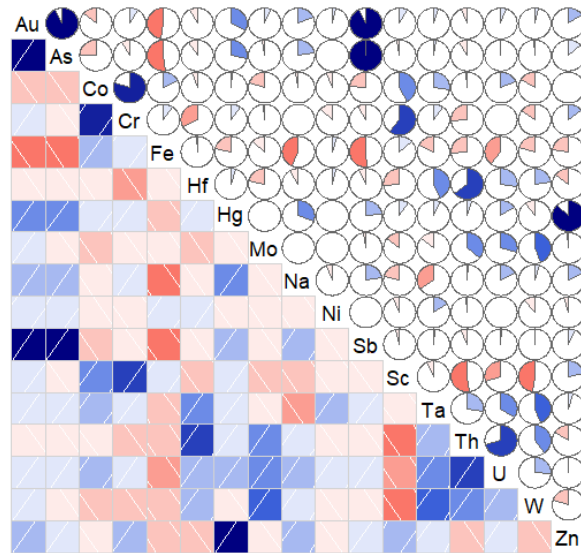


Figura 15.2. Diagrama de correlação da amostra [manicaOutros].

Assim, pode-se observar que o Fe tem correlações negativas fortes com o Au, As, Sb e Na, colocando-se em grupos muito distantes. Uma forma de procurar obviar a esta distorção na análise é considerar o módulo do valor da correlação e não o valor da correlação em si. Para isso é necessário alterar ligeiramente o código, como ilustra o exemplo abaixo.

```
#####
# O mesmo mas com 1-abs(cor())
# Neste caso ao utilizar a
# função abs, as correlações fortes
# negativas são consideradas fortes
#####
dissimilaridade <- 1 - abs(cor(manicaOutros[,2:18]))

distancia <- as.dist(dissimilaridade)

grupos = hclust(distancia)

plot(grupos, main="dissimilaridade = 1 - Abs(Correlação)", xlab="")
```

O resultado altera-se radicalmente, consoante se pode constatar na Figura 15.3.

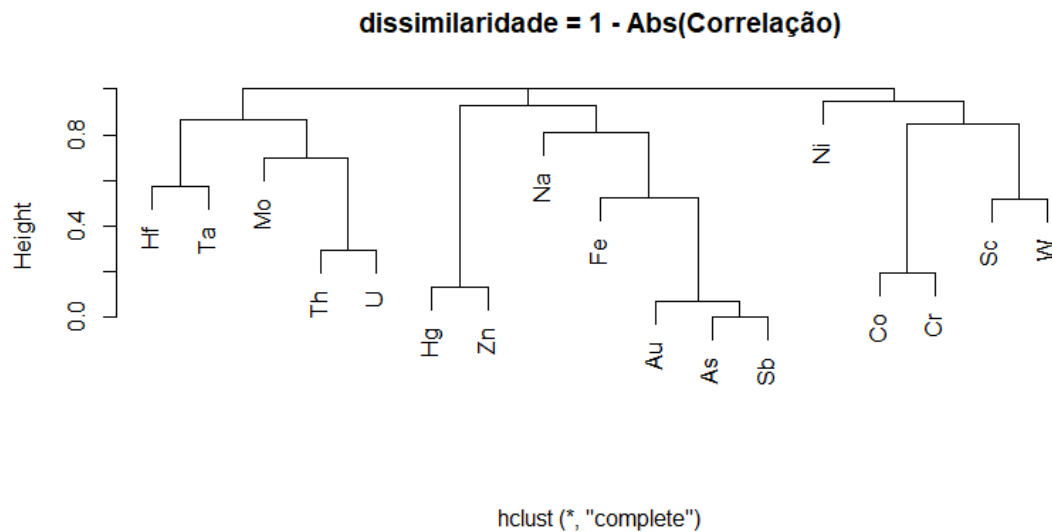


Figura 15.3. Dendrograma com os valores absolutos das correlações.

Pode agora verificar que o Fe passa a estar no grupo do Au, As e Sb uma vez que possui uma forte relação com estes elementos, muito embora seja uma correlação negativa.

Se pretender salientar os agrupamentos criados pode criar um retângulo à volta do número de grupos que pretender.

```
# Desenha caixa à volta dos grupos
rect.hclust(grupos, k= 3, border= "red")
```

A figura com três grupos assinalados pela linha vermelha é a apresentada na Figura 15.4.

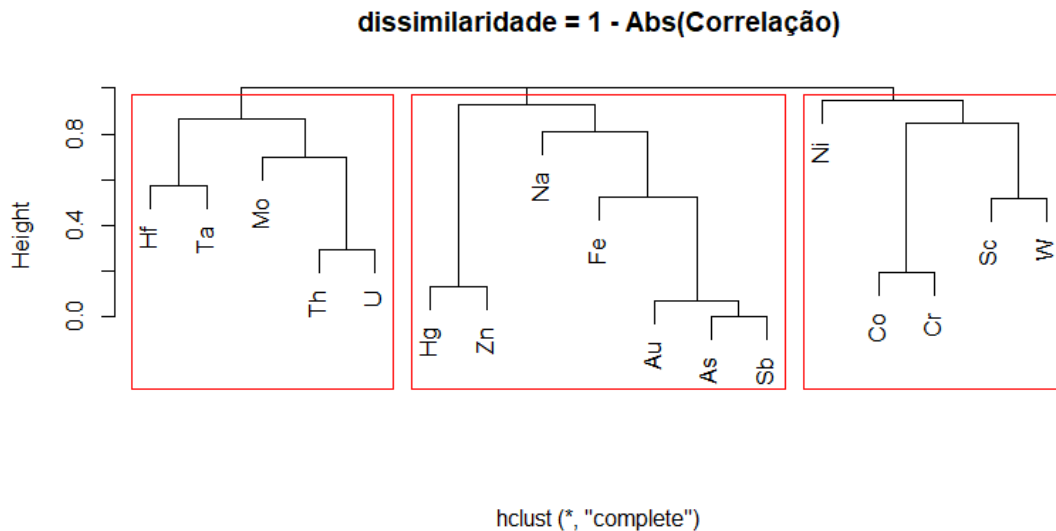


Figura 15.4. A variável [manicaOutros] dividida em 3 agrupamentos.

Pode verificar a correspondência de cada elemento num grupo através da função `cuttree`, como está apresentado abaixo.

```
> gruposCortado <- cutree(grupos, k= 3)
> sort(gruposCortado)
Au As Fe Hg Na Sb Zn Co Cr Ni Sc W Hf Mo Ta Th U
1 1 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
```

Estes grupos parecem indicar três comportamentos distintos para os elementos. Um grupo com comportamento comum do Au, As, Sb, Fe, Na, Hg e Zn, com relações fortes, um segundo grupo com comportamento semelhante do U, Th, Mo, Hf e Ta e um terceiro grupo constituído por Co, Cr, Ni, Sc e W.

Estes grupos parecem indicar um controlo mineralógico, no primeiro grupo controlado pela presença de minerais associados ao ouro e arsénio com a presença de mercúrio que pode estar associado aos trabalhos de garimpo. O segundo grupo possivelmente controlado por minerais do grupo do zircão contendo U, Th e Hf. O terceiro grupo pode ser controlado pela presença de minerais do grupo das cromites, com columbo-tantalites e volframites.

A análise de agrupamentos pode ainda ser realizada para verificar as relações entre as diferentes amostras. Para isso basta usar a *data frame* transposta de [manicaOutros], utilizando a função de matriz transposta, simplesmente `t(valor)`, e calcular novamente a classificação, consoante o exemplo abaixo.

```
#####
# Relação entre
# as amostras
#####
m = t(manicaOutros[,2:18])

colnames(m) = manica$Sample.ID

dissimilaridade = 1 - abs(cor(m))

distancia = as.dist(dissimilaridade)

grupos = hclust(distancia)

plot(grupos, main="dissimilaridade = 1 - Abs(Correlação)", xlab="Amostras")

rect.hclust(grupos, k = 3, border = "red")
```

O resultado é o apresentado na figura 15.5.

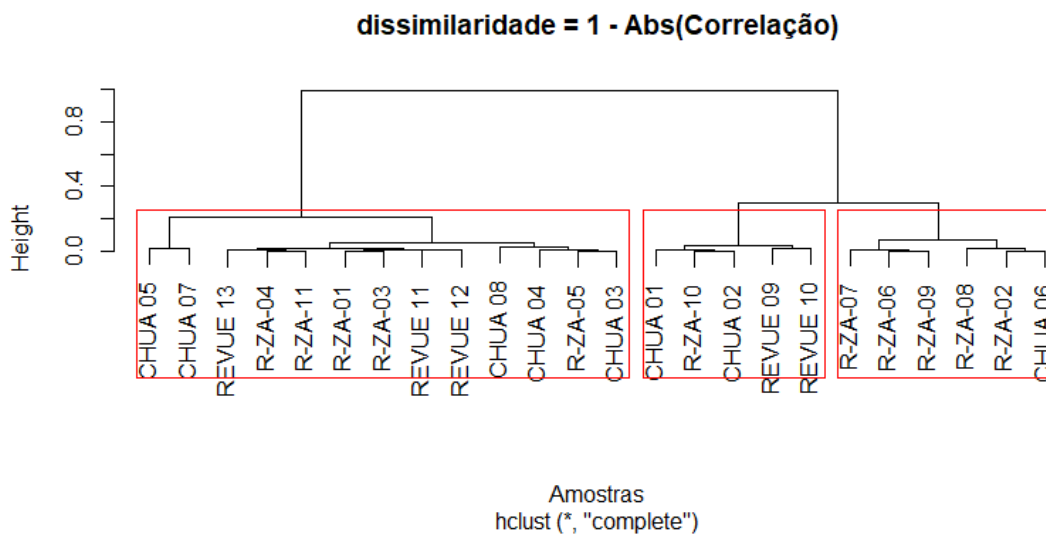


Figura 15.5. Análise de agrupamentos para as amostras.

16- ANÁLISE DE COMPONENTES PRINCIPAIS (PCA)

Quando trabalhamos em dados geológicos, principalmente geoquímicos, muitas vezes coloca-se o problema de que ter muitas variáveis não é uma vantagem, mas antes um problema, pois torna-se difícil separar aquelas que são significativas e mostram as variações entre as nossas amostras ou observações. Este problema é muitas vezes designado de *maldição da dimensionalidade*. A análise de componentes principais (PCA- *Principal Component Analysis*, em inglês) determina qual o eixo que representa a maior variância, num modelo de variação linear, no espaço multidimensional, e designa esse eixo como primeira componente. A segunda componente corresponde à fração da informação que não é explicada por esta componente e é designada por segunda componente. Este processo é repetido procurando sempre os eixos que explicam a variância ainda restante.

Em R existem diversas bibliotecas para realizar a análise de componentes principais; a biblioteca de base [stats] possui a função `prcomp` para esta função.

No exemplo aqui apresentado vai ser utilizado o mesmo conjunto de dados do capítulo 15, assim assumimos que todas as transformações e limpezas de dados já foram realizadas e as variáveis possuem os mesmos nomes e estrutura.

O código abaixo ilustra a aplicação da função `prcomp` e os valores obtidos com esta função.

```
# 1- Criação da PCA
manicaCluster = manica[,2:26]
manicaPCA = prcomp(manicaCluster, center= T, scale=T )# 2- Análise dos valores
summary(manicaPCA)
head(manicaPCA$rotation)
head(manicaPCA$x)
head(manicaPCA$sdev ^2)
```

O resultado destas funções é o seguinte.

```
>summary(manicaPCA)
Importance of components:

          PC1      PC2      PC3      PC4      PC5      PC6      PC7      PC8      PC9
Standard deviation  1.930 1.8177 1.4975 1.29302 1.22940 1.06603 0.99681 0.90076 0.7769
Proportion of Variance 0.219 0.1943 0.1319 0.09835 0.08891 0.06685 0.05845 0.04773 0.0355
Cumulative Proportion 0.219 0.4134 0.5453 0.64364 0.73255 0.79939 0.85784 0.90557 0.9411

>head(manicaPCA$rotation)

          PC1      PC2      PC3      PC4      PC5      PC6      PC7
Au  0.42743715 -0.19744593  0.001777554 -0.24568016  0.07981478 -0.074226804  0.003304597
As  0.44762012 -0.18352083  0.092877733 -0.23749647  0.01348159  0.008761381 -0.028335940
Co -0.18663743 -0.12185318 -0.503623250 -0.17203296  0.15825386  0.231581089  0.126079118
Cr -0.12378649 -0.27582885 -0.439535487 -0.10876093  0.32078005  0.031213309  0.025251977
Fe -0.38783489 -0.03468681  0.108468699 -0.05457964 -0.10968994 -0.141029670 -0.053856699
Hf  0.04709818  0.33386431 -0.132150217 -0.17425461 -0.42579419  0.217625109 -0.133588852
```

```

> head(manicaPCA$x)
      PC1      PC2      PC3      PC4      PC5      PC6      PC7
R-ZA-01  0.5576317 -0.9071719  0.6325159  2.5023432  1.47426584  2.4323136  1.192681553
R-ZA-02 -1.3748320 -1.8531423 -0.9724287 -0.5851114  0.53733920 -0.5357069  0.191527164
R-ZA-03 -1.2066341 -1.4610229 -1.5604088 -0.8358727  0.62738007 -0.9055931 -0.152304531
R-ZA-04 -1.3174861 -1.6204681 -1.9009266  0.5992430  2.11781069  1.4062013 -2.852311151
R-ZA-05 -1.3869167 -1.1255858 -1.1249329 -0.5103080 -0.09253181 -0.6145383  2.945807257
R-ZA-06 -0.4926754 -0.9473581 -0.5624407 -0.2699768  0.50610743 -0.2221422  0.004335516

> manicaPCA$sdev ^2
[1] 3.7236497192 3.3039177624 2.2423950908 1.6718988740 1.5114350825 1.1364176205 0.9936291781

```

São apresentados apenas os resultados parciais para simplificação da análise. Verifique que a função `summary` permite analisar o desvio padrão, a proporção de variância e a proporção cumulativa da mesma. Um critério usado para definir quantas componentes considerar é o de Kaiser, que indica que todos os valores com desvio padrão inferior a 1 devem ser rejeitados. No exemplo estudado significa utilizar as primeiras seis componentes.

A variável `rotation` apresenta o peso de cada variável numa determinada variável. A variável `x` apresenta o peso de cada amostra num determinado componente.

Uma outra forma de representar o peso de cada fator é através de um gráfico de linhas em que o peso de cada componente é representado. O exemplo abaixo apresenta o referido gráfico.

```

screplot(manicaPCA,type="line", main = "Fatores PCA")
abline(h= 1, lty= "dotted", col= "grey3")

```

A figura 16.1. Ilustra o gráfico obtido.

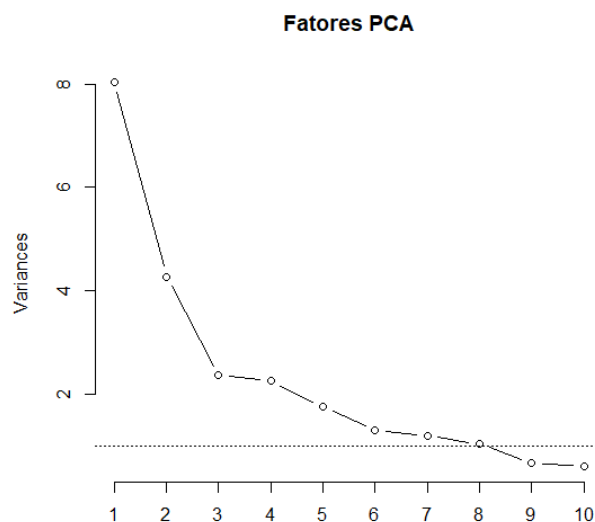


Figura 16.1. Peso de cada componente na explicação da variância.

No gráfico é desenhada a linha de variância igual a 1 para ilustrar o referido anteriormente. Verifique que as primeiras seis componentes estão acima do valor de 1 enquanto que as restantes estão abaixo.

Os componentes principais muitas vezes são apresentados como um gráfico XY com duas variáveis diferentes projetadas. A primeira representa a posição das variáveis (e.g. As, Au, Fe, etc.) normalmente representado como setas e a segunda a posição das amostras, sob a forma de pontos.

O exemplo seguinte utiliza a função `biplot` para fazer a representação deste tipo de gráficos.

```
# 1- O gráfico
biplot(manicaPCA,
       scaling= 1,
       arrow.len = .10,
       col = c("grey4", "blue"),
       cex=c(.6, .8))
abline(h=0,v=0,
       col="grey4", lty="dotted")
# 2- Os pontos das amostras
points(manicaPCA$x[,1], manicaPCA$x[,2],
       pch=16, col=as.numeric(dadosCluster$Zona))
# 3- A legenda
legend(4,8, legend = c("Chua", "Revue", "ZA"),
      col=c(1,2,3), pch = c(16,16,16)
      )
```

A Figura 16.2. é o resultado da representação.

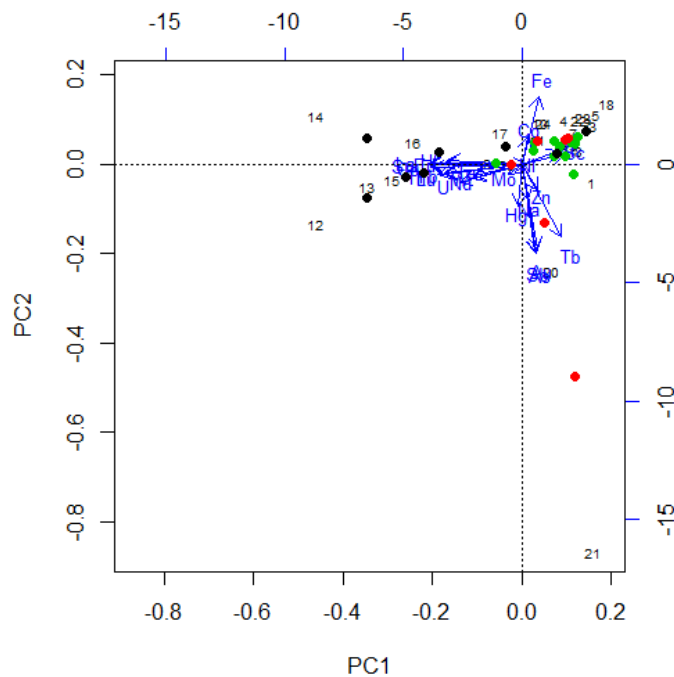


Figura 16.2. Gráfico com as componentes principais e a posição das variáveis (setas) e das amostras (pontos).

Pode-se observar que a componente 1 (PC1) é sobretudo influenciada pelos valores de As, Au e Sb positivos e Fe negativos (note que Fe apresenta uma correlação forte negativa com estes elementos, ver Figura 15.2). A segunda componente é influenciada pelos elementos Th, W, Hf e U pelo lado positivo e Sc e Cr na parte negativa deste eixo.

Esta mesma associação já tinha sido encontrada aquando da análise de agrupamentos e prende-se com o facto das amostras se poderem distinguir entre aquelas que possuem ouro (com arsénio e antimónio associado) como principal mineral pesado e as que possuem tório e urânio (possivelmente no zircão) como o mineral pesado dominante.

Pode-se ainda antever que as amostras REVUE09 e REVUE10 são as que estão na direção das variáveis Au, As e as amostras CHUA01 e CHUA02 na direção dos elementos U e Th estando no mesmo alinhamento da maioria das amostras estudadas. Assim deve ser salientado o comportamento distinto sobretudo das amostras REVUE09 e REVUE10 que estão fora deste alinhamento e que deverão corresponder a valores anómalos para o ouro.

ÍNDICE REMISSIVO

abline, 47, 49
agreggate, 31
as.numeric, 25
barplot, 37
 horiz, 38
boxplot, 51
cbind, 31
colnames, 31
facet_grid(), 58
for() {}, 89
function, 81
 return, 81
 stop, 81
 warning, 81
gather, 60
geom_histogram(), 55
geom_point(), 54
ggplot, 54
 aes(), 54
 annotate, 65
 data, 54
 expand_limits, 67
 geom_text, 69
 gridExtra, 75
 intercept, 69
 scale_color_manual, 65
 scale_shape_manual, 65
 slope, 69
 xlab, 65
 ylab, 65
hclust, 89
head, 23, 60
hist, 39, 40
if, 81
is.numeric, 25
lm, 47
mean, 26, 31
melt, 60
par, 50
 mfcol, 50
 mfrow, 50
paste0, 46
plot, 34
 lty, 35
 main, 35
 pch, 35
 sub, 35
 type, 35
 xlab, 35
 ylab, 35
points, 46
prcomp, 94
print, 80
rbind, 31
require, 46
rnorm, 48
runif, 84
substr, 28
summary, 29
 r.squared, 48
text, 48
 cex, 48
 col, 48
View, 23

ANEXO 1

CÁBULAS DE FUNÇÕES

Operadores aritméticos

Operador	Descrição
+	adição
-	subtração
*	multiplicação
/	divisão
^ or **	exponenciação
x %% y	módulo (x mod y) 5 %% 2 é 1
x %/ y	divisão inteira 5 %/ 2 é 2

Operadores lógicos

Operador	Descrição
<	menor que
<=	menor ou igual a
>	maior que
>=	maior ou igual a
==	exatamente igual a
!=	diferente de
!x	Não x
x y	x OU y
x & y	x E y
isTRUE(x)	testa se X é verdade (TRUE)

Funções numéricas e constantes

Função	Descrição
abs(x)	valor absoluto
sqrt(x)	raiz quadrada
ceil(x)	arredonda para cima (3.475) é 4
floor(x)	arredonda para baixo (3.475) é 3
trunc(x)	truncar (5.99) é 5
round(x, digits=n)	arredonda (3.475, digits=2) é 3.48
signif(x, digits=n)	valor significativo (3.475, digits=2) é 3.5
cos(x), sin(x), tan(x)	também existe acos(x) , cosh(x) , acosh(x) , etc.
log(x)	logaritmo neperiano
log10(x)	logaritmo de base 10
exp(x)	e^x
pi	valor de pi (3.14159...)
2.5e2	notação científica 2.5×10^2

Funções estatísticas

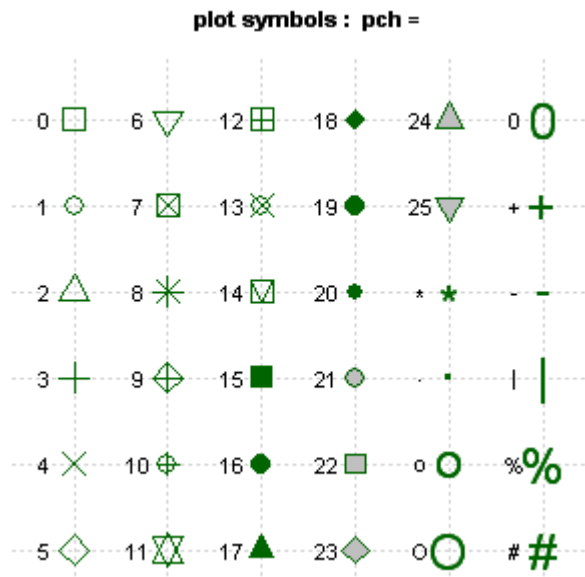
Função	Descrição
mean(x, trim=0, na.rm=FALSE)	média do objeto x # média truncada, removendo valores em falta e # 5 por cento dos valores superiores e inferiores mx <- mean(x,trim=.05,na.rm=TRUE)
sd(x)	desvio padrão do objeto(x). ver também var(x) para a variância e mad(x) para desvio absoluto da mediana.
median(x)	mediana
quantile(x, probs)	quantis em que x é o vetor numérico cujos quantis são calculados e probs é um vetor numérico com probabilidades no intervalo in [0,1]. # 30 e 84 percentil de x y <- quantile(x, c(.3,.84))
range(x)	máximo e mínimo
sum(x)	soma
min(x)	mínimo
max(x)	máximo

Outras funções

Function	Description
seq(from , to, by)	Cria uma sequencia indices <- seq(1,10,2) #indices is c(1, 3, 5, 7, 9)
rep(x, ntimes)	repetir x n vezes y <- rep(1:3, 2) # y is c(1, 2, 3, 1, 2, 3)
cut(x, n)	Divide uma variável em fatores com n níveis y <- cut(x, 5)

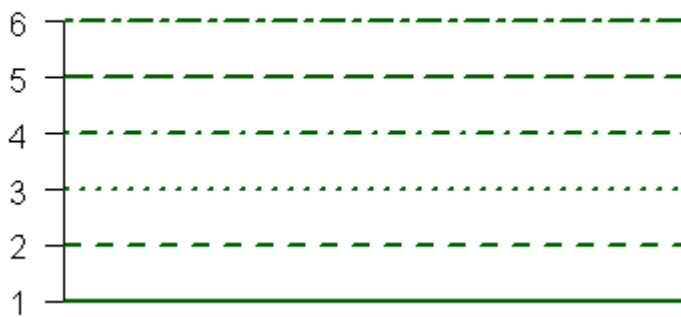
CÁBULAS DE ARGUMENTOS PARA GRÁFICOS

Símbolos



Tipos de linhas

Line Types: lty=



Cores

coral3	deeppink4	gray27	gray87	grey39
coral2	deeppink3	gray26	gray86	grey38
coral1	deeppink2	gray25	gray85	grey37
coral	deeppink1	gray24	gray84	grey36
chocolate4	deeppink	gray23	gray83	grey35
chocolate3	darkviolet	gray22	gray82	grey34
chocolate2	darkturquoise	gray21	gray81	grey33
chocolate1	darkslategrey	gray20	gray80	grey32
chocolate	darkslategray4	gray19	gray79	grey31
chartreuse4	darkslategray3	gray18	gray78	grey30
chartreuse3	darkslategray2	gray17	gray77	grey29
chartreuse2	darkslategray1	gray16	gray76	grey28
chartreuse1	darkslategray	gray15	gray75	grey27
chartreuse	darkslateblue	gray14	gray74	grey26
cadetblue4	darkseagreen4	gray13	gray73	grey25
cadetblue3	darkseagreen3	gray12	gray72	grey24
cadetblue2	darkseagreen2	gray11	gray71	grey23
cadetblue1	darkseagreen1	gray10	gray70	grey22
cadetblue	darkseagreen	gray9	gray69	grey21
burlywood4	darksalmon	gray8	gray68	grey20
burlywood3	darkred	gray7	gray67	grey19
burlywood2	darkorchid4	gray6	gray66	grey18
burlywood1	darkorchid3	gray5	gray65	grey17
burlywood	darkorchid2	gray4	gray64	grey16
brown4	darkorchid1	gray3	gray63	grey15
brown3	darkorchid	gray2	gray62	grey14
brown2	darkorange4	gray1	gray61	grey13
brown1	darkorange3	gray0	gray60	grey12
brown	darkorange2	gray	gray59	grey11
blueviolet	darkorange1	goldenrod4	gray58	grey10
blue4	darkorange	goldenrod3	gray57	grey9
blue3	darkolivegreen4	goldenrod2	gray56	grey8
blue2	darkolivegreen3	goldenrod1	gray55	grey7
blue1	darkolivegreen2	goldenrod	gray54	grey6
blue	darkolivegreen1	gold4	gray53	grey5
blanchedalmond	darkolivegreen	gold3	gray52	grey4
black	darkmagenta	gold2	gray51	grey3
bisque4	darkkhaki	gold1	gray50	grey2
bisque3	darkgrey	gold	gray49	grey1
bisque2	darkgreen	ghostwhite	gray48	grey0
bisque1	darkgray	gainsboro	gray47	grey
bisque	darkgoldenrod4	forestgreen	gray46	greenyellow
beige	darkgoldenrod3	floralwhite	gray45	green4
azure4	darkgoldenrod2	firebrick4	gray44	green3
azure3	darkgoldenrod1	firebrick3	gray43	green2
azure2	darkgoldenrod	firebrick2	gray42	green1
azure1	darkcyan	firebrick1	gray41	green
azure	darkblue	firebrick	gray40	gray100
aquamarine4	cyan4	dodgerblue4	gray39	gray99
aquamarine3	cyan3	dodgerblue3	gray38	gray98
aquamarine2	cyan2	dodgerblue2	gray37	gray97
aquamarine1	cyan1	dodgerblue1	gray36	gray96
aquamarine	cyan	dodgerblue	gray35	gray95
antiquewhite4	cornsilk4	dimgrey	gray34	gray94
antiquewhite3	cornsilk3	dimgray	gray33	gray93
antiquewhite2	cornsilk2	deepskyblue4	gray32	gray92
antiquewhite1	cornsilk1	deepskyblue3	gray31	gray91
antiquewhite	cornsilk	deepskyblue2	gray30	gray90
aliceblue	cornflowerblue	deepskyblue1	gray29	gray89
white	coral4	deepskyblue	gray28	gray88

grey99	lightpink1	mistyrose1	pink4	slategray1	
grey98	lightpink	mistyrose	pink3	slategray	
grey97	lightgrey	mintcream	pink2	slateblue4	
grey96	lightgreen	midnightblue	pink1	slateblue3	yellowgreen
grey95	lightgray	mediumvioletred	pink	slateblue2	yellow4
grey94	lightgoldenrodyellow	mediumturquoise	peru	slateblue1	yellow3
grey93	lightgoldenrod4	mediumspringgreen	peachpuff4	slateblue	yellow2
grey92	lightgoldenrod3	mediumslateblue	peachpuff3	skyblue4	yellow1
grey91	lightgoldenrod2	mediumseagreen	peachpuff2	skyblue3	yellow
grey90	lightgoldenrod1	mediumpurple4	peachpuff1	skyblue2	whitesmoke
grey89	lightgoldenrod	mediumpurple3	peachpuff	skyblue1	wheat4
grey88	lightcyan4	mediumpurple2	papayawhip	skyblue	wheat3
grey87	lightcyan3	mediumpurple1	palevioletred4	sienna4	wheat2
grey86	lightcyan2	mediumpurple	palevioletred3	sienna3	wheat1
grey85	lightcyan1	mediumorchid4	palevioletred2	sienna2	wheat
grey84	lightcyan	mediumorchid3	palevioletred1	sienna1	violetred4
grey83	lightcoral	mediumorchid2	palevioletred	sienna	violetred3
grey82	lightblue4	mediumorchid1	paleturquoise4	seashell4	violetred2
grey81	lightblue3	mediumorchid	paleturquoise3	seashell3	violetred1
grey80	lightblue2	mediumblue	paleturquoise2	seashell2	violetred
grey79	lightblue1	mediumaquamarine	paleturquoise1	seashell1	violet
grey78	lightblue	maroon4	paleturquoise	seashell	turquoise4
grey77	lemonchiffon4	maroon3	palegreen4	seagreen4	turquoise3
grey76	lemonchiffon3	maroon2	palegreen3	seagreen3	turquoise2
grey75	lemonchiffon2	maroon1	palegreen2	seagreen2	turquoise1
grey74	lemonchiffon1	maroon	palegreen1	seagreen1	turquoise
grey73	lemonchiffon	magenta4	palegreen	seagreen	tomato4
grey72	lawngreen	magenta3	palegoldenrod	sandybrown	tomato3
grey71	lavenderblush4	magenta2	orchid4	salmon4	tomato2
grey70	lavenderblush3	magenta1	orchid3	salmon3	tomato1
grey69	lavenderblush2	magenta	orchid2	salmon2	tomato
grey68	lavenderblush1	linen	orchid1	salmon1	thistle4
grey67	lavenderblush	limegreen	orchid	salmon	thistle3
grey66	lavender	lightyellow4	orangered4	saddlebrown	thistle2
grey65	khaki4	lightyellow3	orangered3	royalblue4	thistle1
grey64	khaki3	lightyellow2	orangered2	royalblue3	thistle
grey63	khaki2	lightyellow1	orangered1	royalblue2	tan4
grey62	khaki1	lightyellow	orangered	royalblue1	tan3
grey61	khaki	lightsteelblue4	orange4	royalblue	tan2
grey60	ivory4	lightsteelblue3	orange3	rosybrown4	tan1
grey59	ivory3	lightsteelblue2	orange2	rosybrown3	tan
grey58	ivory2	lightsteelblue1	orange1	rosybrown2	steelblue4
grey57	ivory1	lightsteelblue	orange	rosybrown1	steelblue3
grey56	ivory	lightslategray	olivedrab4	rosybrown	steelblue2
grey55	indianred4	lightslategray	olivedrab3	red4	steelblue1
grey54	indianred3	lightslateblue	olivedrab2	red3	steelblue
grey53	indianred2	lightskyblue4	olivedrab1	red2	springgreen4
grey52	indianred1	lightskyblue3	olivedrab	red1	springgreen3
grey51	indianred	lightskyblue2	oldlace	red	springgreen2
grey50	hotpink4	lightskyblue1	navyblue	purple4	springgreen1
grey49	hotpink3	lightskyblue	navy	purple3	springgreen
grey48	hotpink2	lightseagreen	navajowhite4	purple2	snow4
grey47	hotpink1	lightsalmon4	navajowhite3	purple1	snow3
grey46	hotpink	lightsalmon3	navajowhite2	purple	snow2
grey45	honeydew4	lightsalmon2	navajowhite1	powderblue	snow1
grey44	honeydew3	lightsalmon1	navajowhite	plum4	snow
grey43	honeydew2	lightsalmon	moccasin	plum3	slategray
grey42	honeydew1	lightpink4	mistyrose4	plum2	slategray4
grey41	honeydew	lightpink3	mistyrose3	plum1	slategray3
grey40	grey100	lightpink2	mistyrose2	plum	slategray2

ORGANIZAÇÃO | ORGANIZACIÓN

