Introdução ao R – 2011
Roteiro da aula 4 *

Alexandre Rademaker

26 de Janeiro de 2011

1 Strings

Completando o material dos slides, em sala trabalhamos em um exemplo de uso das funções que lidam com strings. Vimos o caso de recuperação de dados de cotações do site da UOL. [1].

Pensando no desenvolvimento de uma função em R para recuperação de dados do site da UOL, o primeiro passo seria pensar em uma forma de construir a URL onde os parâmetros da consulta são passados como argumentos. Um exemplo desta url está na tabela 1.

http://cotacoes.economia.uol.com.br/acao/cotacoes-historicas.html?codigo=PETR4.SA&beginDay=1&beginMonth=1&beginYear=2008&endDay=9&endMonth=11&endYear=2010&size=200&page=1

Tabela 1: Exemplo de url com parâmetros

No trecho abaixo, desconstruimos os parâmetros da url da tabela 1. Consulte a documentação das funções `unlist`, `strsplit` para detalhes.

```
> p <- unlist(strsplit(strsplit(strsplit(url, "?",
+     fixed = TRUE)[[1]][2], "&")[[1]], "="))
> p

 [1] "codigo"     "PETR4.SA"   "beginDay"   "1"
 [5] "beginMonth" "1"          "beginYear"  "2008"
 [9] "endDay"     "9"          "endMonth"   "11"
[13] "endYear"    "2010"       "size"       "200"
[17] "page"       "1"
```

Os nomes e valores dos parâmetros agora podem ser facilmente separados:

```
> p[1:length(p)%%2 == 0]

[1] "PETR4.SA" "1"        "1"        "2008"     "9"
[6] "11"       "2010"     "200"      "1"

> p[1:length(p)%%2 != 0]

[1] "codigo"     "beginDay"   "beginMonth" "beginYear"
[5] "endDay"     "endMonth"   "endYear"    "size"
[9] "page"
```

No trecho seguinte, realizamos a operação inversa. A partir de uma variável com o caminho base do serviço, e de um vetor com nomes para cada posição:

*Adaptado de [2].

[1] `http://economia.uol.com.br/cotacoes/`

```
> base <- "http://cotacoes.economia.uol.com.br/acao/cotacoes-historicas.html"
> params <- c(codigo = "PETR4.SA", beginDay = 1,
+     beginMonth = 1, beginYear = 2008, endDay = 9,
+     endMonth = 11, endYear = 2010, size = 200,
+     page = 1)
```

A url completa para a requisição é facilmente construída:

```
> url <- paste(base, paste(apply(cbind(names(params),
+     params), 1, paste, collapse = "="), collapse = "&"),
+     sep = "?")
```

2 Entrada de dados

Vimos várias formas de fazer a entrada de dados no R. Vide slides com links relevantes. A forma mais comum de entrada de dados é a recuperação de dados do Excel. Vimos que isto pode ser feito de duas formas, principalmente:

- No Excel, os dados podem ser salvos no formato `txt` ou `csv`. E estes arquivos importados no R.
- No Excel, podemos selecionar uma área qualquer, copiar para a "área de transferência" (clipboard) e "colar" os dados no R.

A operação de copiar/colar dados da área de transferência para o R é comentada em dois endereços:

- `https://stat.ethz.ch/pipermail/r-help/2005-February/066319.html`
- `http://www.johndcook.com/r_excel_clipboard.html`

No Windows, o comando seria:

```
> a <- read.table(file("clipboard"), sep = "\t",
+     dec = ",")
```

Em geral, a função mais usada para importação de dados para o R é o `read.table`, bastante flexível e com um grande variedade de argumentos que controlam seu comportamento.

```
> Squid <- read.table(file = "squidGSI.txt", header = TRUE)
> str(Squid, vec.len = 1)

'data.frame':        2644 obs. of  6 variables:
 $ Sample  : int  1 2 ...
 $ Year    : int  1 1 ...
 $ Month   : int  1 1 ...
 $ Location: int  1 3 ...
 $ Sex     : int  2 2 ...
 $ GSI     : num  10.4 ...

> names(Squid)

[1] "Sample"   "Year"     "Month"    "Location" "Sex"
[6] "GSI"
```

Vejam, por exemplo, o que aconteceria se o separador de casas decimais errado fosse informado. Observem o tipo da coluna `GSI`.

```
> Squid2 <- read.table(file = "squidGSI.txt", dec = ",",
+     header = TRUE)
> str(Squid2, vec.len = 1)
```

```
'data.frame':        2644 obs. of  6 variables:
 $ Sample  : int  1 2 ...
 $ Year    : int  1 1 ...
 $ Month   : int  1 1 ...
 $ Location: int  1 3 ...
 $ Sex     : int  2 2 ...
 $ GSI     : Factor w/ 2472 levels "0.0064","0.007",..: 1533 2466 ...
```

Outro exemplo que mostramos em sala foi o comportamento da função com espaços nos nomes das variáveis. Se o separador de campos não for informado para o arquivo `squidGSI1.txt`, a função `read.table` irá retornar um erro, pois na primeira linha ela encontrará um número diferente de valores que as demais, sete.

```
> Squid3 <- read.table(file = "squidGSI1.txt", dec = ",",
+     header = TRUE, sep = "\t")
```

Na documentação da função `read.table`, está claro que o separador de campos padrão é um ou mais espaços, tabulações e quebra de linhas. No comando acima, restringimos o separador para apenas tabulações, evitando que a string com espaço "GSI Test" seja entendida como dois valores.

Outro problema comum é a ausência de dados em algumas linhas. Duas coisas podem ocorrer neste caso: (1) o valor de uma dada variável (coluna do arquivo) em uma dada observação (linha do arquivo) pode ser omitido; ou/e (2) um valor diferente da string `NA` pode ser usado para sinalizar a ausência da informação.

No arquivo `squidGSI2.txt`, simulamos os dois casos. No comando abaixo, como mantivemos o argumento do separador, a função entendeu que na oitava observação faltava o valor para a variável `Year`. Mas vejam como a variável `Month` passou a ser lida como `Factor` pelo fato do valor `-` ter sido encontrado na décima-segunda observação.

```
> Squid4 <- read.table("squidGSI2.txt", head = TRUE,
+     sep = "\t")
> head(Squid4, 15)

   Sample Year Month Location Sex GSI.Test
1       1    1     1        1   2  10.4432
2       2    1     1        3   2   9.8331
3       3    1     1        1   2   9.7356
4       4    1     1        1   2   9.3107
5       5    1     1        1   2   8.9926
6       6    1     1        1   2   8.7707
7       7    1     1        1   2   8.2576
8       8   NA     1        3   2   7.4045
9       9    1     1        3   2   7.2156
10     10    1     2        1   2   6.8372
11     11    1     1        1   2   6.3882
12     12    1     -        1   2   6.3672
13     13    1     2        1   2   6.2998
14     14    1     1        1   2   6.0726
15     15    1     6        1   2   5.8395

> str(Squid4)

'data.frame':        2644 obs. of  6 variables:
 $ Sample  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Year    : int  1 1 1 1 1 1 1 NA 1 1 ...
 $ Month   : Factor w/ 13 levels "-","1","10","11",..: 2 2 2 2 2 2 2 2 2 6 ...
 $ Location: int  1 3 1 1 1 1 1 3 3 1 ...
 $ Sex     : int  2 2 2 2 2 2 2 2 2 2 ...
 $ GSI.Test: num  10.44 9.83 9.74 9.31 8.99 ...
```

Para contornar este problema, podemos usar o argumento `na.strings`, vide documentação.

```
> Squid5 <- read.table("squidGSI2.txt", head = TRUE,
+     sep = "\t", na.strings = "-")
> head(Squid5, 15)
```

```
   Sample Year Month Location Sex GSI.Test
1       1    1     1        1   2  10.4432
2       2    1     1        3   2   9.8331
3       3    1     1        1   2   9.7356
4       4    1     1        1   2   9.3107
5       5    1     1        1   2   8.9926
6       6    1     1        1   2   8.7707
7       7    1     1        1   2   8.2576
8       8   NA     1        3   2   7.4045
9       9    1     1        3   2   7.2156
10     10    1     2        1   2   6.8372
11     11    1     1        1   2   6.3882
12     12    1    NA        1   2   6.3672
13     13    1     2        1   2   6.2998
14     14    1     1        1   2   6.0726
15     15    1     6        1   2   5.8395
```

```
> str(Squid5)
```

```
'data.frame':        2644 obs. of  6 variables:
 $ Sample  : int  1 2 3 4 5 6 7 8 9 10 ...
 $ Year    : int  1 1 1 1 1 1 1 NA 1 1 ...
 $ Month   : int  1 1 1 1 1 1 1 1 1 2 ...
 $ Location: int  1 3 1 1 1 1 1 3 3 1 ...
 $ Sex     : int  2 2 2 2 2 2 2 2 2 2 ...
 $ GSI.Test: num  10.44 9.83 9.74 9.31 8.99 ...
```

Nos dois trechos acima, a função `head` é usada para mostrar as primeiras linhas dos data.frames.

A função `read.table` também pode ler diretamente arquivos compactados. No trecho a seguir, o primeiro caso é a leitura de arquivos zip que, por poderem conter mais de um arquivo, precisam ser lidos com auxílio da função `unz`. No segundo caso, arquivos Gzip [2], podem ser lidos diretamente pela `read.table`.

```
> dados1 <- read.table(unz("antarctic-birds.zip",
+     "antarctic-birds.txt"))
> dados2 <- read.table("antarctic-birds1.txt.gz")
```

Não apenas arquivos locais podem ser lidos, urls também podem ser passadas diretamente para a `read.table`.

```
> params <- c(key = "0AmWelpCNYLymdDFkbGdjLWhTTWxUd2hndWM4d2VvR3c",
+     hl = "en", single = "true", gid = "0", output = "csv")
> url <- paste("http://spreadsheets.google.com/pub",
+     paste(apply(cbind(names(params), params),
+         1, paste, collapse = "="), collapse = "&"),
+     sep = "?")
> d <- read.csv(url, header = FALSE)
> str(d, vec.len = 1)
```

```
'data.frame':        19 obs. of  12 variables:
 $ V1 : Factor w/ 19 levels "1/23/2011 15:35:26",..: 19 1 ...
 $ V2 : Factor w/ 6 levels "","2","3","4",..: 6 4 ...
 $ V3 : Factor w/ 5 levels "","2","3","5",..: 5 4 ...
 $ V4 : Factor w/ 6 levels "","1","3","4",..: 6 2 ...
 $ V5 : Factor w/ 5 levels "","3","4","5",..: 5 4 ...
```

[2] GNU Zip, http://www.gzip.org/

```
 $ V6 : Factor w/ 5 levels "","1","3","5",..: 5 1 ...
 $ V7 : Factor w/ 6 levels "","2","3","4",..: 6 4 ...
 $ V8 : Factor w/ 5 levels "","2","3","5",..: 5 4 ...
 $ V9 : Factor w/ 6 levels "","1","2","3",..: 6 4 ...
 $ V10: Factor w/ 6 levels "","2","3","4",..: 6 4 ...
 $ V11: Factor w/ 19 levels "andre.grego@gmail.com",..: 5 2 ...
 $ V12: Factor w/ 6 levels "","1","2","3",..: 6 3 ...
```

Neste caso, estamos lendo uma planilha no Google Docs. A planilha foi previamente compartilhada para que possa ser lida por qualquer pessoa que conheça a url e sem autenticação. Para saber mais sobre como interagir com o Google Docs, sugiro a leitura dos links:

- `http://bit.ly/haX197`
- `http://bit.ly/fo8uc1`
- `http://www.omegahat.org/RGoogleDocs/`

Se houver interesse, podemos falar mais sobre a importação de dados de bancos de dados relacionais como o Oracle, SQL Server ou MySQL. O pacote `RODBC` [1], usado para leitura de dados de BD relacionais é, no entanto, muito bem documentado.

3 Manipulando data.frames

Como uma revisão da aula anterior, verificamos novamente algumas formas de filtrar lihas e colunas de um data.frame:

```
> tmp <- Squid$Sex
> head(tmp)

[1] 2 2 2 2 2 2

> Sel <- Squid$Sex == 1
> SquidF <- Squid[Sel, ]
> SquidM <- Squid[Squid$Sex == 2, ]
```

De fato, filtros podem ser feitos com expressões booleanas complexas:

```
> a <- Squid[Squid$Location == 1 & Squid$Year ==
+     1, ]
> b <- Squid[Squid$Sex == 1 & (Squid$Location ==
+     1 | Squid$Location == 2), ]
```

A idéia de filtrar um data.frame a partir de expressões booleanas sobre os próprios componentes do data.frame é bastante flexível, mas requer cuidados. O trecho a seguir, o segundo filtro foi feito de forma errada, dado que foi feito sobre o data.frame `SquidF`, que tem quantidade de linhas diferentes do data.frame original.

```
> SquidF <- Squid[Squid$Sex == 1, ]
> nrow(SquidF)

[1] 1402

> nrow(Squid)

[1] 2644

> SquidF1 <- SquidF[Squid$Location == 1, ]
```

Uma função úteil para filtragem tanto de linhas quanto de colunas de um data.frame é a `subset`, consulte a documentação. A seguir, um pequeno exemplo, recuperamos as linhas do data.frame `Squid` onde a variável `Location` é igual a um. Além disso, recuperamos apenas as colunas `Sex`, `Location` e `Sample`.

```
> a <- subset(Squid, Location == 1, c("Sex", "Location",
+     "Sample"))
> head(a)

  Sex Location Sample
1   2        1      1
3   2        1      3
4   2        1      4
5   2        1      5
6   2        1      6
7   2        1      7
```

Também vimos na última aula como podemos ordenar valores de um vetor. No caso de data.frames, a ordenação dos valores de uma coluna pode ser usada para reordenar as linhas do data.frame:

```
> Ord1 <- order(Squid$GSI)
> tmp <- Squid[Ord1, ]
> head(tmp)

     Sample Year Month Location Sex    GSI
2283   2283    3     7        1   1 0.0064
2644   2644    4    12        1   1 0.0070
1519   1519    2     8        1   1 0.0072
2282   2282    3     7        1   1 0.0124
545     545    1     5        1   1 0.0131
2643   2643    4    10        1   1 0.0132
```

Obviamente, não precisamos da variável `Ord1`, poderíamos simplesmente ter digitado:

```
> tmp <- Squid[order(Squid$GSI), ]
> head(tmp)

     Sample Year Month Location Sex    GSI
2283   2283    3     7        1   1 0.0064
2644   2644    4    12        1   1 0.0070
1519   1519    2     8        1   1 0.0072
2282   2282    3     7        1   1 0.0124
545     545    1     5        1   1 0.0131
2643   2643    4    10        1   1 0.0132
```

4 Factors

Recordamos a importância de armazenar variáveis categórias como variávies do tipo `Factor`.

```
> Squid$fLocation <- factor(Squid$Location)
> Squid$fSex <- factor(Squid$Sex, levels = c(1,
+     2), labels = c("M", "F"))
```

Funções como `lm` e `boxplot` lidam de forma diferente com diferentes tipos de variáveis. Quando variáveis categóricas são armazenadas como números, são interpretadas como em um intervalo contínuo e não discreto.

A função `lm` também comporta-se de forma completamente diferente com variáveis categórias ou variáveis numéricas. No trecho a seguir, a função `lm` para cálculo da regressão linear é executa usando as variáveis categóricas.

```
> M1 <- lm(GSI ~ fSex + fLocation, data = Squid)
> summary(M1)
```

```
> boxplot(GSI ~ fSex, data = Squid)
```

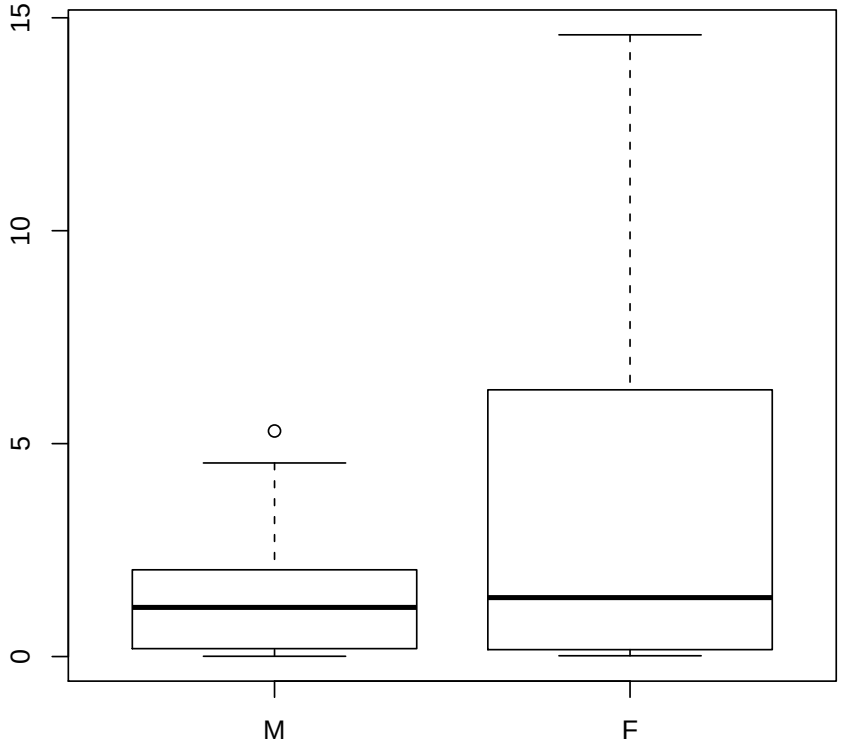

Figura 1: boxplot do data.frame Squid

```
Call:
lm(formula = GSI ~ fSex + fLocation, data = Squid)

Residuals:
    Min      1Q  Median      3Q     Max
-3.4137 -1.3195 -0.1593  1.2039 11.2159

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.35926    0.07068  19.230   <2e-16 ***
fSexF        2.02481    0.09427  21.479   <2e-16 ***
fLocation2  -1.85525    0.20027  -9.264   <2e-16 ***
fLocation3  -0.14248    0.12657  -1.126   0.2604
fLocation4   0.58756    0.34934   1.682   0.0927 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.415 on 2639 degrees of freedom
Multiple R-squared: 0.1759,        Adjusted R-squared: 0.1746
F-statistic: 140.8 on 4 and 2639 DF,  p-value: < 2.2e-16
```

No próximo trecho, para comparação, são usadas as variáveis numéricas.

```
> M2 <- lm(GSI ~ Sex + Location, data = Squid)
> summary(M2)

Call:
lm(formula = GSI ~ Sex + Location, data = Squid)

Residuals:
    Min      1Q  Median      3Q     Max
-3.2960 -1.2531 -0.1968  1.2743 11.2856

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.66743    0.17271  -3.864 0.000114 ***
Sex          2.04076    0.09569  21.327  < 2e-16 ***
Location    -0.09974    0.05720  -1.744 0.081312 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.454 on 2641 degrees of freedom
Multiple R-squared: 0.1487,        Adjusted R-squared: 0.148
F-statistic: 230.6 on 2 and 2641 DF,  p-value: < 2.2e-16
```

Variáveis categórias também podem ter seus níveis trocados de ordem:

```
> Squid$fLocation <- factor(Squid$Location, levels = c(2,
+     3, 1, 4))
> summary(Squid$fLocation)

   2    3    1    4
 157  447 1991   49
```

Referências

[1] Brian Ripley, , and from 1999 to Oct 2002 Michael Lapsley. *RODBC: ODBC Database Access*, 2010. R package version 1.3-2, http://CRAN.R-project.org/package=RODBC, http://cran.r-project.org/web/packages/RODBC/vignettes/RODBC.pdf.

[2] Ieno Zuur and Meesters. *A Beginner's Guide to R*. Springer, 2009.