Introdução ao R

Mineração de Dados - Laboratório 1

Thiago Ferreira Covões

O propósito deste primeiro laboratório é introduzir o ambiente computacional para estatística R que será usado ao longo da disciplina Mineração de Dados. Caso você queira se aprofundar, sugere-se o material disponível em https://cran.r-project.org/doc/manuals/r-release/R-intro.html, http://r4ds.had.co.nz/ e http://adv-r.had.co.nz/.

Este material foi baseado no material do Prof. Carlos da Silva dos Santos.

Uma sessão inicial com o R

Recomenda-se digitar todo os comandos apresentados a seguir. Linhas com fundo destacado indicam comandos para o interpretador R. Linhas que começam com ## ou *[k]*, onde k é algum número natural, indicam a saída impressa pelo interpretador.

Um tipo de dado fundamental do R é o *vetor*. Podemos criar um vetor pela concatenação de números, usando a função *c()*. Os comandos abaixo criam dois vetores numéricos, denominados x e y. A função *typeof()* imprime o tipo do objeto passado como parâmetro.

```
x <- c(1.1, 2.0, 3.3, 4.2, 5.1)
x
```

```
## [1] 1.1 2.0 3.3 4.2 5.1
```

```
y <- c(-2, 0.4, 2.1, -3, 2)
y
```

```
## [1] -2.0  0.4  2.1 -3.0  2.0
```

```
z <- 1L
w <- c("hello", "world") #não podia faltar..
length(x)
```

```
## [1] 5
```

```
length(y)
```

```
## [1] 5
```

```
length(z)
```

```
## [1] 1
```

```
length(w)
```

```
## [1] 2
```

```
typeof(x)
```

```
## [1] "double"
```

```
typeof(y)
```

```
## [1] "double"
```

```
typeof(z)
```

```
## [1] "integer"
```

```
typeof(w)
```

```
## [1] "character"
```

Os dois valores lógicos são definidos pelo R como *TRUE* e *FALSE*, podendo ser abreviados como *T* e *F*, respectivamente. Os operadores relacionais usuais, como $<$, $>$, $<=$, $>=$, $==$, podem ser aplicados a vetores inteiros:

```
z <- c (0, 2, 3, 3.3, 4, 4.7, 5.2)
z
```

```
## [1] 0.0 2.0 3.0 3.3 4.0 4.7 5.2
```

```
z < 3
```

```
## [1]  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE
```

```
z <= 3
```

```
## [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE
```

```
z > 4
```

```
## [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
```

```
z >= 4
```

```
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

```
z == 2
```

```
## [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
```

Para criar sequências de números inteiros, usamos a notação *n:m*, que também pode ser usada para criar sequências decrescentes:

```
xx <- 1:8
xx
```

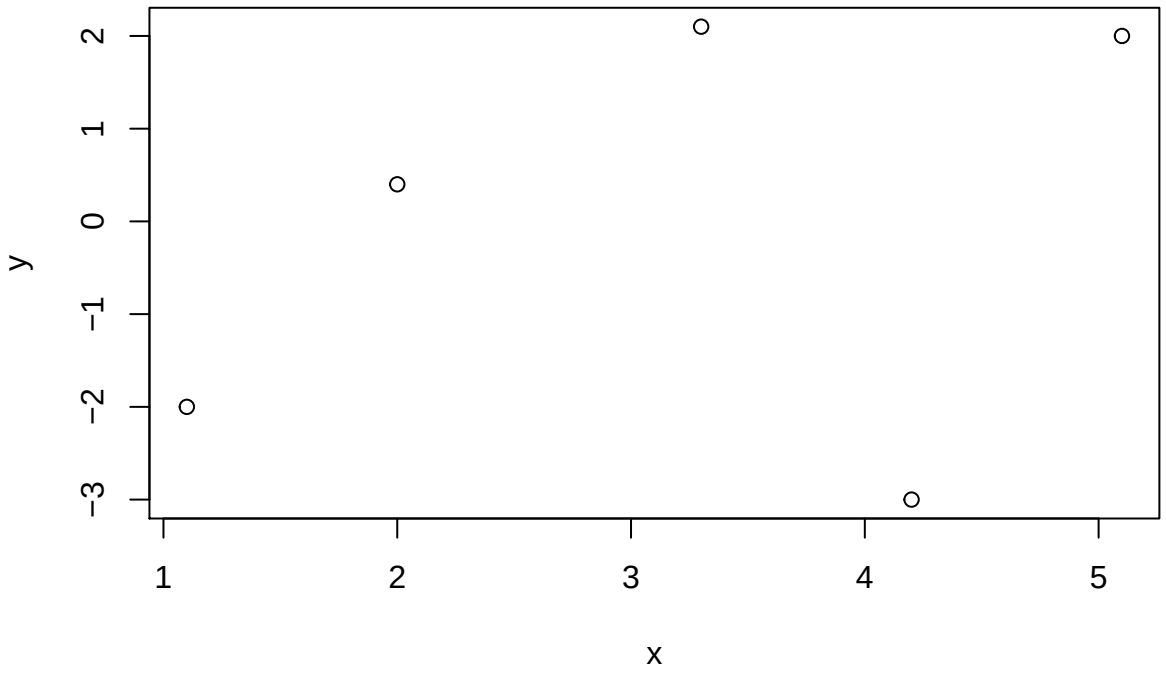
```
## [1] 1 2 3 4 5 6 7 8
```

```
yy <- 5:0
yy
```

```
## [1] 5 4 3 2 1 0
```

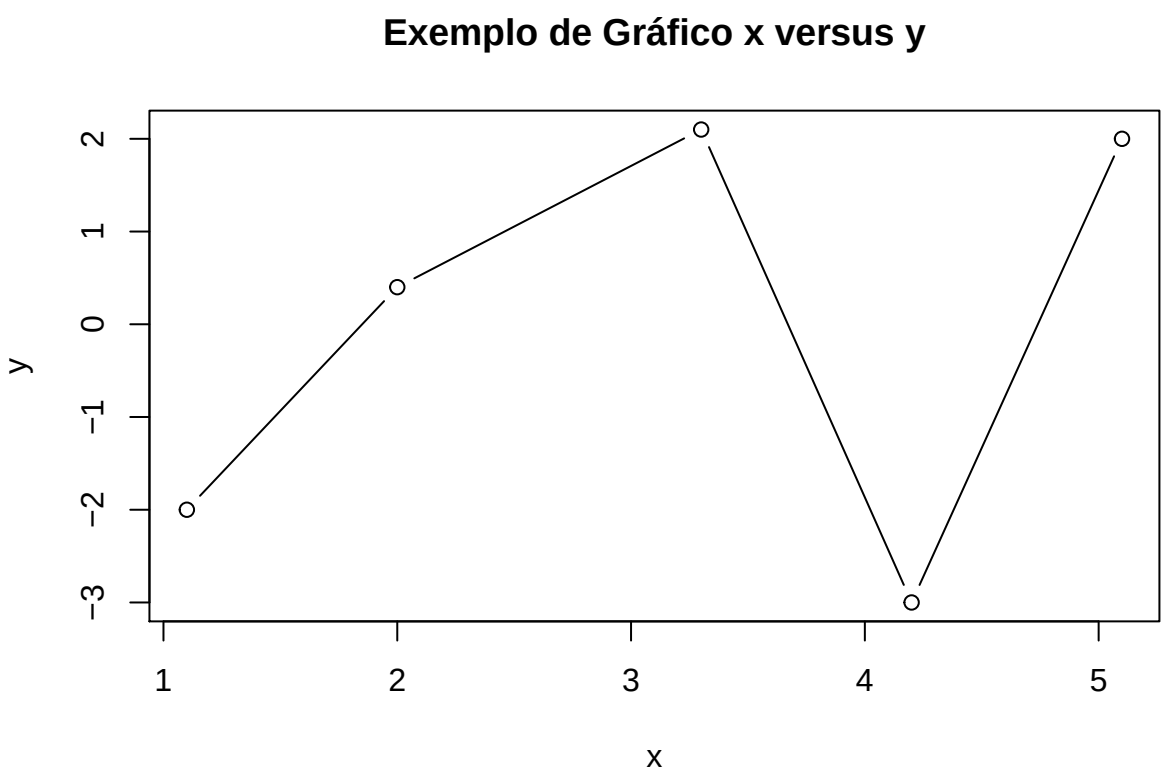
Para fazer um gráfico com os vetores criados, usamos o comando *plot*:

```
plot(x, y)
```



Para obter mais controle sobre a saída do gráfico, utilizamos os parâmetros *xlab* e *ylab* para colocar rótulos nos eixos e a função *title* para dar um nome ao gráfico. O significado dos parâmetros pode ser consultado na ajuda do programa. Para obter ajuda para uma determinada funçao, só é necessário digitar o sinal de interrogação seguido pelo nome da função, como em *?plot*.

```
plot(x, y, type="b", xlab="x", ylab="y")
title("Exemplo de Gráfico x versus y")
```



Os vetores são indexados usando colchetes ({[]}). No R, a primeira posição de um vetor tem índice 1. A

notação *n:m* permite gerar *fatias* de um vetor, selecionando os elementos das posições n até m (inclusive).

```
x
```

```
## [1] 1.1 2.0 3.3 4.2 5.1
```

```
x[1]
```

```
## [1] 1.1
```

```
x[3]
```

```
## [1] 3.3
```

```
x[2:4]
```

```
## [1] 2.0 3.3 4.2
```

Em conjunto com a indexação, os operadores relacionais permitem selecionar subconjuntos de vetores:

```
z
```

```
## [1] 0.0 2.0 3.0 3.3 4.0 4.7 5.2
```

```
z[z < 3]
```

```
## [1] 0 2
```

```
z[z >= 4]
```

```
## [1] 4.0 4.7 5.2
```

Funções podem ser aplicadas a vetores inteiros (elemento a elemento), tais como *ceiling* (arredondamento para cima) e *sin* (seno):

```
ceiling(x)
```

```
## [1] 2 2 4 5 6
```

```
sin(x)
```

```
## [1]  0.8912074  0.9092974 -0.1577457 -0.8715758 -0.9258147
```

Além disso, diversas funções úteis já vem implementadas de forma eficiente:

```
mean(x)
```

```
## [1] 3.14
```

```
median(x)
```

```
## [1] 3.3
```

```
sum(x)
```

```
## [1] 15.7
```

```
order(x)
```

```
## [1] 1 2 3 4 5
```

```
sort(x, decreasing = TRUE)
```

```
## [1] 5.1 4.2 3.3 2.0 1.1
```

O nome das funções são em geral informativos, por exemplo, você consegue dizer qual o propósito das funções *is.integer*, *as.numeric*, *as.character*? Fácil? E essas duas: *paste* e *paste0*?

Uma característica importante de funções no R é o uso de **parâmetros nomeados**. Por exemplo:

```r
mean(c(1:10, NA), trim = 0.1, na.rm = TRUE)
```

```
## [1] 5.5
```

Isso torna o código mais fácil de ler e torna desnecessário se preocupar com a ordem dos parâmetros. Mas, alguns cuidados tem que ser tomados. Qual a razão da resposta abaixo ser diferente?

```r
mean(c(1:10, NA), trim = 0.1, na.r = TRUE)
```

```
## [1] 5.5
```

```r
mean(c(1:10, NA), trim = 0.1, na_rm = TRUE)
```

```
## [1] NA
```

Valores *NA* (valores ausentes) são importantes quando trabalhamos com dados reais. No R, no começo, o tratamento que eles recebem pode não ser intuitivo, então uma atenção extra é necessária:

```r
x <- c(1,2,NA,3,4,NA)
x[x>2]
```

```
## [1] NA  3  4 NA
```

Os objetos presentes na área de trabalho (*workspace*) podem ser listados com a função *ls()* e removidos com a função *rm()*:

```r
ls()
```

```
## [1] "w"  "x"  "xx" "y"  "yy" "z"
```

```r
rm(x, y)
ls()
```

```
## [1] "w"  "xx" "yy" "z"
```

Exercício

Agora que você já sabe as principais características do R e como buscar informações, descubra o que o código abaixo está fazendo:

```r
z <- factor(c("ciclano", "fulano", "beltrano", "fulano","ciclano", "fulano"))
t <- table(z)
which.max(t)
```

Outros tipos de dados

Os outros três principais tipos de dados no R são: Matrizes, Listas e Data Frames. Uma função útil para verificar as principais características estruturais de uma variável é a *str*.

Matrizes

Uma *matrix* é um array com duas dimensões em que todos os elementos possuem o mesmo **tipo**, pode ser criada da seguinte forma:

```r
A <- matrix(1:4, nrow = 2)
B <- array(1:4, dim = c(2, 2))
stopifnot(identical(A,B))
```

Existem diversas funções para facilitar manipulações e operações com matrizes:

```
cbind(A,B)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    1    3
## [2,]    2    4    2    4
```

```
rbind(A, c(5, 6))
```

```
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
## [3,]    5    6
```

```
t(A)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
```

```
A * B
```

```
##      [,1] [,2]
## [1,]    1    9
## [2,]    4   16
```

```
A %*% B
```

```
##      [,1] [,2]
## [1,]    7   15
## [2,]   10   22
```

```
colnames(A) <- c("A1", "A2")
rownames(A) <- c("O1", "O2")
A
```

```
##    A1 A2
## O1  1  3
## O2  2  4
```

Listas

Uma lista é um conjunto de elementos **sem restrição** quanto ao tipo deles.

```
l <- list(c(1,2,3), "v1", list("v2","v3"))
l
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "v1"
##
## [[3]]
## [[3]][[1]]
## [1] "v2"
##
## [[3]][[2]]
```

```
## [1] "v3"
```

```
set.seed(123)
l2 <- list("loteria" = list("hoje" = list("numeros" = sample.int(60, 6), "valor" = 3.5),
                            "ontem"= list("numeros" = sample.int(60, 6), "valor" = 7)))
l2
```

```
## $loteria
## $loteria$hoje
## $loteria$hoje$numeros
## [1] 18 47 24 51 53  3
##
## $loteria$hoje$valor
## [1] 3.5
##
##
## $loteria$ontem
## $loteria$ontem$numeros
## [1] 32 53 60 27 54 25
##
## $loteria$ontem$valor
## [1] 7
```

O acesso a elementos da lista pode ser feito usando o operador *$* ou colchetes:

```
stopifnot(identical(l2$loteria$ontem, l2[["loteria"]][["ontem"]]))

intersect(l2$loteria$ontem$numeros, l2$loteria$hoje$numeros)
```

```
## [1] 53
```

```
union(l2$loteria$ontem$numeros, l2$loteria$hoje$numeros)
```

```
##  [1] 32 53 60 27 54 25 18 47 24 51  3
```

E se quisermos aplicar uma função a cada elemento de uma lista?

```
v1 <- list()
for(v in seq_along(l2$loteria)){
  nome <- names(l2$loteria)[v]
  v1[nome] <- paste(l2$loteria[[v]], collapse = " -- ")
}
v1
```

```
## $hoje
## [1] "c(18, 47, 24, 51, 53, 3) -- 3.5"
##
## $ontem
## [1] "c(32, 53, 60, 27, 54, 25) -- 7"
```

```
v2 <- lapply(l2$loteria, paste, collapse = " -- ")
stopifnot(identical(v1,v2))
```

Elementos de uma lista também podem ser obtidos usando apenas um colchetes, nesse caso temos uma lista como retorno **sempre**.

```
a <- list(1:3, "opa", list(c(4:2)))
a[1]
```

```
## [[1]]
```

```
## [1] 1 2 3
```

```
a[[1]]
```

```
## [1] 1 2 3
```

```
a[2][1]
```

```
## [[1]]
## [1] "opa"
```

```
a[2][[1]]
```

```
## [1] "opa"
```

Data Frames

Data.frames serão a estrutura que iremos utilizar com maior frequência. Um data.frame pode ser visto como uma lista de vetores do mesmo tamanhos. Portanto, é uma estrutura bi-dimensional. Cada elemento da lista é uma coluna do data.frame. A vantagem em relação a matrizes é que um data.frame permite termos colunas de diferentes tipos (os elementos de uma lista podem ser distintos, mas de um vetor devem ser iguais, logo, uma coluna terá todos os valores do mesmo tipo).

```
df <- data.frame(V1 = c(1,2,3,4,5),
                 V2 = c("fulano","ciclano","beltrano","fulano", "ciclano"))
df
```

```
##   V1       V2
## 1  1   fulano
## 2  2  ciclano
## 3  3 beltrano
## 4  4   fulano
## 5  5  ciclano
```

```
df$V1
```

```
## [1] 1 2 3 4 5
```

```
df$V2[3]
```

```
## [1] beltrano
## Levels: beltrano ciclano fulano
```

```
df[3,]
```

```
##   V1       V2
## 3  3 beltrano
```

```
df[3,2]
```

```
## [1] beltrano
## Levels: beltrano ciclano fulano
```

```
df[3,"V2"]
```

```
## [1] beltrano
## Levels: beltrano ciclano fulano
```

Estatística Descritiva

Nesta seção, veremos algumas medidas que tentam resumir um conjunto de dados, denominadas *estatísticas descritivas*. O nosso objetivo é obter rapidamente um quadro geral de um conjunto de dados que estiver sendo analisado.

Embora possamos criar um data.frame manualmente, o mais usual é carregar um conjunto de dados a partir de alguma representação em arquivo. O R já provê uma série de conjuntos de dados, que podem ser carregados com o comando *data*. No presente caso, nós vamos carregar o conjunto de dados *iris*.

```
data(iris)
names(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
```

```
iris[1, 1]
```

```
## [1] 5.1
```

O conjunto de dados iris é carregado, dando origem a um dataframe de mesmo nome. O comando *names* mostra os nomes das colunas do conjunto de dados, isto é, seus atributos. Podemos imprimir as primeiras linhas do data.frame:

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Para obter um resumo do data.frame usamos a funçao *summary*:

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

Para atributos numéricos, a função *summary* imprime os valores mínimo e máximo encontrados no conjunto de dados, a média e a mediana. Além disso, imprime também o primeiro e o terceiro *quartis*. O primeiro quartil de um atributo é o valor $q1$ tal que $1/4$ das instâncias têm valor menor que $q1$ para aquele atributo. O terceiro quartil é o valor $q3$ tal que $3/4$ das instâncias têm valor menor que $q3$ para aquele atributo.

Para atributos *categóricos* (i.e. aqueles que têm valores discretos, não ordenados), a função *summary* imprime

a *frequência* de cada valor do atributo, ou seja, o número de instâncias do conjunto de dados que têm aquele valor de atributo.

Os valores impressos pela função *summary* nos permitem ter uma ideia da distribuição dos dados. Outra medida importante para compreendermos o espalhamento dos dados é a *variância*. A variância de um atributo x é dada pela seguinte fórmula:

$$variancia(x) = s_x^2 = \frac{1}{m-1}\sum_{i=1}^{m}(x_i - \bar{x})^2$$

onde $\bar{x}$ é o valor médio de x. No R, calculamos a variância com a função *var()*. Podemos calcular a variância de cada atributo do conjunto de dados usando a notação *data.frame$nomeAtributo*, que referencia uma coluna de atributos do dataframe:

```
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4
##  [18] 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5
##  [35] 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0
##  [52] 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8
##  [69] 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4
##  [86] 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8
## [103] 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7
## [120] 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7
## [137] 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

```
var(iris$Sepal.Length)
```

```
## [1] 0.6856935
```

Já vimos como computar média e mediana. Mas, muitas vezes queremos esses valores dentro de alguns grupos. Por exemplo, e se quisermos computar a média do comprimento de sépala para cada classe?

```
sprintf("Média Geral: %.3f", mean(iris$Sepal.Length))
```

```
## [1] "Média Geral: 5.843"
```

```
idx_setosa <- iris$Species == "setosa"
sprintf("Média setosa: %.3f", mean(iris$Sepal.Length[idx_setosa]))
```

```
## [1] "Média setosa: 5.006"
```

Poderíamos fazer espécie por espécie, mas tem jeitos mais fáceis (em R, normalmente, tem mais de um jeito de fazer algo):

```
tapply(iris$Sepal.Length, iris$Species, mean)
```

```
##     setosa versicolor  virginica
##      5.006      5.936      6.588
```

```
aggregate(Sepal.Length ~ Species, data = iris, mean)
```

```
##      Species Sepal.Length
## 1     setosa        5.006
## 2 versicolor        5.936
## 3  virginica        6.588
```

```
aggregate(Sepal.Length ~ Species, data = iris, function(x) c(min = min(x), max = max(x)))
```

```
##      Species Sepal.Length.min Sepal.Length.max
## 1     setosa              4.3              5.8
```

```
## 2 versicolor              4.9             7.0
## 3  virginica              4.9             7.9
```

```
aggregate(. ~ Species, data = iris, function(x) c(min = min(x), max = max(x)))
```

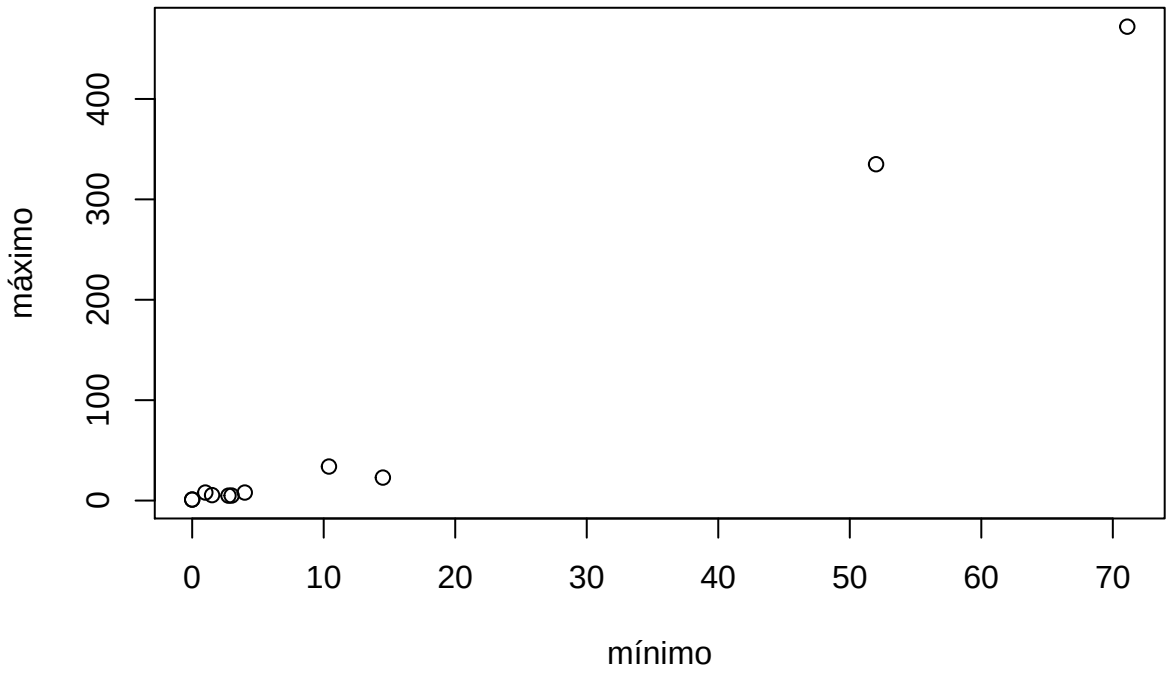
```
##      Species Sepal.Length.min Sepal.Length.max Sepal.Width.min
## 1     setosa              4.3             5.8            2.3
## 2 versicolor              4.9             7.0            2.0
## 3  virginica              4.9             7.9            2.2
##   Sepal.Width.max Petal.Length.min Petal.Length.max Petal.Width.min
## 1             4.4              1.0             1.9            0.1
## 2             3.4              3.0             5.1            1.0
## 3             3.8              4.5             6.9            1.4
##   Petal.Width.max
## 1             0.6
## 2             1.8
## 3             2.5
```

A notação $x \sim y$ é chamada de fórmula e é utilizada em diversas funções. Neste caso, estamos dizendo que queremos agrupar o comprimento da sépala pela espécie. O valor . significa todas as demais colunas cujo nome não aparecem no lado direito.

Exercícios

Resolva os exercícios considerando o dataset *mtcars* que está incluso no R (*?mtcars*).

1. Faça um gráfico de dispersão em que cada ponto deve corresponder a um atributo (coluna) e ter como coordenadas o seu valor mínimo e máximo (dica: *range* e *apply*). O gráfico deve ser igual a esse:



2. Quantas milhas por galão faz o carro automático mais econômico para os diferentes números de cilindros (dica: parâmetro *subset* da função *aggregate*).

3. Considerando apenas carros Mercedes e Toyota, compute o percentual de carros automáticos (dica: *rownames*, *grep*).