ME115 - Linguagem R

Atividade Prática 01 - Gabarito

1º semestre de 2023

Introdução

Operações exploradas:

- criação de vetores utilizando: `c()`, "`:`", `seq()` e `rep()`
- indexação (`[ ]`) de vetores através de: índice numérico, vetores numéricos, classes dos vetores, índices resultantes de operações lógicas (usamos o `<`)
- classe de objeto (comando `class()`)
- instalação (`install.packages()`) e carregamento (`library()`) de pacotes
- criação e exploração do objeto `data.frame`
- realização de gráfico usando o comando `plot()` usando escala original e logarítmica das variáveis

Observação: os exercícios 16 a 20 não foram feitos em aula, mas vocês podem tentar resolvê-los.

Exercícios

Os exercícios abaixo são do Capítulo 2 do Livro: https://rafalab.github.io/dsbook/r-basics.html

1. Qual é a soma dos 100 primeiros números inteiros positivos? Como você faria isso no R?

   Solução:

   ```
   x <- 1:100
   sum(x)
   ```

   ```
   ## [1] 5050
   ```

   Ou podemos fazer apenas

   ```
   sum(1:100)
   ```

   ```
   ## [1] 5050
   ```

   Podemos resolver também como uma progressão aritmética, com $a_1 = 1$ (o primeiro termo), $a_n = 100$ (o n-ésimo termo) $n = 100$ (número de elementos) e $r = 1$ (razão entre os elementos), então a soma dos n primeiros termos é dada por $S = \frac{n(a_1+a_n)}{2} = \frac{n(n+1)}{2}$.

   ```
   n <- 100
   n*(n + 1)/2
   ```

   ```
   ## [1] 5050
   ```

2. Agora faça a mesma coisa para calcular a soma dos números inteiros de 1 a 1000.

   Solução:

   ```
   x <- 1:1000
   sum(x)
   ```

   ```
   ## [1] 500500
   ```

3. Observe o resultado ao digitar o seguinte código em R:

```
n <- 1000
x <- seq(1, n)
sum(x)
```

Com base no resultado, o que você acha que as funções `seq` e `sum` fazem? Explore a ajuda do R.

Solução: A função `seq(1,n)` cria uma sequência numérica regular de 1 a n, enquanto a função `sum(x)` realiza o somatório do x, ou seja, realiza o somatório da sequência criada.

Observação:

```
seq(1, 10, length.out=21)
```

A função `seq` com o argumento `length.out=21` cria uma sequência numérica, com o tamanho da sequecia igual a 21.

Usando a progressão aritmética, sabemos que $a_n = a_{n-1} + r$ e $a_n = a_1 + (n-1) * r$, então $r = \frac{a_n - a_1}{n-1}$. No nosso caso, o código cria uma sequência de 1 a 10 de tamanho 21, com razão 0,45.

4. Use a função `c()` para criar um vetor com as médias das temperaturas máximas em janeiro nas cidades: Beijing, Lagos, Paris, Rio de Janeiro, San Juan e Toronto, que são, respectivamente, 35, 88, 42, 84, 81, and 30 graus Fahrenheit. Chame este objeto de `temp`.

Solução:

```
temp <- c(35, 88, 42, 84, 81, 30)
temp
```

```
## [1] 35 88 42 84 81 30
```

5. Crie um vetor com os nomes das cidades e chame esse objeto de `city`.

Solução:

```
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
          "San Juan", "Toronto")
city
```

```
## [1] "Beijing"        "Lagos"          "Paris"          "Rio de Janeiro"
## [5] "San Juan"       "Toronto"
```

6. Use a função `names()` e os objetos definidos nos itens anteriores para associar às temperaturas os nomes das cidades correspondentes.

Solução:

```
names(temp) <- city
temp
```

```
##        Beijing          Lagos          Paris Rio de Janeiro       San Juan
##             35             88             42             84             81
##        Toronto
##             30
```

7. Considerando o vetor `temp` que você criou:

   a) Use os operadores `[` and `:` para acessar as temperaturas das três primeiras cidades listadas.

   Solução:

   ```
   temp[1:3]
   ```

```
## Beijing   Lagos   Paris
##      35      88      42
```

b) Use o operador `[` operator para acessar as temperaturas de Paris e San Juan.

Solução:

```
temp[c("Paris","San Juan")]
```

```
##    Paris San Juan
##       42       81
```

8. Use o operador `:` para criar a sequência de números: 12, 13, 14, . . . , 73.

Solução:

```
12:73
```

```
##  [1] 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
## [26] 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61
## [51] 62 63 64 65 66 67 68 69 70 71 72 73
```

9. Crie um vetor contendo todos os números ímpares positivos e menores do que 100.

Solução:

```
seq(1,100, by=2)
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49
## [26] 51 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

10. Crie um vetor de números que inicia em 6, não ultrapassa 55 e aumenta em incrementos de 4/7: $6, 6 + 4/7, 6 + 8/7$, etc. Quantos elementos estão neste vetor? Dica: estude a função `seq()` e seus argumentos.

Solução:

```
seq(6, 55, by=4/7)
```

```
##  [1]  6.000000  6.571429  7.142857  7.714286  8.285714  8.857143  9.428571
##  [8] 10.000000 10.571429 11.142857 11.714286 12.285714 12.857143 13.428571
## [15] 14.000000 14.571429 15.142857 15.714286 16.285714 16.857143 17.428571
## [22] 18.000000 18.571429 19.142857 19.714286 20.285714 20.857143 21.428571
## [29] 22.000000 22.571429 23.142857 23.714286 24.285714 24.857143 25.428571
## [36] 26.000000 26.571429 27.142857 27.714286 28.285714 28.857143 29.428571
## [43] 30.000000 30.571429 31.142857 31.714286 32.285714 32.857143 33.428571
## [50] 34.000000 34.571429 35.142857 35.714286 36.285714 36.857143 37.428571
## [57] 38.000000 38.571429 39.142857 39.714286 40.285714 40.857143 41.428571
## [64] 42.000000 42.571429 43.142857 43.714286 44.285714 44.857143 45.428571
## [71] 46.000000 46.571429 47.142857 47.714286 48.285714 48.857143 49.428571
## [78] 50.000000 50.571429 51.142857 51.714286 52.285714 52.857143 53.428571
## [85] 54.000000 54.571429
```

```
length(seq(6, 55, by=4/7) )
```

```
## [1] 86
```

11. Qual é a classe dos seguintes objetos: `a <- seq(1, 10, 0.5)` e `b <- seq(1, 10)`?

Solução:

```
a <- seq(1, 10, 0.5)
class(a)
```

```
## [1] "numeric"
```

```
b <- seq(1, 10)
class(b)
```

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

12. Defina o seguinte vetor: `x <- c("1", "3", "5")` e force que ele contenha números inteiros.

Solução:

```
x <- c("1", "3", "5")
class(x)
```

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

```
x <- as.integer(x)
class(x)
```

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

13. Crie a seguinte matriz:

$$A_{4\times5} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{bmatrix} = \begin{bmatrix} 1 & 5 & 9 & 101 & 105 \\ 2 & 6 & 10 & 102 & 106 \\ 3 & 7 & 11 & 103 & 107 \\ 4 & 8 & 12 & 104 & 108 \end{bmatrix}.$$

Solução:

```
A <- matrix(c(1:12,101:108), nrow=4, ncol=5)
A
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    5    9  101  105
## [2,]    2    6   10  102  106
## [3,]    3    7   11  103  107
## [4,]    4    8   12  104  108
```

14. Considerando a matriz que você criou no Exercício 13:

a) Use os operadores `[` e `:` para acessar os elementos a_{21}, a_{22}, a_{23}.

Solução:

```
A[2,1:3]
```

```
## [1]  2  6 10
```

b) Acesse os elementos da terceira coluna.

Solução:

```
 A[,3]
```

```
## [1]  9 10 11 12
```

c) Acesse os elementos da quarta linha.

Solução:

```
 A[4,]
```

```
## [1]   4   8  12 104 108
```

d) Altere o valor a_{11} por 20.

Solução:

```
A[1,1]<-20
A
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   20    5    9  101  105
## [2,]    2    6   10  102  106
## [3,]    3    7   11  103  107
## [4,]    4    8   12  104  108
```

15. Você pode criar um conjunto de dados usando a função `data.frame`. Exemplo:

```
temp <- c(35, 88, 42, 84, 81, 30)
city <- c("Beijing", "Lagos", "Paris", "Rio de Janeiro",
          "San Juan", "Toronto")
city_temps <- data.frame(name = city, temperature = temp)
```

Use o código acima para criar o conjunto de dados, mas adicione uma variável que indique a temperatura em Celsius.

Solução:

```
city_temps$tempC <- 5*(city_temps$temp - 32)/9
head(city_temps)
```

```
##             name temperature     tempC
## 1        Beijing          35  1.666667
## 2          Lagos          88 31.111111
## 3          Paris          42  5.555556
## 4 Rio de Janeiro          84 28.888889
## 5       San Juan          81 27.222222
## 6        Toronto          30 -1.111111
```

16. Carregue o pacote `dslabs`. Veja a ajuda desse pacote. Carregue também o conjunto de dados `murders`. Qual a classe de `murders`?

Solução:

```
library(dslabs)
data(murders)
class(murders)
```

```
## [1] "data.frame"
```

17. Use a função `str()` para explorar a estrutura do conjunto de dados `murders`.

Leia a seção [2.13] - Indexação (https://rafalab.github.io/dsbook/r-basics.html#indexing)

Calcule a taxa de mortes por arma de fogo por 100 mil habitantes para cada estado e guarde em um objeto com nome `murder_rate`. Use um operador lógico para criar um vetor lógico chamado `low` que indica quais estados possuem taxa abaixo de 1.

Solução:

```
str(murders)
```

```
## 'data.frame':    51 obs. of  5 variables:
##  $ state     : chr  "Alabama" "Alaska" "Arizona" "Arkansas" ...
##  $ abb       : chr  "AL" "AK" "AZ" "AR" ...
```

```
##  $ region    : Factor w/ 4 levels "Northeast","South",..: 2 4 4 2 4 4 1 2 2 2 ...
##  $ population: num  4779736 710231 6392017 2915918 37253956 ...
##  $ total     : num  135 19 232 93 1257 ...
```

Note que essa função trás algumas informações sobre a estrutura do nosso data.frame, como número de variáveis e observações, os nomes das colunas e o tipo delas.

```
murder_rate <- murders$total/murders$population*10^5
low <- murder_rate < 1
```

Extras

Os exercícios 18 a 25 não foram feitos em aula, mas vocês podem tentar resolvê-los.

18. Use os resultados do exercício anterior (exercício 17) e a função `which()` para determinar os índices (posições do vetor) de `murder_rate` que estão abaixo de 1.

    Solução:

    ```
    which(low)
    ```

    ```
    ##  [1] 12 13 16 20 24 30 35 38 42 45 46 51
    ```

19. Use os resultados do exercício anterior para listar os nomes dos estados com taxas de mortes por arma de fogo abaixo de 1 (por 100 mil habitantes).

    Solução:

    ```
    murders$state[low]
    ```

    ```
    ##  [1] "Hawaii"        "Idaho"         "Iowa"          "Maine"
    ##  [5] "Minnesota"     "New Hampshire" "North Dakota"  "Oregon"
    ##  [9] "South Dakota"  "Utah"          "Vermont"       "Wyoming"
    ```

    Ou podemos utilizar

    ```
     murders$state[which(low)]
    ```

    ```
    ##  [1] "Hawaii"        "Idaho"         "Iowa"          "Maine"
    ##  [5] "Minnesota"     "New Hampshire" "North Dakota"  "Oregon"
    ##  [9] "South Dakota"  "Utah"          "Vermont"       "Wyoming"
    ```

20. Liste os estados da região *"Northeast"* com taxas de mortes por arma de fogo abaixo de 1 (por 100 mil habitantes). Dica: use o vetor lógico `low` e o operador lógico `&`.

    Solução:

    ```
    murders$state[ murders$region == "Northeast" & low]
    ```

    ```
    ## [1] "Maine"         "New Hampshire" "Vermont"
    ```

21. Calcule a média das taxas de mortes por arma de fogo por 100 mil habitantes entre os estados.

    Solução:

    ```
    mean(murder_rate)
    ```

    ```
    ## [1] 2.779125
    ```

22. Quantos estados estão abaixo da média?

    Solução:

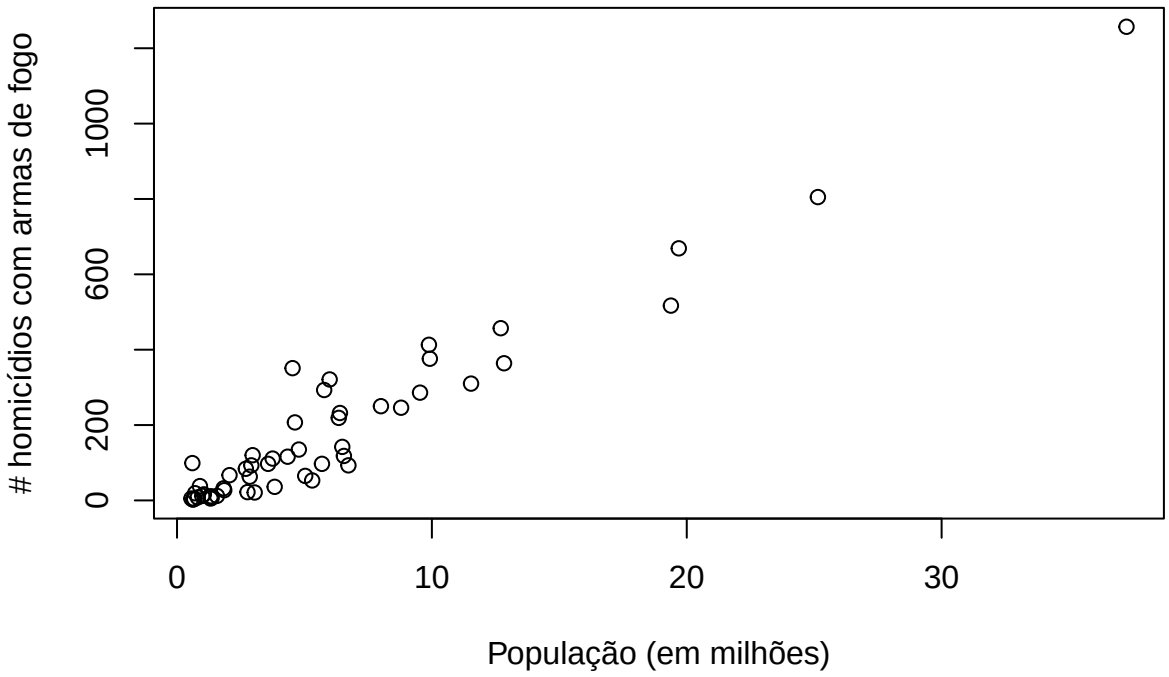
    ```
    low_mean <- murder_rate < mean(murder_rate)
    sum(low_mean)
    ```

```
## [1] 27
```

Leia a seção [2.15] - Gráficos básicos (https://rafalab.github.io/dsbook/r-basics.html#basic-plots)

23. O código a seguir produz um gráfico do total de mortes versus tamanho da população.

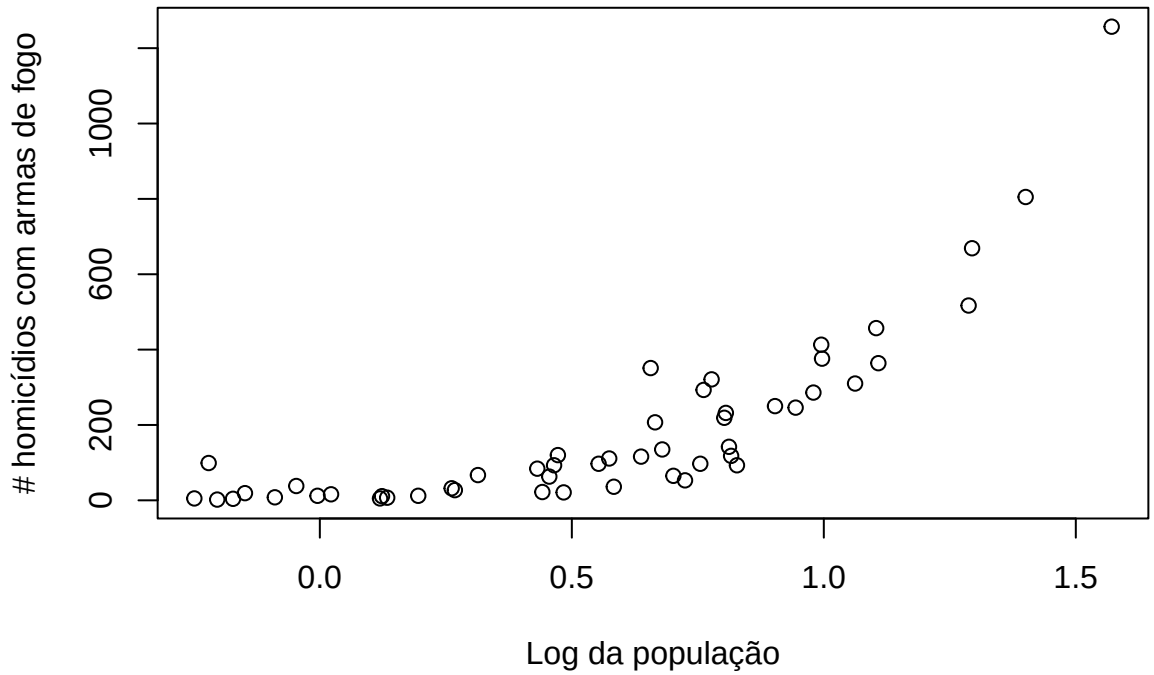
```
library(dslabs)
data(murders)

population_in_millions <- murders$population/10^6
total_gun_murders <- murders$total

plot(population_in_millions, total_gun_murders,
     ylab="# homicídios com armas de fogo",
     xlab="População (em milhões)")
```



Como muitos estados têm população abaixo de 5 milhões, o gráfico não mostra muito bem os dados. Refaça o gráfico usando a escala logarítmica. Transforme usando `log10()` e refaça o gráfico.

Solução:

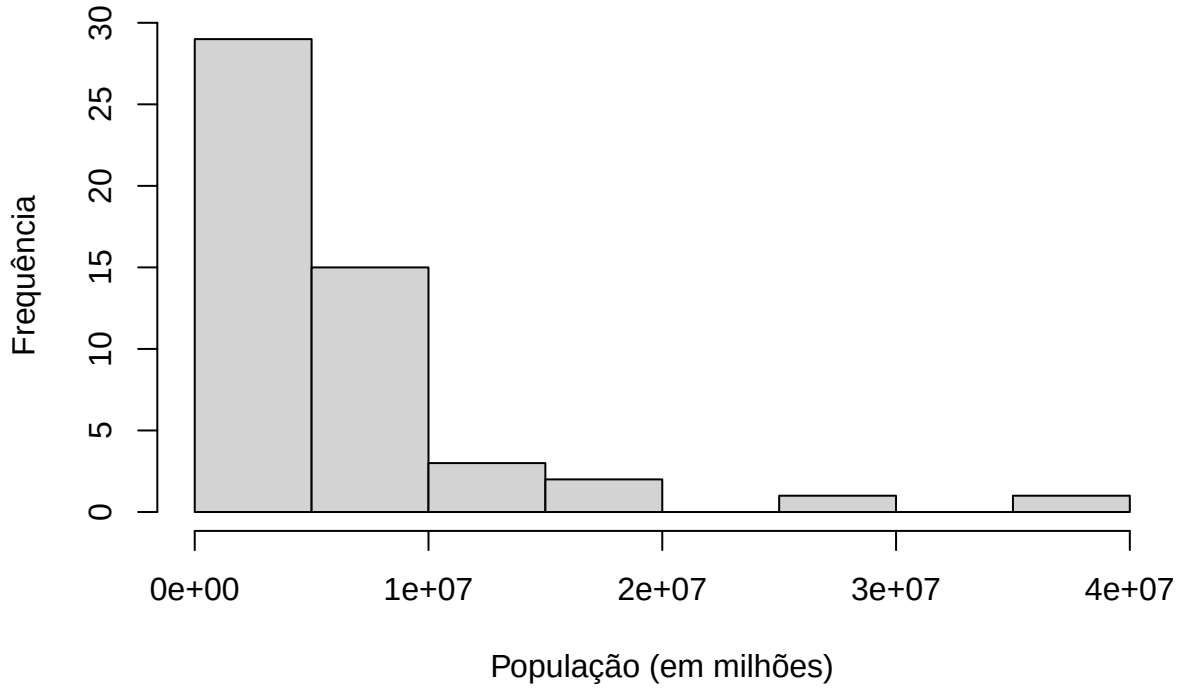
```
plot(log10(population_in_millions), total_gun_murders,
     ylab="# homicídios com armas de fogo",
     xlab="Log da população")
```



24. Apresente um histograma do total da população de cada estado. Dica: `hist()`.

Solução:

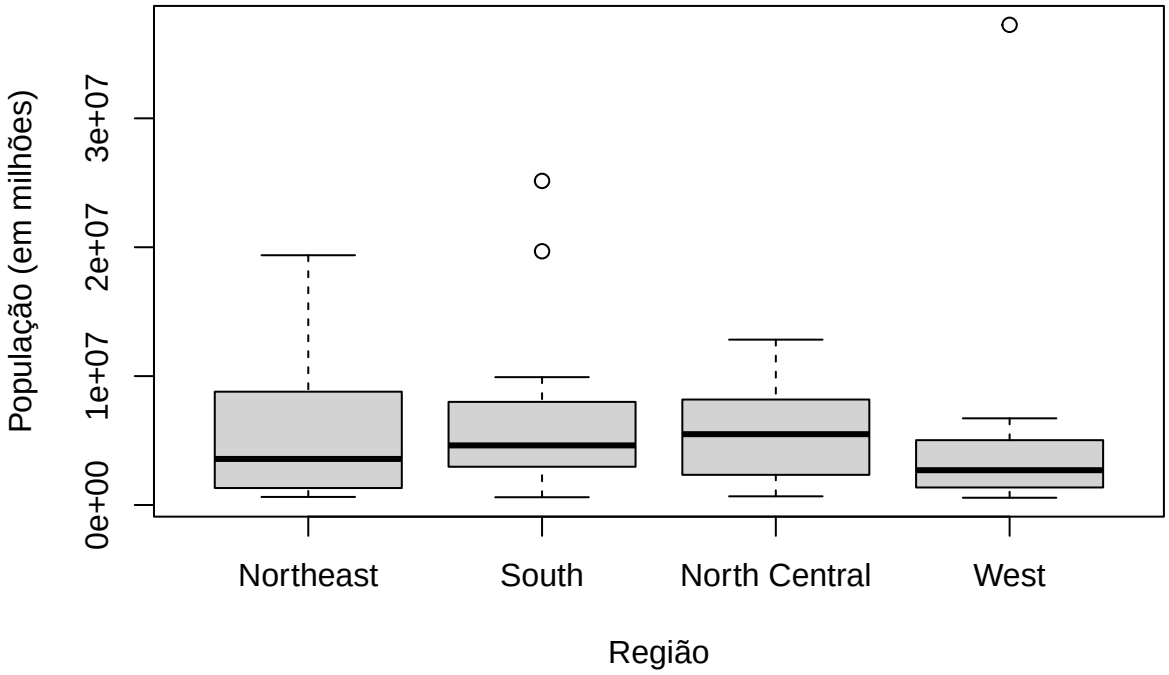
```
hist(murders$population,
     ylab="Frequência",
     xlab="População (em milhões)",
     main="")
```



25. Apresente boxplots do total da população de cada estado por região (cada boxplot representa uma região diferente).

Solução:

```
boxplot(population~region,data=murders,
     ylab="População (em milhões)",
     xlab="Região")
```



Agrecimentos

Esse material foi incrementado do material produzido pelas Profas. Samara Kiihl e Tatiana Benaglia para o curso de ME115.