Estruturas de Armazenamento e Gráficos - BÁSICO

Cleber Almeida Pereira

30/04/2021

Apresentação

A **linguagem R** é uma linguagem de programação utilizada para análises estatísticas e representação através de gráficos.

Este material foi desenvolvido utilizando o **RStudio** Version 1.3.1093, através da ferramenta **R Markdown**.

O **RStudio** facilita o uso da **linguagem R** através de uma interface gráfica para manipulação de dados, cálculos estatísticos e criação de gráficos.

O intuito é apresentar algumas estruturas de armazenamento e usos básicos destas estruturas de um modo que pode ser aplicado em *Data Science* nos processos de análises de dados e representação gráfica destes.

Estruturas de Armazenamento

Nesta seção apresentaremos:

- Vetor
- Fator
- Lista
- Data frame
- Matriz

Importante destacar que, diferente de outras linguagens de programação, em **R** o primeiro índice é **1** e não **0**.

1. Vetor

Todos os elementos devem ser do mesmo tipo.

Criaremos 2 vetores, sendo o primeiro com nome de cidades e o segundo com siglas de unidades da federação (UF) das cidades criadas.

a. declarar um Vetor

Como exemplo vamos declarar um vetor com nome de quatro cidades do estado de Mato Grosso do Sul (BRASIL):

```
cidades <- c("Campo Grande", "Corumbá", "Ladário", "Miranda")
cidades
```

```
## [1] "Campo Grande" "Corumbá"      "Ladário"      "Miranda"
```

Na saída do código (##) são exibidos os valores do nosso vetor com índice máximo igual a 4, ou seja, contendo 4 elementos.

b. acessar valor pelo índice

```
cidades[3]
```

```
## [1] "Ladário"
```

```
cidades[0]
```

```
## character(0)
```

Note que, ao utilizar índice **0** nos foi retornada a informação do tipo de dado armazenado, no caso tipo *character*.
Assim, ao declar um vetor com números (ex: c(1,2,3,4)), ao acessar o índice **0** será obtido como retorno o tipo *numeric*.

```
cidades[5]
```

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

No exemplo acima, foi utilizado índice 5, no caso um índice inexistente, sendo apresentado saída com valor **NA** para indicar que não há resposta disponível.
NA significa *not applicable* (não aplicável ou não se aplica), *not available* (não disponível) ou *no answer* (sem resposta).

c. atualizar o valor de um índice existente

```
cidades[4] <- "Aquidauana"
cidades
```

```
## [1] "Campo Grande" "Corumbá"      "Ladário"      "Aquidauana"
```

Ao exibir os valores do vetor podemos observar que foi alterado o nome da cidade do índice 4, que antes era "Miranda" e agora é "Aquidauana".

d. adicionar novo índice e atribuir um valor

```
cidades[5] <- "Miranda"
cidades
```

```
## [1] "Campo Grande" "Corumbá"      "Ladário"      "Aquidauana"   "Miranda"
```

Foi adicionado o município "Miranda" no índice 5.
É preciso saber o tamanho do vetor para evitar atribuir valor a um índice já existente. Outro erro que pode acontecer é informar um índice muito longe criando uma série de índices novos e fazendo com que o valor seja atribuído a todos os índices novos.
Utilizando o vetor **cidades** como exemplo, se ao invés de informar índice 5 o usuário tivéssemos informado índice 10, teríamos criados todos os índices inexistentes, ou seja, de 5 a 10 e atribuído o valor "Miranda" a cada um deles.

e. acessar um intervalo de valores

```
cidades[2:4]
```

```
## [1] "Corumbá"    "Ladário"    "Aquidauana"
```

Na saída estão exibidos os nomes dos municípios dos índices do intervalo 2 a 4.

f. omitir um elemento do vetor

```
cidades[-4]
```

```
## [1] "Campo Grande" "Corumbá"      "Ladário"      "Miranda"
```

Indicamos o índice que não queremos utilizando o sinal **-** antes do índice.
Observe que na saída não foi exibido o 4º elemento, "Aquidauana".

g. remover um elemento do vetor

```
cidades[6] <- "Whashington"
cidades
```

```
## [1] "Campo Grande" "Corumbá"      "Ladário"      "Aquidauana"   "Miranda"
## [6] "Whashington"
```

```
cidades <- cidades[-6]
cidades
```

```
## [1] "Campo Grande" "Corumbá"      "Ladário"      "Aquidauana"   "Miranda"
```

Primeiro foi atribuído o nome da cidade "Whashington" ao índice 6 e na saída foram exibidos todos os elementos do vetor.
Para remover este último nome inserido, vamos utilizar uma técnica bem simples.
Vamos atribuir o próprio vetor exceto o índice indicado com sinal negativo **-** como vimos anteriormente.

h. limpar o vetor

```
cidades <- NULL # Limpa o vetor
cidades
```

```
## NULL
```

Para limpar o vetor atribuímos o valor **NULL** (nulo).

i. declarando nosso segundo vetor (Sigla dos Estados)

Para as próximas estruturas precisaremos representar a sigla dos estados para cada município do vetor **cidades**, então vamos criar um vetor com as siglas:

```
estados.sigla <- c("MS", "MS", "MS", "MS", "MS", "MS")
estados.sigla
```

```
## [1] "MS" "MS" "MS" "MS" "MS" "MS"
```

Note que todos os estados são iguais.
Se utilizarmos este vetor não poderemos apresentar um bom exemplo de análise por unidades da federação.
Então vamos precisar criar cidades de outros estados brasileiros para podermos ter um melhor exemplo durante a análise.

Para isso adicionamos um conteúdo *bônus* logo a seguir.

Bônus - Funções

Vamos aproveitar a necessidade de utilizarmos outros municípios e siglas de unidades da federação e, ao invés de inserirmos novos municípios vamos alterar o conteúdo dos vetores **cidades** e **estados.sigla** utilizando funções.

Para isso vamos criar duas funções próprias para atribuir os novos valores.
São funções com exemplos simples com um certo teor de revisão.

Função: setar vetor *cidades*

```
set.vetor.cidades <- function(){
 vetor_aux <- c("Campo Grande", "Cuiabá", "Curitiba", "Aquidauana")
 vetor_aux[4] <- "São Paulo"
 vetor_aux[5] <- "Miranda"
 return(vetor_aux)
}
```

Acima foi declarada uma função que não recebe nenhum argumento.
Ela possui um vetor auxiliar que recebe os valores desejados e retorna este vetor quando solicitada.
Nela escrevemos três linhas de atribuição apena para "revisar" um pouco do que vimos até agora.
Observe que primeiro criamos um vetor com 4 cidades, depois alteramos o 4º elemento e, por fim atribuímos um 5º valor.

Utilizamos a função criada atribuindo o seu retorno (*vetor_aux*) ao vetor **cidades** chamando-a como a seguir:

```
cidades <- set.vetor.cidades()
cidades
```

```
## [1] "Campo Grande" "Cuiabá"       "Curitiba"     "São Paulo"    "Miranda"
```

Na saída apresentamos o novo conteúdo do vetor **cidades**.

Função: setar vetor *estados.sigla*

Agora precisamos alterar o conteúdo do vetor **estados.sigla** para que contenha os respectivos valores de UF para cada município do vetor **cidades**.
Para isso vamos criar uma função que altera aquele vetor.

```
set.vetor.estados.siglas <- function(){
 vetor_aux <- c("MS", "MT", "PR", "SP", "MS")
 return(vetor_aux)
}
```

Como temos cinco cidades atribuímos respectivamente as cinco siglas de seus respectivos Estados.

Assim como fizemos com o vetor **cidades** vamos chamar a função e alterar o conteúdo do vetor **estados.sigla**:

```
estados.sigla <- set.vetor.estados.siglas()
estados.sigla
```

```
## [1] "MS" "MT" "PR" "SP" "MS"
```

Pronto, agora temos o registro de cinco cidades e suas respectivas unidades da federação.

Mais adiante utilizaremos estes vetores na seção **Data frames**

---

2. Fator

Esta função é utilizada para iterar sobre um **vetor**.
Basicamente, ela possibilita expressar uma variável categórica.

Como exemplo vamos usar a função sobre um vetor com siglas das Unidades da Federação e criar uma variável **UF** para armazenar a informação de Unidade da Federação de .

a. declarar um Fator

Como exemplo vamos usar a função sobre um vetor com siglas das Unidades da Federação.

```
siglas <- c("SP", "GO", "RS", "DF", "GO", "SP", "MS", "MT", "SP", "MS")
UF <- as.factor(siglas)
is.factor(UF)
```

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

```
UF
```

```
##  [1] SP GO RS DF GO SP MS MT SP MS
## Levels: DF GO MS MT RS SP
```

```
UF[1]
```

```
## [1] SP
## Levels: DF GO MS MT RS SP
```

```
str(UF)
```

```
##  Factor w/ 6 levels "DF","GO","MS",..: 6 2 5 1 2 6 3 4 6 3
```

Primeiro criamos um vetor de siglas.
Na linha seguinte declaramos o fator **UF** utilizando a função *as.factor()*. Também poderíamos utilizar a função *factor()*.
Na terceira saída verificamos se **UF** é um fator.
Observe que na quarta saída os valores são apresentados em ordem alfabética, mas na terceira saída, quando indicamos o índice **1** o valor retornado é o original "SP" e não o primeiro valor ordenado "DF", portando não ocorre uma ordenação dos elementos, mas uma identificação da classificação de categorias.
Na última linha estão as informações sobre o fator.
Note que há uma representação ordinal para indicar a sequência dos registros lidos.
Assim, o número 1 refere-se a "DF" que foi a 1ª categoria identificada por ordem alfabética, o número 2 refere-se a "GO", o número 3 a "MS" e assim por diante, sendo apresentados na ordem de suas posições.

---

3. Lista

São estruturas muito úteis pois podem armazenar objetos de diversos tipos.
Deste modo, as listas podem ser criadas contendo objetos de outras classes.

a. criar uma lista de vários objetos

Neste exemplo vamos criar uma lista contendo 2 vetores e um fator criados anteriormente.

```
lista <- list(cidades, estados.sigla, UF)
lista
```

```
## [[1]]
## [1] "Campo Grande" "Cuiabá"       "Curitiba"     "São Paulo"    "Miranda"
##
## [[2]]
## [1] "MS" "MT" "PR" "SP" "MS"
##
## [[3]]
##  [1] SP GO RS DF GO SP MS MT SP MS
## Levels: DF GO MS MT RS SP
```

```
str(lista)
```

```
## List of 3
##  $ : chr [1:5] "Campo Grande" "Cuiabá" "Curitiba" "São Paulo" ...
##  $ : chr [1:5] "MS" "MT" "PR" "SP" ...
##  $ : Factor w/ 6 levels "DF","GO","MS",..: 6 2 5 1 2 6 3 4 6 3
```

Na última saída estão sendo exibidos os tipos de cada elemento desta lista.

b. acessar um elemento da lista

Vamos tentar acessar os elementos da lista que criamos acima.

```r
lista$cidades
```

```
## NULL
```

```r
lista$estados.sigla
```

```
## NULL
```

```r
lista$UF
```

```
## NULL
```

```r
lista[1]
```

```
## [[1]]
## [1] "Campo Grande" "Cuiabá"       "Curitiba"     "São Paulo"    "Miranda"
```

> Note que só foi possível acessar o valor através do índice da lista.
> O cifrão (**$**) é utilizado para indicar um elemento da lista, mas neste caso os objetos passados são atribuídos a cada índice.

Então vamos criar outra lista.

```r
pessoa <- list(nome="João",
               sexo="M",
               idade=42,
               cidade="Campo Grande")
pessoa
```

```
## $nome
## [1] "João"
##
## $sexo
## [1] "M"
##
## $idade
## [1] 42
##
## $cidade
## [1] "Campo Grande"
```

```r
str(pessoa)
```

```
## List of 4
##  $ nome  : chr "João"
##  $ sexo  : chr "M"
##  $ idade : num 42
##  $ cidade: chr "Campo Grande"
```

Esta é uma lista que contém informações sobre uma pessoa.
Na última saída estão sendo exibidas as informações sobre cada elemento.
A principal diferença é que estamos utilizando variáveis para indentificar cada elemento, com intuito de facilitar acessar seus valores.
Vamos tentar acessá-los.

```
pessoa$nome
```

```
## [1] "João"
```

```
pessoa[1]
```

```
## $nome
## [1] "João"
```

```
pessoa[[1]]
```

```
## [1] "João"
```

```
pessoa["nome"]
```

```
## $nome
## [1] "João"
```

```
pessoa[["nome"]]
```

```
## [1] "João"
```

Nos exemplos acima utilizamos três métodos diferentes:

- o **$** para indicar a variável identificadora;
- o **índice**;
- o **nome** da variável.

Note que somente quando utilizamos duplo colchetes [[ ]] conseguiremos acessar um valor diretamente.

c. filtar elementos de uma lista pela variável identificadora

```
pessoa[c('nome','idade')]
```

```
## $nome
## [1] "João"
##
## $idade
## [1] 42
```

```
pessoa[c('nome','idade')][["nome"]]
```

```
## [1] "João"
```

```
pessoa[c('nome','idade')][["idade"]]
```

```
## [1] 42
```

Nos exemplos acima utilizamos um vetor com os parâmetros de filtro, no caso os nomes das variáveis. De modo simples estamos dizendo "filtre por nome e idade".
No primeiro exemplo é obtido uma lista contendo apenas os parâmetros solicitados.
Nos exemplos seguintes é informado qual é o argumento desejado para se acessar o valor. Também poderíamos ter informado o índice.

d. lista de listas

Vamos criar uma lista de pessoas.
Para isso vamos criar mais duas pessoas:

```
pessoa2 <- list(nome="Maria",
              sexo="F",
              idade=22,
              cidade="São Paulo")
pessoa3 <- list(nome="Ana",
               sexo="F",
               idade=32,
               cidade="Curitiba")
```

Agora vamos criar uma lista de pessoas e por fim vamos listar apenas os nomes delas.

```
pessoas <- list(pessoa, pessoa2)
pessoas[[3]] <- pessoa3
for (p in pessoas){
 print(p[["nome"]])
}
```

```
## [1] "João"
## [1] "Maria"
## [1] "Ana"
```

Neste exemplo utilizamos o **for** para iterar na lista.
Embora seja uma possibilidade, ao invés de criar uma lista de listas, como a que criamos para pessoas, recomendo o uso do **Data frame**.

Então vamos criar uma lista de cidades com os vetores já criados anteriormente.

```
cidades.lista = list(cidade= cidades,
                     uf = estados.sigla)
cidades.lista
```

```
## $cidade
## [1] "Campo Grande" "Cuiabá"       "Curitiba"     "São Paulo"    "Miranda"
##
## $uf
## [1] "MS" "MT" "PR" "SP" "MS"
```

Vamos utilizar esta lista na seção sobre **Data frames**.

4. Data frame

O tipo **Data frame** pode ser considerado como a melhor forma para se armazenar dados para realização de uma análise.
Nele, cada linha é uma observação do conjunto de dados e cada coluna representa uma variável.

a. Criar um data frame com lista

```
df.cidades <- data.frame(cidades.lista)
df.cidades
```

```
##         cidade uf
## 1 Campo Grande MS
## 2       Cuiabá MT
## 3     Curitiba PR
## 4    São Paulo SP
## 5      Miranda MS
```

b. Criar um data frame com vetores

```
dfA <- data.frame(cidades, estados.sigla)
dfA
```

```
##        cidades estados.sigla
## 1 Campo Grande            MS
## 2       Cuiabá            MT
## 3     Curitiba            PR
## 4    São Paulo            SP
## 5      Miranda            MS
```

```
dfA[1]
```

```
##        cidades
## 1 Campo Grande
## 2       Cuiabá
## 3     Curitiba
## 4    São Paulo
## 5      Miranda
```

```
dfA[2]
```

```
##   estados.sigla
## 1            MS
## 2            MT
## 3            PR
## 4            SP
## 5            MS
```

A primeira saída apresenta o *Data frame* **dfA** com os valores dos vetores atribuídos. Nas demais saídas acessamos cada um dos vetores pela sua posição.

Outra forma de acessar os valores é pelo nome da coluna.

```
dfA$cidades
```

```
## [1] "Campo Grande" "Cuiabá"       "Curitiba"     "São Paulo"    "Miranda"
```

c. Filtrar valores do data frame por fatiamento simples

```
dfA[1,2]
```

```
## [1] "MS"
```

Obtemos o valor armazenado na linha **1** coluna **2**.

```
dfA[,1]
```

```
## [1] "Campo Grande" "Cuiabá"       "Curitiba"     "São Paulo"    "Miranda"
```

Obtemos todas as linhas da primeira coluna

```
dfA[1,]
```

```
##        cidades estados.sigla
## 1 Campo Grande            MS
```

Na saída estão apresentados os valores da primeira linha de todas as colunas

d. Filtrar valores do data frame por coordenadas com vedores

```
dfA[c(2:4), c(0,2)]
```

```
## [1] "MT" "PR" "SP"
```

No exemplo acima selecionamos os valores das linhas 2 a 4 apenas da última coluna, Tal exemplo é mais usual quando temos 3 ou mais colunas.

Então vamos alterar o Data Frame adicionando mais uma coluna.

e. Adicionar coluna

Primeiro vamos verificar os nomes das colunas.

```
names(dfA)
```

```
## [1] "cidades"       "estados.sigla"
```

Agora vamos criar um vetor com os o tamanho da população para cada cidade e em seguida adicioná-lo ao **Data frame**.

```
populacao <- c(1500, 2000, 1800, 12500, 300)
dfA[3] <- populacao
#dfA[3]
dfA
```

```
##        cidades estados.sigla    V3
## 1 Campo Grande            MS  1500
## 2       Cuiabá            MT  2000
## 3     Curitiba            PR  1800
## 4    São Paulo            SP 12500
## 5      Miranda            MS   300
```

> Note que a coluna criada recebeu nome v3 que corresponde ao tipo **Vetor** associado ao valor **3** informado.
> Também poderíamos ter passado diretamente o nome "população".

f. Alterar o nome de uma coluna

Vamos alterar o nome das colunas pelos índices.

```
names(dfA)[3] <- "População"
dfA
```

```
##        cidades estados.sigla População
## 1 Campo Grande            MS      1500
## 2       Cuiabá            MT      2000
## 3     Curitiba            PR      1800
## 4    São Paulo            SP     12500
## 5      Miranda            MS       300
```

```
names(dfA)[1:2] <- c("Município", "UF")
dfA
```

```
##      Município UF População
## 1 Campo Grande MS      1500
## 2       Cuiabá MT      2000
## 3     Curitiba PR      1800
## 4    São Paulo SP     12500
## 5      Miranda MS       300
```

Primeiramente foi alterado o nome da coluna de índice **3**.
Em seguida foram alterados os nomes das colunas com índice **1** e **2**.

Agora sim um bom exemplo de como podemos utilizar um filtro um pouco mais complexo.

```
dfA[c(2:4), c(1,3)]
```

```
##   Município População
## 2    Cuiabá      2000
## 3  Curitiba      1800
## 4 São Paulo     12500
```

No exemplo acima selecionamos as linhas 2 a 4 apenas da 1ª e 3ª coluna.

g. Outras funções e métodos úteis para trabalhar com Data frames

Verificar o número de linhas x colunas:

```
dim(dfA)
```

```
## [1] 5 3
```

Verificar os tipos de dados:

```
str(dfA)
```

```
## 'data.frame':   5 obs. of  3 variables:
##  $ Município: chr  "Campo Grande" "Cuiabá" "Curitiba" "São Paulo" ...
##  $ UF       : chr  "MS" "MT" "PR" "SP" ...
##  $ População: num  1500 2000 1800 12500 300
```

```
str(dfA$Município)
```

```
##  chr [1:5] "Campo Grande" "Cuiabá" "Curitiba" "São Paulo" "Miranda"
```

Na última saída verificamos o tipo de dados de uma coluna específica.

Acessar e obter valores de uma coluna:

```
dfA$cidades # Acessa uma coluna ...Se ela existir
```

```
## NULL
```

```
dfA$Município
```

```
## [1] "Campo Grande" "Cuiabá"       "Curitiba"     "São Paulo"    "Miranda"
```

```r
dfA["Município"]
```

```
##      Município
## 1 Campo Grande
## 2       Cuiabá
## 3     Curitiba
## 4    São Paulo
## 5      Miranda
```

```r
dfA$Município[1]
```

```
## [1] "Campo Grande"
```

Na última saída foi acessado o 1º registro da coluna especificada.

h. Filtrando com a função *filter()* dplyr

Quando informado um valor desejado de uma coluna é obtida a resposta se o valor informado está ou não em cada registro do **Data frame** .

```r
cidade.MS <- dfA[['UF']] == "MS"
cidade.MS
```

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

Observe que foi retornado valor **TRUE** para o primeiro e último índice do **Data frame**, pois são municípios da Unidade da Federação **MS**.

Uma forma de retornar os valores pelo parâmetro desejado é usando a função *filter()* do pacote **dplyr**.
O pacote dplyr é simples de usar e auxilia na manipulação de dados.

```r
#library(dplyr)
cidade.MS <- filter(dfA, dfA[['UF']] == "MS")
cidade.MS
```

```
##      Município UF População
## 1 Campo Grande MS      1500
## 2      Miranda MS       300
```

Foi retornado um **Data frame** contendo apenas os registros desejados.
Foi importada a biblioteca **dplyr**, porém deixei comentado para não gerar o código no documento.

---

5. Matriz

Consiste em uma estrutura na qual cada elemento é indexado pelo índice da linha e pelo índice da coluna, assim como nas matrizes que estudamos em matemática.
Armazenam um único tipo de dado.

a. Criar uma matriz

Como exemplo vamos criar matrizes (5x5) com valores de 1 a 25.

```
matrizA <- matrix(seq(1:25), nrow = 5)
matrizA
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    6   11   16   21
## [2,]    2    7   12   17   22
## [3,]    3    8   13   18   23
## [4,]    4    9   14   19   24
## [5,]    5   10   15   20   25
```

Observe que foi necessário definir o número de linhas.
Caso esta informação fosse omitida seria gerada uma matriz de uma só coluna linha a linha.
Ao invés de linhas também poderíamos ter definido o número de colunas.

```
matrizB <- matrix(seq(1:25),
            ncol = 5,
            byrow = TRUE,
            dimnames = list(c(seq(1:5)),
                            c('A','B','C','D','E')
            )
)
matrizB
```

```
##    A  B  C  D  E
## 1  1  2  3  4  5
## 2  6  7  8  9 10
## 3 11 12 13 14 15
## 4 16 17 18 19 20
## 5 21 22 23 24 25
```

Neste, diferente do exemplo anterior, foi definido o número de colunas em **ncol** = 5.
No argumento seguinte foi definido o parâmetro **byrow** para que os valores fossem inseridos em cada coluna linha a linha, de modo que só se inicia uma nova linha quando preencher o total de colunas passada como argumento.
Caso este parâmetro não fosse definido, os valores seriam indexados uma coluna inteira após a outra.
Outro parâmetro que foi definido foi o **dimnames** recebendo uma lista de dois elementos. O primeiro é um vetor com os valores que compõem os índices das linhas. O segundo elemento é um vetor com os índices das colunas.

b. Filtrar matriz

Podemos filtrar a matriz da seguinte maneira:

```
matrizB[c(1:3), c("B","C")]
```

```
##    B  C
## 1  2  3
## 2  7  8
## 3 12 13
```

No primeiro argumento definimos a seleção das linhas 1 a 3 e, no segundo argumento selecionamos apenas as colunas com índices “B” e “C”.

Gráficos

Para demostrar alguns modelos gráficos vamos utilizar o *Data frame* **dfA** declarado anteriormente.

```
dfA
```

```
##      Município UF População
## 1 Campo Grande MS      1500
## 2       Cuiabá MT      2000
## 3     Curitiba PR      1800
## 4    São Paulo SP     12500
## 5      Miranda MS       300
```

Informações sobre os valores do Data frame

```
summary(dfA$População)
```

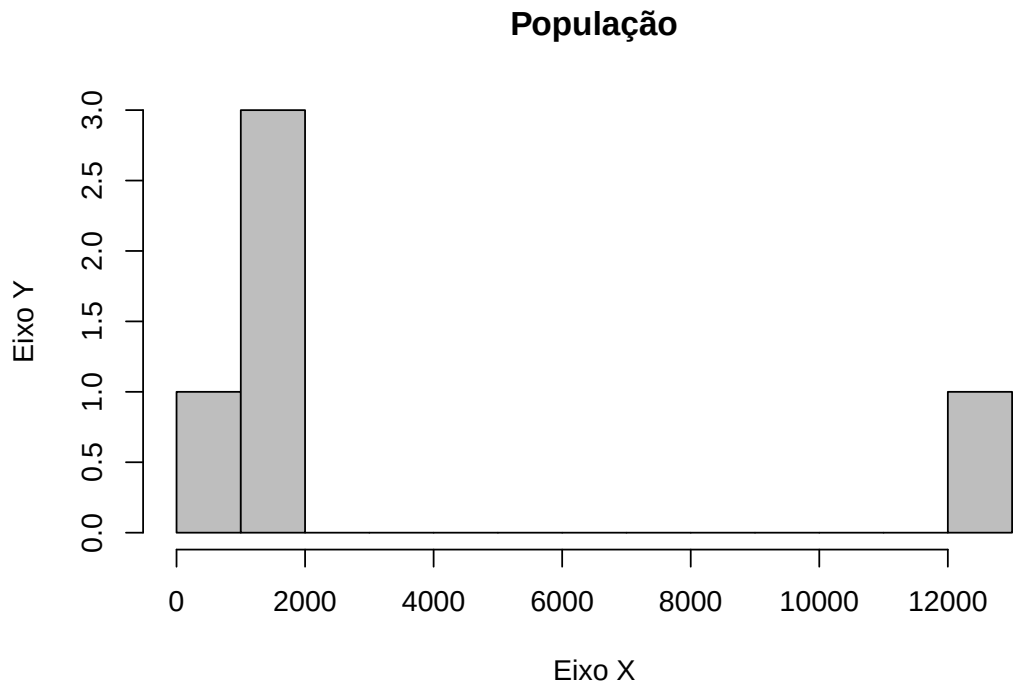
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     300    1500    1800    3620    2000   12500
```

A função *summary()* é usada para retornar informações como média, mediana, valor mínimo e máximo.

1. Histograma

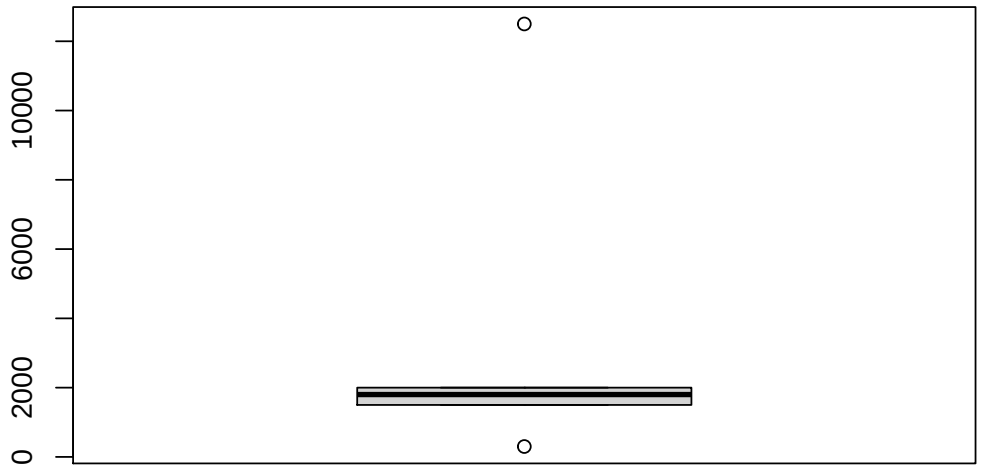
Vamos criar um **histograma** com os dados da coluna **População**.

```
hist(dfA$População,
    main="População",
    xlab="Eixo X",
    ylab="Eixo Y",
    col="grey",
    breaks=10)
```



2. Boxplot

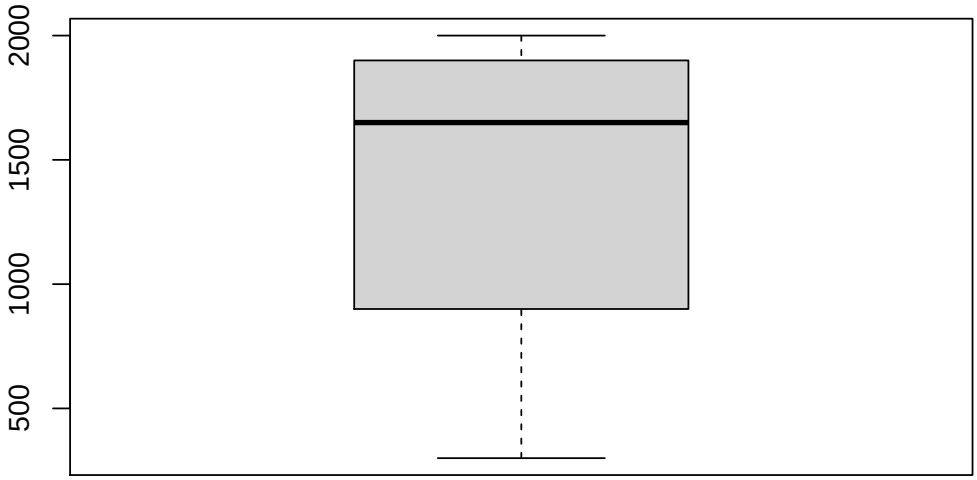
```
boxplot(dfA$População)
```



>Note que na coluna **População** existem valores muito distantes da média, então vamos utilizar a função *filter()* para remover tais valores apenas para demonstrar o uso desta função no caso de *outliers* (valores discrepantes).

```
dfB <- dfA %>%
 filter(População <= 6000)
```

```
boxplot(dfB$População)
```


2000
1500
1000
500

3. Table

a. verificando a quantidade de ocorrências para cada coluna

```
table(dfA$Município)
```

```
##
## Campo Grande       Cuiabá     Curitiba      Miranda    São Paulo
##            1            1            1            1            1
```

```
table(dfA$UF)
```

```
##
## MS MT PR SP
##  2  1  1  1
```

```
table(dfA$UF, exclude="PR")
```

```
##
## MS MT SP
##  2  1  1
```

b. quantidade de registros

```
length(table(dfA$Município))
```

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

```
length(table(dfA$UF))
```

```
## [1] 4
```

c. obter proporção das ocorrências pelo total de registros

```
prop.table(table(dfA$Município))
```

```
##
## Campo Grande       Cuiabá     Curitiba      Miranda    São Paulo
##          0.2          0.2          0.2          0.2          0.2
```

```
prop.table(table(dfA$UF))
```

```
##
##  MS  MT  PR  SP
## 0.4 0.2 0.2 0.2
```

```r
prop.table(table(dfA$UF))*100
```

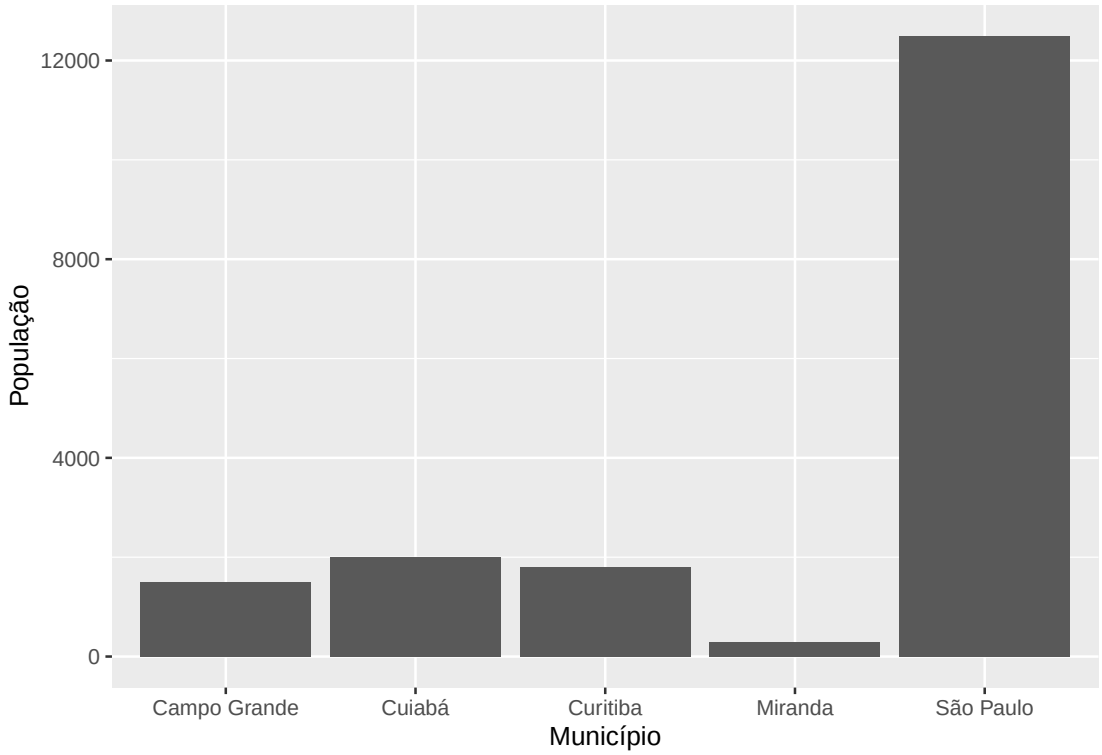
```
##
## MS MT PR SP
## 40 20 20 20
```

Na última linha o valor foi convertido em percentual

4. Gráfico de Barras

População por cidade

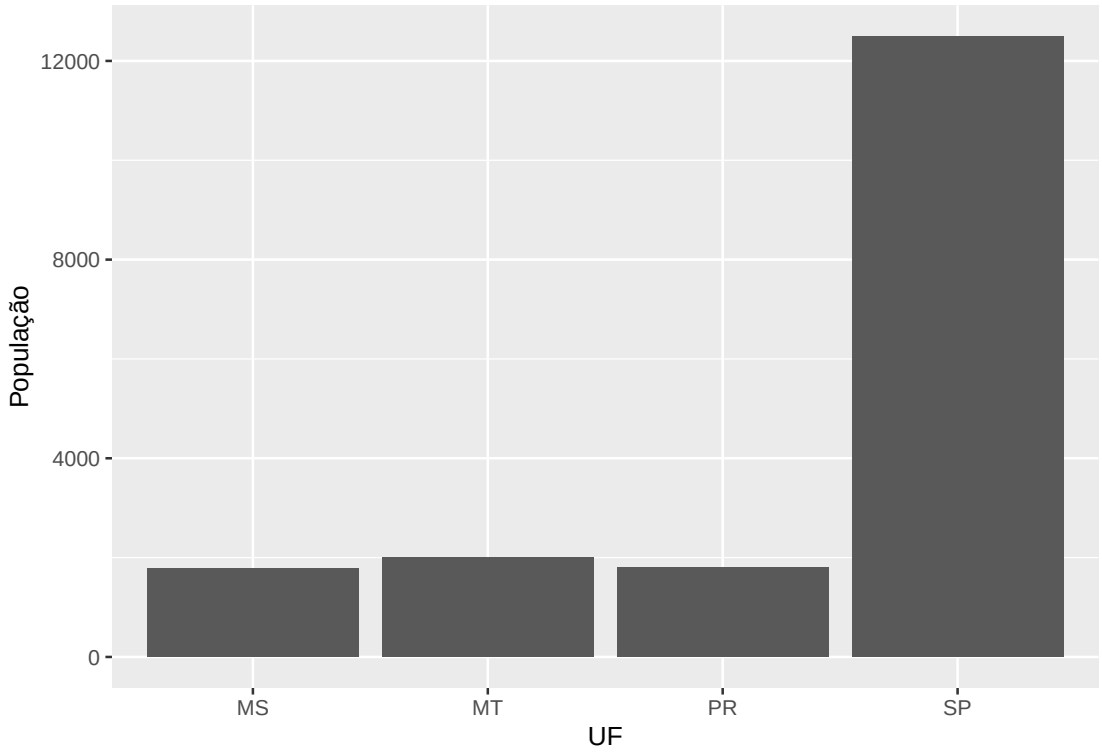
```
#library(ggplot2)
ggplot(dfA, aes(x=Município, y=População))+
 geom_bar(stat='identity')
```



>Neste exemplo utilizamos a classe **gglplot** para criar o gráfico de barras.

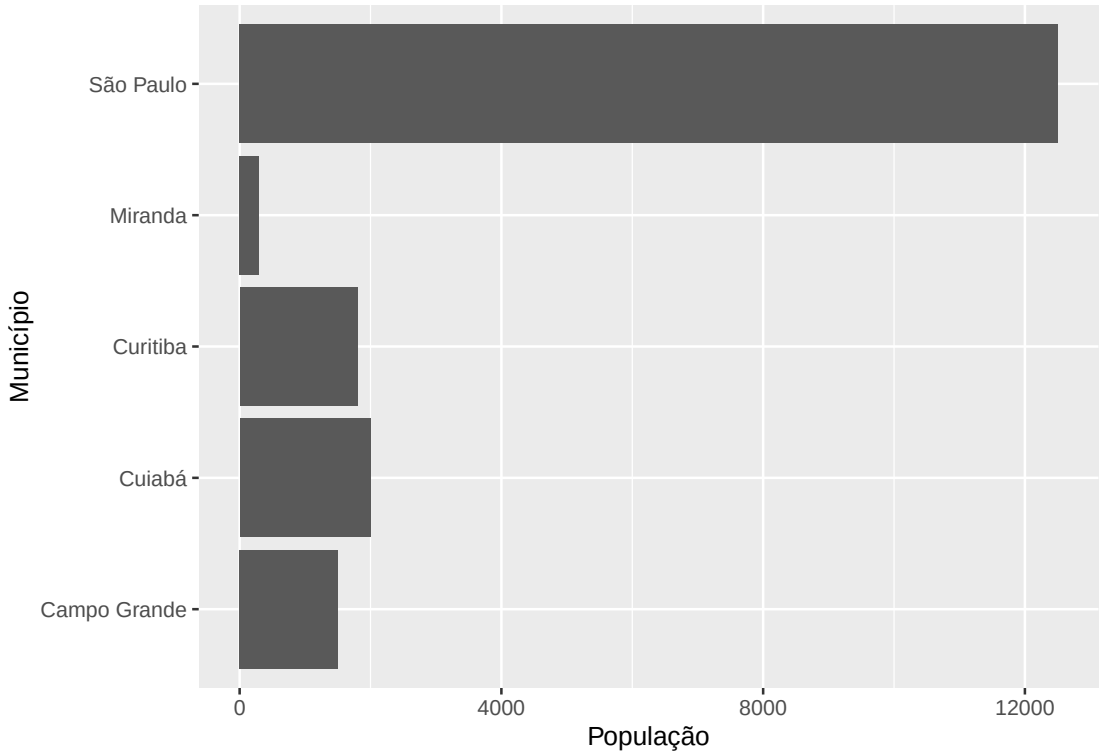
População por Unidade da Federação

```
ggplot(dfA, aes(x=UF, y=População))+
 geom_bar(stat='identity')
```


12000
8000
4000
0
População
MS
MT
PR
SP
UF

5. Gráfico de Barras Invertidas

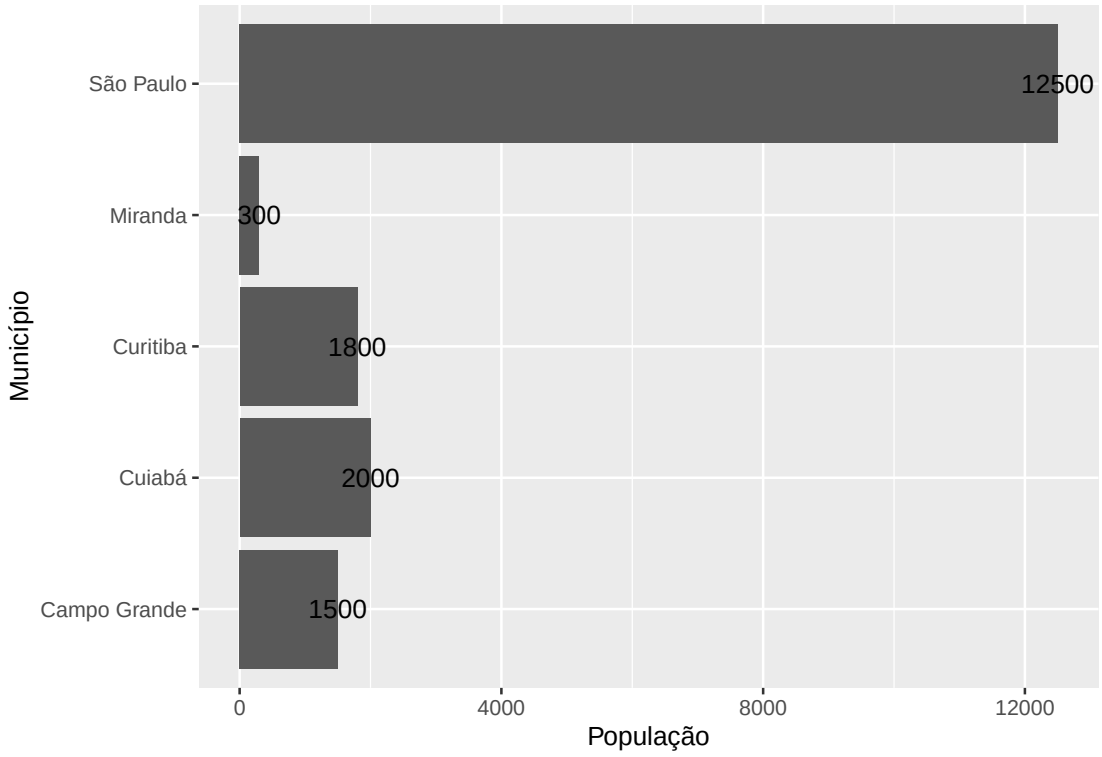
```
ggplot(dfA, aes(x=Município, y=População))+
 geom_bar(stat='identity')+
 coord_flip()
```



>O comando *coord_flip()* converte para barra invertida.

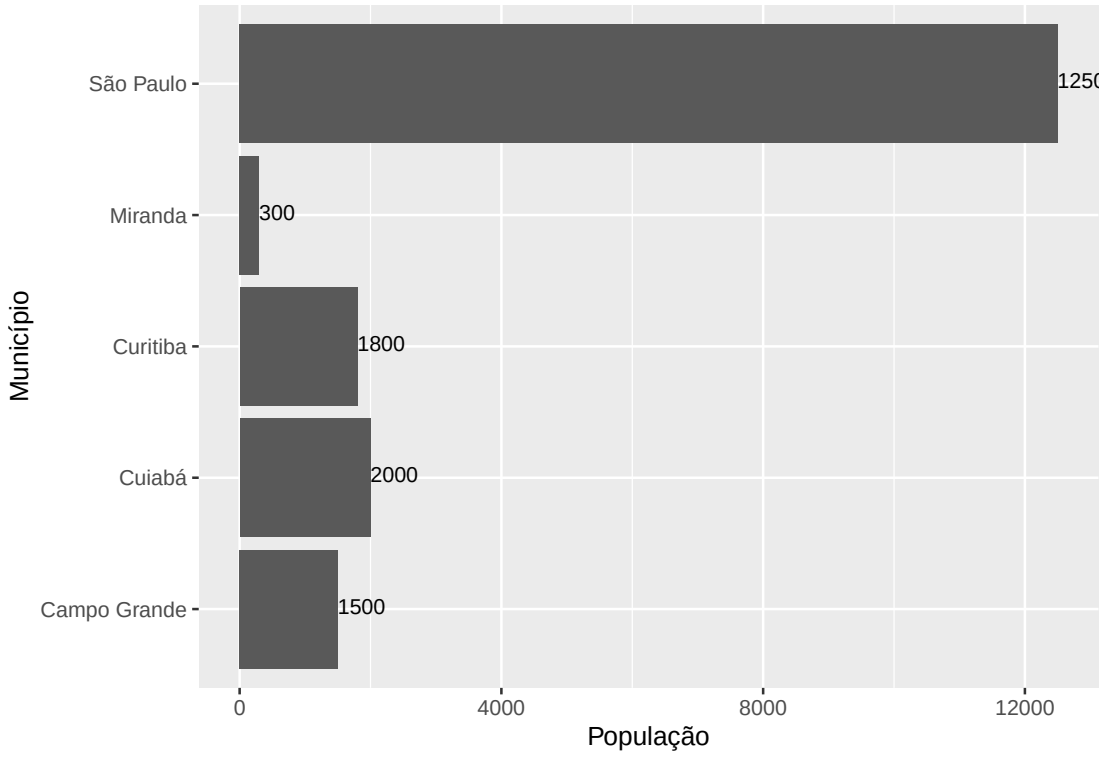
a. adicionar rótulos de dados

```
ggplot(dfA, aes(x=Município, y=População))+
 geom_bar(stat='identity')+
 geom_text(aes(label=População))+
 coord_flip()
```



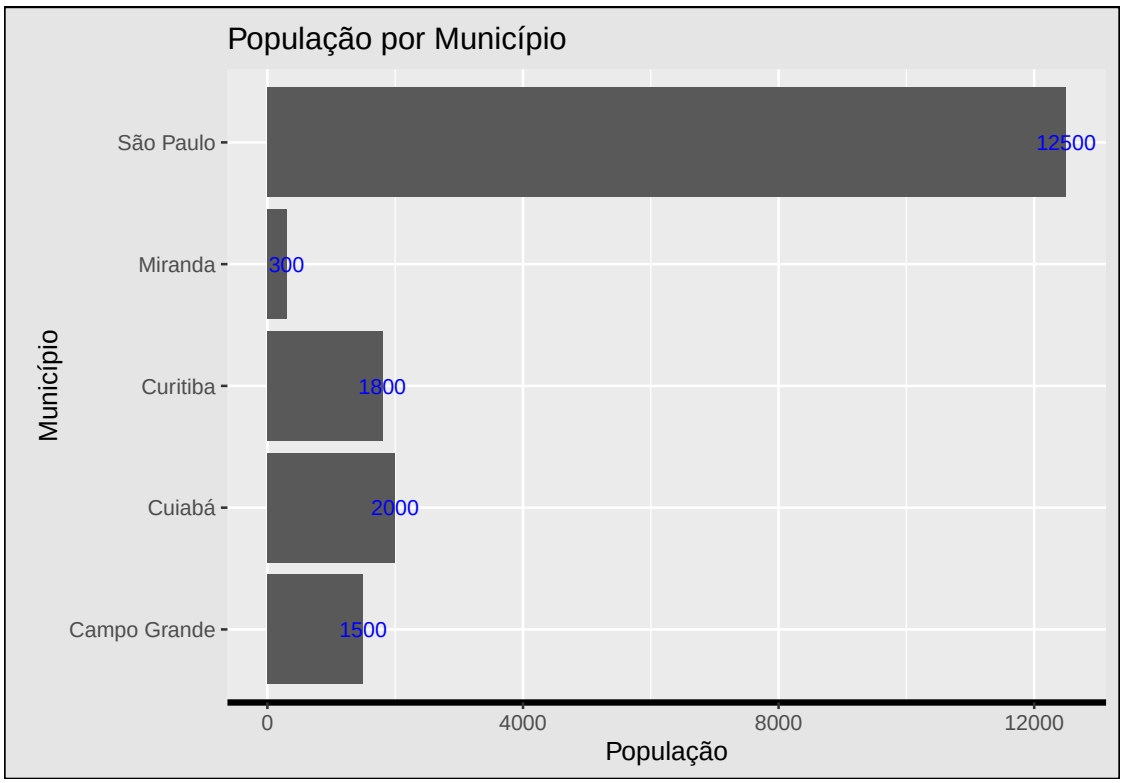
b. Ajustar posição dos rótulos de dados

```
ggplot(dfA, aes(x=Município, y=População))+
 geom_bar(stat='identity')+
 geom_text(aes(label=População), hjust=0, vjust=0.3, size=3)+
 coord_flip()
```



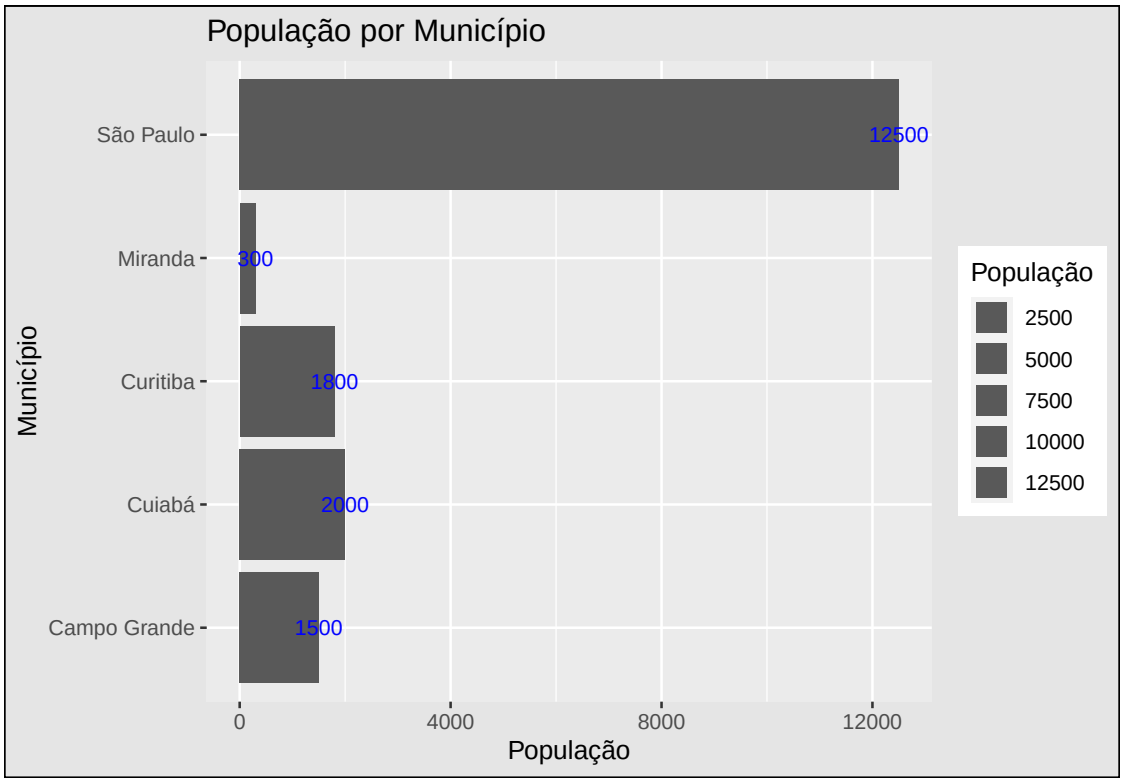
c. Formatando título, margens da área do gráfico e outras configurações

```
ggplot(dfA, aes(x=Município, y=População))+
 ggtitle("População por Município")+
 geom_bar(stat='identity')+
 geom_text(aes(label=População), hjust=0.5, vjust=0.5, size=3, color="blue")+
 coord_flip()+ # Muda para barra invertida
 #theme(axis.title.x=element_text(vjust=-0.5))+
 #theme(axis.title.y=element_text(angle=90, vjust=-0.5))+
 theme(plot.margin = unit(c(0.3,0.2,0.2,0.5), "cm"))+
 #theme(axis.line = element_line(arrow = arrow()))+
 #theme(axis.line.x = element_line(arrow = arrow()))+
 theme(axis.line.x = element_line(size=1.25))+
 theme(plot.background=element_rect(color="black",
                                    fill = "grey90"))
```



d. Formatando legenda

```
ggplot(dfA, aes(x=Município, y=População, size=População))+
 ggtitle("População por Município")+
 geom_bar(stat='identity')+
 geom_text(aes(label=População), hjust=0.5, vjust=0.5, size=3, color="blue")+
 coord_flip()+
 theme(plot.background=element_rect(color="black",
                                    fill = "grey90"))
```



Dicas

##Acessar ajuda sobre pacotes no RStudio

Na janela "*Source*" (local onde digitamos o código), você pode acessar ajuda sobre os pacotes utilizados neste projeto digitanto **?** seguido no nome do pacote, assim:

- **?c** para vetor;
- **?factor** para fator;