TEXTO DE APOIO

AULA 1

Probabilidade e Estatística Aplicadas

Professor Rogério de Oliveira

EaD | Universidade Presbiteriana Mackenzie

Sumário

PROBABILIDADE, ESTATÍSTICA E INTRODUÇÃO AO R

INTRODUÇÃO

Probabilidade e estatística são uma base importante para diferentes campos da ciência da computação, engenharias, tecnologia da informação e ciências humanas e biológicas etc. São também a base do que atualmente chamamos de Ciência de Dados, com intersecções com o aprendizado de máquina, inteligência artificial e muitas outras áreas. São conceitos fundamentais para compreendermos quaisquer conjuntos de dados que envolvem estatística descritiva de dados, uso de amostras, análise de distribuições e correlação, além de técnicas de inferência, como regressão e teste de hipóteses, instrumentos úteis, se não essenciais, para a tomada de decisões.

Para aplicarmos esses conceitos, você empregará a linguagem R. R é uma linguagem de código aberto (*open source*), com uma série de recursos para Análise de Dados, incluindo funções estatísticas, de visualização dos dados e de amplo uso no mercado em diversas áreas.

INTRODUÇÃO AO R

Aqui você terá uma introdução à linguagem R e ao ambiente R Studio. É esperado que, nesse ponto, você já tenha familiaridade com alguma linguagem de programa (Python, C++, Java) e editores de programação. Assim, faremos uma introdução baseada em exemplos, assumindo que você já saiba os conceitos básicos como variáveis, tipos de dados, atribuição, controles de fluxo em programas etc.

Você pode, nesse ponto, preferir acessar outros materiais da internet com cursos, videoaulas etc.: fique à vontade para buscar outras formas de se familiarizar com a linguagem R.

ACESSANDO O RSTUDIO

Neste componente, você pode empregar o RStudio Cloud, um ambiente em nuvem gratuito para o desenvolvimento de aplicações. Se preferir o uso de um ambiente local, acesse RProject e RStudio fazendo a instalação do R e depois do RStudio.

Semelhante a outros editores de linguagem de programação, você encontra no RStudio uma barra de opções (arquivo, edit, code, view etc.) e áreas de trabalho:

- Uma área para edição dos programas.
- Uma console para comandos e saídas dos programas.
- Uma área de arquivos, plots etc.
- Uma área de objetos, onde você pode inspecionar o estado de suas variáveis.

HELP

Antes de começar, o R fornece documentação extensa. Por exemplo, inserindo?`c` ou `help(c)` no prompt da console, exibe-se a documentação da função c em R.

```
help(table)
```

Se você não tiver certeza do nome da função, pode realizar uma pesquisa com a função `apropos`.

```
apropos('nova')
```

```
## [1] "anova"            "manova"           "power.anova.test" "stat.anova"
## [5] "summary.manova"
```

ARITMÉTICA COM R

Em sua forma mais básica, R pode ser usado como uma calculadora simples, empregando os operadores `+, -, *, /, ^` (potência) ou `%%` (módulo).

E você pode executar esses comandos no console:

```
# Uma adição
5  +  3
```

```
## [1] 8
```

```
# Uma subtração
```

```r
5  -  3
```

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

```r
# Uma multiplicação
3  *  5
```

```
## [1] 15
```

```r
# Uma divisão
( 5  +  5 ) /  2
```

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

ATRIBUIÇÃO DE VARIÁVEL

Uma variável, como em outras linguagens, permite que você armazene um valor ou um objeto e empregue o nome dessa variável para acesso aos valores armazenados. Os valores são atribuídos com <-, ou **indiferentemente**, com = (a partir de todas as versões mais recentes de R). Prefira empregar o =.

```r
# São equivalentes
my_var1 <- 4
my_var2 = 4
```

O R implementa tipos fracos, a exemplo do Python em que o tipo da variável é definido dinamicamente sem a necessidade de declaração da variável.

Os valores podem ser exibidos automaticamente a partir do nome da variável. Mas somente a última saída é exibida.

```r
my_var1
```

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

```r
my_var2
```

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

Ou você pode optar pelo print dos valores.

```r
print(my_var1)
```

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

```r
print(my_var2)
```

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

Observe que o R é sensível a maiúsculas e minúsculas! Assim:

```
My_var = 5
my_var = 4
print(My_var == my_var)
```

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

TIPOS DE DADOS BÁSICOS EM R

Alguns dos tipos mais básicos são:

```
my_numeric  =  42.55

my_text = "algum texto"

my_caracter = 'a'

my_logical  = TRUE # Boolean

sprintf("Valores = %s , %s , %s , %d", my_numeric, my_text, my_caracter, my_
logical)
```

```
## [1] "Valores = 42.55 , algum texto , a , 1"
```

```
sprintf("Tipos = %s , %s , %s , %s", class(my_numeric), class(my_text), class(my_
caracter), class(my_logical))
```

```
## [1] "Tipos = numeric , character , character , logical"
```

Aspas simples ou duplas são usadas indiferentemente. O comando `sprintf` permite a formatação da saída em C-Style e o comando `class` pode ser empregado para exibir a classe ou o tipo de dado.

VETORES

Um tipo de dado essencial em R são os vetores. Lembre-se de que o R é uma linguagem para trabalhar com coleções de dados, e a coleção mais simples é a de vetores. Para criar esses vetores você pode empregar o comando `c( )`, que significa **combine**, e são estruturas bastante semelhantes às listas em Python (embora não possuam os mesmos métodos).

```r
vetor_numerico = c(1, 2, 3)

vetor_caracter = c("a", "b", "c")

vetor_mix = c(1, 'a', TRUE)

cat('Tipos = ', class(vetor_numerico), class(vetor_caracter), class(vetor_mix))

## Tipos =  numeric character character

help(cat)
```

Como você pode ver acima, os vetores podem ter vários tipos de dados e assumem o tipo mais abrangente. O comando `cat( )` 'Concatenate and Print' é empregado aqui para exibir os tipos de dados como alternativa ao `print()` e `sprintf()` usados anteriormente.

Como nos arrays de outras linguagens, os vetores são listas ordenadas e indexadas e você pode acessar e alterar os valores referenciando o índice dos elementos.

```r
cat('Antes: ', vetor_numerico, '\n')

## Antes:  1 2 3

vetor_numerico[2] = 99
cat('Depois: ', vetor_numerico, '\n')

## Depois:  1 99 3

for (i in 1:3) cat('\n', vetor_numerico[i])

##
##  1
##  99
##  3
```

Você deve notar que os índices começam por 1 (diferentemente de listas em Python que iniciam com 0) e, abaixo, você encontra algumas formas que serão úteis para inicializar vetores.

```r
x = c(1,-2,3)
y = seq(1,10,0.5)
z = c(1:10)
t = rep(c(1,2),each=3) # bem sofisticado aqui... :-)

cat('x= ', x, '\ny= ', y, '\nz= ', z, '\nt= ', t)

## x=  1 -2 3
## y=  1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9 9.5 10
## z=  1 2 3 4 5 6 7 8 9 10
## t=  1 1 1 2 2 2
```

O R, muito mais orientado a trabalhar com coleções de dados que com variáveis simples, permite que uma série de funções sejam aplicadas diretamente às coleções, sem a necessidade de iterações (loops), sendo assim mais próximo do comportamento de listas em Python do que seria comum em arrays de C++ ou Java.

Portanto, no lugar de:

```
x = seq(1,10,1)
cat('Antes ', x)
```

```
## Antes  1 2 3 4 5 6 7 8 9 10
```

```
for (i in 1:10){           # C++ e Java arrays funcionam desse modo
  x[i] = x[i] + 1
}

cat('\nDepois ', x)
```

```
##
## Depois  2 3 4 5 6 7 8 9 10 11
```

Você pode simplesmente escrever:

```
x = seq(1,10,1)
cat('Antes ', x)
```

```
## Antes  1 2 3 4 5 6 7 8 9 10
```

```
x = x + 1                  # em R a operação + é feita para todos os elementos da
coleção, sem a necessidade das iterações

cat('\nDepois ', x)
```

```
##
## Depois  2 3 4 5 6 7 8 9 10 11
```

E, de modo análogo, outras operações:

```
x = seq(1,10,1)
cat('Antes ', x)
```

```
## Antes  1 2 3 4 5 6 7 8 9 10
```

```
cat('\nDepois... \n')
```

```
##
## Depois...
```

```
print(2*x)
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
print(x+1)
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11
```

```
print(sum(x))
```

```
## [1] 55
```

```
print(max(x), min(x))
```

```
## [1] 10
```

```
print(sqrt(x))
```

```
##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
3.000000
## [10] 3.162278
```

O que se aplica também para funções lógicas:

```
x = seq(1,10,1)
```

```
print(x > 5)
```

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

o que fornece uma forma bastante prática para você fazer a seleção de valores em uma coleção:

```
x = seq(1,10,1)
```

```
L = x > 5          # vetor lógico selecionando os índices que correspondem a
valores > 5
```

```
print(x[L])
```

```
## [1]  6  7  8  9 10
```

MATRIZES E ARRAYS

Matrizes e Arrays também podem ser criados. As matrizes são coleções de duas dimensões, enquanto os arrays são generalizações de matrizes para mais (ou menos) de duas dimensões. Mas essas estruturas serão pouco empregadas aqui, sendo muito mais frequente o uso de vetores.

```r
X = matrix(c(1,2,3,4,5,6),nrow=2,ncol=3); X                # Uma matriz 2x3
```

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

```r
X = array(c(1,2,3,4,5,6,7,8,9,10,1,1),dim=c(3,2,2)); X     # Um array: 2 matrizes 3x2
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
##
## , , 2
##
##      [,1] [,2]
## [1,]    7   10
## [2,]    8    1
## [3,]    9    1
```

FATORES

Para lidar com medições nas escalas nominais e ordinais, o R fornece vetores de tipo fator. Um fator é uma variável que pode assumir apenas um número discreto (finito) de valores distintos (os `levels`). Para criar um fator você pode aplicar a função `factor` a um vetor de qualquer classe:

```r
# gender com 5 “male” e 10 “female”
gender = c(rep(“male”,5), rep(“female”, 10))

sprintf(gender, ‘\n’, class(gender) , ‘\n’)                    # como vetor
```

```
##  [1] “male”   “male”   “male”   “male”   “male”   “female” “female” “female”
“female”
## [10] “female” “female” “female” “female” “female” “female”
```

```r
gender = factor(gender)                                        # como factor
sprintf(as.character(gender), ‘\n’, class(gender) , ‘\n’)
```

```
##  [1] “male”   “male”   “male”   “male”   “male”   “female” “female” “female”
“female”
## [10] “female” “female” “female” “female” “female” “female”
```

```r
# factor faz com que os valores sejam armazenados internamente 1=female, 2=male
internally (ordem alfabética) e o R tratará gender como uma variável nominal e
não como caracteres!

cat(gender, ‘\n’)
```

```
## 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1
```

```
summary(gender)
```

```
## female   male
##     10      5
```

O uso de fatores envolve uma série de detalhes e, por ora, é suficiente você saber que existem essas coleções que criam valores nominais e ordinais.

```
rating = c(rep("large",1), rep("medium",2), rep("small",3))
rating = ordered(rating)
print(rating)
```

```
## [1] large  medium medium small  small  small
## Levels: large < medium < small
```

CARACTERES

Vale a pena também você olhar algumas funções úteis para a manipulação de caracteres em R como a conversão para caracteres, a concatenação de strings e substrings:

```
x = as.character(3.14) # converte para caracteres
cat(x, class(x), '\n')
```

```
## 3.14 character
```

```
y = paste('Universidade','Presbiteriana','Mackenzie') # concatena strings
print(y)
```

```
## [1] "Universidade Presbiteriana Mackenzie"
```

```
print( substr (y, start = 14, stop = 26) ) # substring
```

```
## [1] "Presbiteriana"
```

INPUT PELO TECLADO

Você pode fazer a entrada de dados pelo teclado. Isso pode ser feito pelo comando `readline`, o que é equivalente ao comando `input` do Python.

```
nome =  readline(prompt="Entre com seu nome: ")
idade =  as.numeric(readline(prompt="Entre com sua idade: "))

cat('nome: ', nome, 'idade: ', idade, '\n')
```

```
## nome:  Rogério idade:  56
```

ESTRUTURAS DE CONTROLE DE FLUXO

Estruturas e controle de fluxo como `for, while, if` são bastante análogas às outras linguagens, inclusive na sintaxe:

```
if(cond) expr
if(cond) cons.expr  else  alt.expr

for(var in seq) expr

while(cond) expr

repeat expr

break
next
```

Você pode ver alguns exemplos abaixo, mas deixaremos para explorar melhor esses comandos sugerindo que você comece pelo `help('for')`.

```
for(i in 1:5) print(1:i)
```

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

```
for(n in c(2,5,10,20,50)) {
   x = stats::rnorm(n)
   cat(n, ": ", sum(x^2), "\n", sep = "")
}
```

```
## 2: 4.44905
## 5: 4.534608
## 10: 22.04178
## 20: 29.33503
## 50: 50.45543
```

```
f = factor(sample(letters[1:5], 10, replace = TRUE))
for(i in unique(f)) print(i)
```

```
## [1] "d"
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "e"
```

```
a = 5; b = 6
if (a < b) print('menor') else print('maior')
```

```
## [1] "menor"
```

E você pode empregar `{}` para definir blocos de instruções que são executadas em conjunto.

PACOTES

Existem muitos pacotes que implementam funcionalidades adicionais ao R e você empregará alguns deles ao longo do curso. Você deve empregar o comando `library()` para 'importar' as funcionalidades de uma biblioteca já instalada. Caso o pacote ainda não tenha sido instalado, você deverá usar o `install.packages()`, mas este último comando, em geral, precisa ser feito uma única vez em seu ambiente.

```
# install.packages('ggplot2')  # comentado aqui
library(ggplot2)
```

O comando acima instala e importa um pacote bastante popular para gráficos e visualização de dados em R.

OUTROS RECURSOS

O R possui, ainda, muito mais recursos do que você aprendeu aqui, como recursos para orientação a objetos, suporte a operações em arquivos etc. Mas o aprendizado de uma linguagem, assim como aprender uma língua, não pode ser feito de uma única vez: requer o uso e a prática, e aqui você tem mais do que o suficiente para prosseguir no uso do R. E você ainda pode acessar à documentação do R.

DATAFRAMES

Uma das formas mais comuns de tratarmos dados em R são os dataframes. Os dataframes são tabelas de dados semelhantes às que você encontra em um banco de dados relacional (SQL). Eles são uma lista ou coleção de vetores de igual comprimento.

```r
nr = c(1, 2, 3)
nome = c("Ana", "Adriana", "Daniel")
estudante = c(TRUE, FALSE, TRUE)
idade = c(15, 18, 17)
df = data.frame (nr, nome, estudante, idade)

print(df)
```

VETOR DE COLUNA

De modo semelhante ao SQL, você pode referenciar os atributos (as colunas) a partir do nome da coluna como vetores.

```r
print(df$nome)
```

```
## [1] "Ana"     "Adriana" "Daniel"
```

```r
print(df$idade)
```

```
## [1] 15 18 17
```

Note que, em R, empregamos o símbolo '$' para referenciar o atributo no lugar de '.', como é empregado, em geral, para fazer referências de objetos.

Cada atributo sendo um vetor, você pode aplicar todas as funções que são aplicáveis a vetores para esses atributos e pode ver algumas das funções que empregamos anteriormente aqui:

```r
sprintf('Idade mínima: %d', min(df$idade))
```

```
## [1] "Idade mínima: 15"
```

```r
sprintf('Idade média: %.2f', mean(df$idade))
```

```
## [1] "Idade média: 16.67"
```

```r
print('Antes: ')
```

```
## [1] "Antes: "
```

```r
print(df)

# Somando 1 na idade
df$idade = df$idade + 1

print('Depois: ')
```

```
## [1] "Depois: "
```

```r
print(df)
```

Ou, ainda, operações lógicas:

```
print(df$idade >= 18)
```

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

Você verá como empregar isso abaixo para a seleção de linhas de uma tabela.

SELECIONANDO DADOS DE UM DATAFRAME

Uma das operações mais importantes sobre os dados é a seleção de dados de interesse. Você pode estar interessado somente em algumas linhas do seu conjunto de dados como 'os alunos com mais de 18 anos', 'os clientes do estado de São Paulo', 'os produtos com preço abaixo de ...' determinado valor, ou ainda querer selecionar somente alguns atributos do conjunto de dados.

Nos exemplos a seguir, empregaremos o `mtcars`, um dos datasets built-in do R. Mais adiante, você aprenderá como importar dados de fontes próprias.

CARACTERÍSTICAS DOS DADOS

Os comandos abaixo permitem explorar a estrutura básica de um dataset, como o número de linhas, colunas, tipos de dados etc. e você, provavelmente, começará uma exploração de dados por esses comandos para entender a estrutura dos dados em mãos antes de selecioná-los e analisá-los de fato.

Você pode executar cada um dos comandos em separado para exibir suas saídas.

```
# execute cada um dos comandos isoladamente
head(mtcars)     # cabeçalho do dataframe
tail(mtcars)     # final do dataframe
dim(mtcars)      # dimensões
```

```
## [1] 32 11
```

```
ncol(mtcars)     # nr de colunas
```

```
## [1] 11
```

```
nrow(mtcars)     # nr de linhas
```

```
## [1] 32
```

```
names(mtcars)    # nome das colunas
```

```
##  [1] “mpg”  “cyl”  “disp” “hp”   “drat” “wt”   “qsec” “vs”   “am”   “gear”
“carb”

rownames(mtcars) # nome das linhas

##  [1] “Mazda RX4”           “Mazda RX4 Wag”       “Datsun 710”
##  [4] “Hornet 4 Drive”      “Hornet Sportabout”   “Valiant”
##  [7] “Duster 360”          “Merc 240D”           “Merc 230”
## [10] “Merc 280”            “Merc 280C”           “Merc 450SE”
## [13] “Merc 450SL”          “Merc 450SLC”         “Cadillac Fleetwood”
## [16] “Lincoln Continental” “Chrysler Imperial”   “Fiat 128”
## [19] “Honda Civic”         “Toyota Corolla”      “Toyota Corona”
## [22] “Dodge Challenger”    “AMC Javelin”         “Camaro Z28”
## [25] “Pontiac Firebird”    “Fiat X1-9”           “Porsche 914-2”
## [28] “Lotus Europa”        “Ford Pantera L”      “Ferrari Dino”
## [31] “Maserati Bora”       “Volvo 142E”

str(mtcars)      # Estrutura dos dados, como exibido no RStudio na janela
superior direita

## ‘data.frame’:    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...

summary(mtcars)  # Summary statistics

##       mpg             cyl             disp             hp             drat
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0   Min.   :2.760
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5   1st Qu.:3.080
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0   Median :3.695
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7   Mean   :3.597
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0   3rd Qu.:3.920
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0   Max.   :4.930
##        wt             qsec             vs              am              gear
##  Min.   :1.513   Min.   :14.50   Min.   :0.0000   Min.   :0.0000   Min.
:3.000
##  1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000   1st Qu.:0.0000   1st
Qu.:3.000
##  Median :3.325   Median :17.71   Median :0.0000   Median :0.0000   Median
:4.000
##  Mean   :3.217   Mean   :17.85   Mean   :0.4375   Mean   :0.4062   Mean
:3.688
##  3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd
Qu.:4.000
##  Max.   :5.424   Max.   :22.90   Max.   :1.0000   Max.   :1.0000   Max.
:5.000
##       carb
##  Min.   :1.000
```

```
##  1st Qu.:2.000
##  Median :2.000
##  Mean   :2.812
##  3rd Qu.:4.000
##  Max.   :8.000
```

SELEÇÃO DE LINHAS E COLUNAS

Sendo uma estrutura de duas dimensões, os dados de um dataframe podem ser referenciados do seguinte modo:

```
dataframe [ linhas , colunas ]
```

Onde `linhas` e `colunas` podem conter uma ou mais posições (uma seleção de dados). É comum também nos referirmos à seleção de dados como 'slice' (fatias) de dados.

A seleção de colunas pode ser feita referenciando o nome ou a posição dos atributos. E você deve notar a vírgula indicando que, neste caso, não há nenhum filtro para linhas. Execute isoladamente cada um dos comandos abaixo para exibir as saídas. As três seleções de `mtcars` produzem a mesma seleção de atributos.

```
# execute cada um dos comandos isoladamente
print(names(mtcars))
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
"carb"
```

```
mtcars[ , c('cyl','disp','hp')]
mtcars[ , c(2,3,4)]
mtcars[ , c(2:4)]
```

A seleção de linhas pode ser feita indicando-se a posição da linha ou, o que é mais interessante, um predicado de seleção.

```
# execute cada um dos comandos isoladamente
mtcars[ 1:6, ]                # equivalente a head(mtcars)
mtcars[ mtcars$cyl > 6, ]     # exibe somente veículos com mais de 6 cilindros
```

Você pode, agora, combinar a seleção de linhas e colunas para trazer quaisquer 'fatias' de seu interesse dos dados.

```
# execute cada um dos comandos isoladamente
mtcars[ mtcars$cyl > 6 & mtcars$am == 1, c('cyl','mpg','am') ]      # exibe
somente veículos com mais de 6 cilindros e transmissão am
```

Ou, ainda, selecionar um único vetor de dados desse modo:

```
# execute cada um dos comandos isoladamente
mtcars[ mtcars$cyl > 6 & mtcars$am == 1, ]$hp     # exibe somente o vetor de
valores hp para veículos com mais de 6 cilindros e transmissão am
```

```
## [1] 264 335
```

Como a seleção de dados de um dataframe retorna também um dataframe, você pode empregar a seleção para produzir dataframes próprios com seus dados de interesse.

```
# execute cada um dos comandos isoladamente

my_mtcars = mtcars[ mtcars$cyl > 6 & mtcars$am == 1, c('cyl','mpg','am') ]

head(my_mtcars)
nrow(my_mtcars)
```

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

```
summary(my_mtcars)
```

```
##       cyl         mpg             am
##  Min.   :8   Min.   :15.0   Min.   :1
##  1st Qu.:8   1st Qu.:15.2   1st Qu.:1
##  Median :8   Median :15.4   Median :1
##  Mean   :8   Mean   :15.4   Mean   :1
##  3rd Qu.:8   3rd Qu.:15.6   3rd Qu.:1
##  Max.   :8   Max.   :15.8   Max.   :1
```

```
print( my_mtcars[ my_mtcars$mpg > 15, ]) # todos os comandos aplicáveis a um
dataframe são aplicáveis em my_mtcars
```

IMPORTANDO UM DATAFRAME

O R permite importar dataframes de diferentes fontes de dados como bases SQL, arquivos .xlsx, .csv, .json etc. sendo necessário para alguns tipos de dados pacotes específicos para as funções e, por ora, vamos nos limitar a empregar arquivos com extensão .csv.

Você poderá acessá-los indicando uma `url` de origem dos dados de um diretório local em sua máquina. Se estiver empregando o RStudio.Cloud, lembre-se de que sua máquina R encontra-se na nuvem e, para o acesso 'local', é necessário o upload do arquivo.

```
surveys = read.csv("https://github.com/csc-training/da-with-r/raw/master/
DataFiles/portal_data_joined.csv")

head(surveys)
summary(surveys)
```

```
##    record_id         month             day            year         plot_id
##  Min.   :    1   Min.   : 1.000   Min.   : 1.0   Min.   :1977   Min.   : 1.00
##  1st Qu.: 8964   1st Qu.: 4.000   1st Qu.: 9.0   1st Qu.:1984   1st Qu.: 5.00
##  Median :17762   Median : 6.000   Median :16.0   Median :1990   Median :11.00
##  Mean   :17804   Mean   : 6.474   Mean   :16.1   Mean   :1990   Mean   :11.34
##  3rd Qu.:26655   3rd Qu.:10.000   3rd Qu.:23.0   3rd Qu.:1997   3rd Qu.:17.00
##  Max.   :35548   Max.   :12.000   Max.   :31.0   Max.   :2002   Max.   :24.00
##
##   species_id            sex            hindfoot_length     weight
##  Length:34786       Length:34786       Min.   : 2.00   Min.   :  4.00
##  Class :character   Class :character   1st Qu.:21.00   1st Qu.: 20.00
##  Mode  :character   Mode  :character   Median :32.00   Median : 37.00
##                                        Mean   :29.29   Mean   : 42.67
##                                        3rd Qu.:36.00   3rd Qu.: 48.00
##                                        Max.   :70.00   Max.   :280.00
##                                        NA's   :3348    NA's   :2503
##     genus              species              taxa            plot_type
##  Length:34786       Length:34786       Length:34786       Length:34786
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##
```