UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
IMD – INSTITUTO METRÓPOLE DIGITAL
Disciplina: IMD0033 – PROBABILIDADE – 24N12

ROTEIRO 2

Executando o básico com o R/Rstudio

1. Objetivo

Guiar o aluno no processo de utilização básica do R/Rstudio e auxiliar na compreensão do uso das funções necessárias para a sua utilização em análise descritiva de dados.

2. Expectativa

Apresentar os passos básicos para a utilização do R/Rstudio, para que o aluno possa construir mais um ambiente de estudo, em âmbito particular e com isso poder treinar e construir conceitos de análise estatística de dados bem como suas operações e aplicação.

3. Passos

1. Acessar o Help/ Buscando ajuda

O R possui um sistema de ajuda para auxilia-lo na busca de informações sobre funções e pacotes. Para pedir ajuda sobre determinada função, basta digitar '?' seguido do nome da função. Por exemplo, vamos pedir ajuda sobre a função sum()(soma).

```
?sum
```

Se estiver utilizando o RStudio, você verá que a ajuda aparecerá na aba help.
Se você quiser ver exemplos da função, basta fazer:

```
example(sum)
## sum> ## Pass a vector to sum, and it will add the elements together.
## sum> sum(1:5)
## [1] 15
## sum> ## Pass several numbers to sum, and it also adds the elements.
## sum> sum(1, 2, 3, 4, 5)
## [1] 15
## sum> ## In fact, you can pass vectors into several arguments, and everything gets added.
## sum> sum(1:2, 3:5)
## [1] 15
## sum> ## If there are missing values, the sum is unknown, i.e., also missing, .…
## sum> sum(1:5, NA)
## [1] NA
## sum> ## ... unless  we exclude missing values explicitly:
## sum> sum(1:5, NA, na.rm = TRUE)
## [1] 15
```

Você pode pedir ajuda sobre um tópico específico. Suponha que você deseja pesquisar sobre mapas, digite:

```
??maps
```

Você verá que uma série de tópicos relacionados serão exibidos na aba help.

2. Obtendo uma ideia do potencial com o comando demo()
3. Na utilização de bases de dados já cadastradas nos pacotes com o comando data()
4. Preparando o ambiente de trabalho com o comando getwd()

Uma das primeiras coisas a se fazer, antes de começarmos a utilizar o RStudio, é definir o diretório de trabalho. Para verificar o diretório atual, podemos executar o comando getwd() no console.

Defina sua pasta de trabalho utilizando a função setwd(). Após isso, crie um novo script indo em **File > New File > R Script**. O R script será utilizado para organizar todos os comandos que você deseja executar. O R executa os comandos na sequência do seu script. Linhas iniciadas com **#** serão ignoradas.

5. Criando variáveis/ armazenando objetos

```
x <- 3
x + 10
## [1] 13
```

No exemplo anterior, criamos uma variável x para armazenar o valor numérico 3. Ao realizarmos a operação x + 10, automaticamente o valor atribuído a x é recuperado. Para criarmos uma variável, utilizamos o símbolo <- ou =. No entanto, a notação <- é a mais utilizada**.** Faça o exemplo seguinte e analise os resultados.

```
x <- 25
x = 30
31 -> x
x <- x + 1
```

Os nomes das variáveis podem conter letras, números, pontos e sublinha (_), mas **não** podem iniciar com um número ou um ponto seguido de número.

6. Listando e removendo variáveis e objetos

Para listar todas as variáveis criadas, usamos a função ls(). Digite a função no console e verifique as variáveis disponíveis no momento. Para remover uma variável, usamos a função rm() com o nome da variável que desejamos remover. Vamos atribuir 3 à uma variável de nome k1 e depois vamos removê-la do nosso ambiente.

```
k1 <- 3
rm(k1)
```

Sempre que começamos a executar um novo script, é recomendável limpar todas as variáveis carregadas, visando evitar conflitos e outros problemas. Basta utilizar o comando:

```
rm(list = ls())
```

7. Vetores

O objeto que conheceremos agora se chama **vetor**. Na linguagem R, um vetor é uma sequência de dados de um mesmo tipo. Para criarmos um vetor no R a função utilizada é a c() e o processo é bastante simples. Vejamos um exemplo de criação de um vetor com os componentes {3, 23, 44} atribuído a uma variável y.

```
y <- c(3, 23, 44)
y
## [1]  3 23 44
```

Como já foi mencionado, os vetores armazenam dados de um mesmo tipo. Isso quer dizer que não podemos armazenar um dado numérico e um dado tipo texto (character) no mesmo vetor. Seguem alguns exemplos.

```
z <- c("João", "Paulo", "Pedro", "Francisco")
class(z)
## [1] "character"
k <- c(TRUE, FALSE, TRUE, TRUE)
class(k)
## [1] "logical"
```

Ao tentarmos criar um vetor com tipos de dados heterogêneos, o R converterá para character. O entendimento desse procedimento facilitará a detecção e a solução de alguns problemas no processo de manipulação e tratamento de dados.

```
k <- c(TRUE, 3, "Pedro")
class(k)
## [1] "character"
```

Agora que conhecemos um pouco de vetores, vamos criar uma variável tipo factor a partir de uma variável numérica. Suponha que você esteja diante de um conjunto de dados em que os gêneros feminino e masculino estejam representados pelos números 1 e 2, respectivamente. Podemos transformar essa informação em fator da seguinte forma:

```
genero <- c(1, 2, 1, 2)
genero <- factor(genero, levels = c(1, 2), labels = c("feminino", "masculino"))
genero
## [1] feminino  masculino feminino  masculino
## Levels: feminino masculino
```

Para mais detalhes, leia sobre a função factor().

✗ Indexando vetores

Uma vez que aprendemos como criar vetores, chegou o momento de aprendermos a manipulá-los e a realizar algumas operações. Vamos começar criando um vetor com cinco elementos numéricos e depois realizar algumas operações.

```
#Criando o vetor e atribuindo à variavél de nome "v"
v <- c(20, 12, 35, 19, 60)
```

Uma vez que temos nosso vetor v, podemos recuperar todos os valores de uma só vez ou apenas um ou alguns componentes desejados. A posição inicial de um vetor no R possui valor 1 e segue da esquerda para a direita. Portanto, se quisermos obter o valor contido na posição 2 do nosso vetor v, usamos a seguinte notação: v[2]. Vejamos:

```
v[2]
## [1] 12
```

Podemos também selecionar um intervalo de componentes do vetor, o que é bastante útil e rápido. Se quisermos selecionar os dados das posições 2 a 4 do nosso vetor, basta fazermos:

```
v[2:4]
## [1] 12 35 19
```

Para exibirmos todos os componentes do vetor **exceto** o de uma determinada posição, utilizamos o sinal – ao indexar o nosso vetor. O exemplo seguinte selecionará todos os elementos do nosso vetor v, **exceto o elemento da posição 4**.

```
v[-4]
## [1] 20 12 35 60
```

Como um último exemplo, vamos indexar nosso vetor para que retorne apenas as posições 2 e 5. Isso pode ser feito facilmente da seguinte maneira:

```
v[c(2, 5)]
## [1] 12 60
```

> **NOTA:** Há diferentes formas de indexarmos vetores e obtermos o mesmo resultado. Com o tempo, você vai descobrindo a forma mais usual e conveniente para chegar ao resultado desejado.

✗ Operações matemáticas e lógicas com vetores

Além de indexar, devemos também conhecer como é que o R interpreta algumas operações quando utilizamos vetores. Com um vetor contendo dados numéricos, podemos realizar diversas operações. Seguem exemplos utilizando o vetor v criado no tópico anterior.

```
V
## [1] 20 12 35 19 60
v * 3
## [1]  60  36 105  57 180
v + 1
## [1] 21 13 36 20 61
v / 7
## [1] 2.857143 1.714286 5.000000 2.714286 8.571429
```

Podemos observar que ao realizarmos qualquer operação entre um vetor numérico e um número, a operação é realizada para cada um dos componentes do vetor, gerando um vetor resultado.
Podemos também somar dois vetores, desde que um deles seja de **tamanho igual ou múltiplo do outro**. O exemplo a seguir esclarece.

```
j <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
w <- c(11, 12, 13, 14, 15)
#Soma dos vetores j e w
j + w
##  [1] 12 14 16 18 20 17 19 21 23 25
```

Ao realizar uma operação lógica com um vetor obtemos como resultado um vetor lógico. Vejamos uma operação com o vetor w do exercício anterior.

```
W >= 13
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

Esse vetor resultado é importante para filtrarmos elementos do nosso vetor. Suponhamos que desejamos selecionar os elementos do vetor w que sejam maiores ou iguais a 13. Podemos fazer isso de uma maneira muito simples, utilizando a expressão w >= 13 como indexador do nosso vetor w. Se quisermos saber em quais posições do vetor se encontram os elementos que satisfazem à condição, podemos utilizar a função which(). Vejamos:

```
w[w >= 13]
## [1] 13 14 15
which(w >= 13)
## [1] 3 4 5
```

8. Funções matemáticas e estatísticas importantes

Diversas funções matemáticas e estatísticas podem ser facilmente aplicadas a vetores numéricos. Vejamos algumas funções importantes abaixo.

Função	Descrição
sum()	Retorna a soma
mean()	Retorna a média

Função	Descrição
sd()	Retorna o desvio padrão
median()	Retorna a mediana
var()	Retorna a variância
cor()	Retorna a correlação entre dois vetores
min()	Retorna o mínimo
max()	Retorna o máximo
range()	Retorna o mínimo e o máximo
summary()	Retorna um sumário dos dados
quantile()	Retorna os quantis do conjunto numérico

> **NOTA:** Lembre-se que para ler sobre qualquer função no R basta digitarmos ? seguidos do nome da função no console.

Faça os exemplos seguintes e veja os resultados.

```
vetor1 <- c(1, 8, 11, 23.5, 12.8, 16, 25)
vetor2 <- c(10, 7.5, 10, 20.1, 24, 12, 25)
sum(vetor1)
mean(vetor2)
cor(vetor1, vetor2)
range(vetor1)
```

9. Mais sobre Data Frames

Em síntese, data.frames são tabelas de dados. Em seu formato, são bem parecidos com as matrizes, no entanto, possuem algumas diferenças significativas. Podemos idealizar os data.frames como sendo matrizes em que cada coluna pode armazenar um tipo de dado diferente. Logo, estamos lidando com um objeto bem mais versátil do que as matrizes e os vetores. Vejamos na prática!
Observe a tabela a seguir. Ela possui dados hipotéticos de seis universitários.

nome	altura	idade	sexo	peso	fumante	uf	renda
João	1.80	22	masculino	78.3	sim	PB	2
Pedro	1.77	21	masculino	82.1	não	AL	5
Amanda	1.71	18	feminino	66.5	sim	PE	10
Fábio	1.65	20	masculino	88.1	não	PE	20
Fernanda	1.66	23	feminino	58.0	sim	SP	10
Gustavo	1.63	19	masculino	75.4	não	CE	NA

Vamos criar um data.frame no R com esses mesmos dados. A função que usaremos é a data.frame(). Uma vez criado, armazenaremos esses dados na variável df1.

```
df1 <- data.frame(
  nome = c("João", "Pedro", "Amanda", "Fábio", "Fernanda", "Gustavo"),
  altura = c(1.80, 1.77, 1.71, 1.65, 1.66, 1.63),
  idade = c(22, 21, 18, 20, 23, 19),
  sexo = c("masculino", "masculino", "feminino", "masculino", "feminino", "masculino"),
  peso = c(78.3, 82.1, 66.5, 88.1, 58, 75.4),
  fumante = c(TRUE, FALSE, FALSE, FALSE, TRUE, FALSE),
  uf = c("PB", "AL", "PE", "PE", "SP", "CE"),
  renda = c(2, 5, 10, 20, 10, NA)
)
```

O primeiro ponto a ser observado é que nosso data frame foi criado através de vários vetores. Cada um dos vetores possui um determinado tipo de dado. Vamos exibir o nosso data.frame.

```
df1
```

nome	altura	idadesexo	peso	fumanteuf	renda
<fctr>	<dbl>	<dbl><fctr>	<dbl>	<lgl><fctr>	<dbl>
João	1.80	22masculino	78.3	TRUEPB	2
Pedro	1.77	21masculino	82.1	FALSEAL	5
Amanda	1.71	18feminino	66.5	FALSEPE	10
Fábio	1.65	20masculino	88.1	FALSEPE	20
Fernanda	1.66	23feminino	58.0	TRUESP	10
Gustavo	1.63	19masculino	75.4	FALSECE	*NA*

6 rows

Uma das funções básicas mais importantes para começarmos a trabalhar com data.frames é a str(). Essa função dá uma visão clara da estrutura do nosso objeto, bem como informa os tipos de dados existentes.

```
str(df1)
## 'data.frame':    6 obs. of  8 variables:
##  $ nome   : Factor w/ 6 levels "Amanda","Fábio",..: 5 6 1 2 3 4
##  $ altura : num  1.8 1.77 1.71 1.65 1.66 1.63
##  $ idade  : num  22 21 18 20 23 19
##  $ sexo   : Factor w/ 2 levels "feminino","masculino": 2 2 1 2 1 2
##  $ peso   : num  78.3 82.1 66.5 88.1 58 75.4
##  $ fumante: logi  TRUE FALSE FALSE FALSE TRUE FALSE
##  $ uf     : Factor w/ 5 levels "AL","CE","PB",..: 3 1 4 4 5 2
##  $ renda  : num  2 5 10 20 10 NA
```

Analisando o resultado da função, podemos verificar que nosso data frame possui 6 observações e 8 variáveis. As observações e variáveis nada mais são do que nossas linhas e colunas, respectivamente. Uma outra informação importante é saber o tipo de dado que cada variável (coluna) apresenta. Podemos facilmente constatar que quatro das nossas variáveis são numéricas, três são fatores e uma lógica.

10. Importação e armazenamento de dados

Para importar arquivos de texto para R, como .txt ou .csv, utilizaremos o pacote readr.

Veja alguns exemplos:

```
library(readr)
dados_txt <- readr::read_table2(file = "data/mtcars.txt")
## Error: 'data/mtcars.txt' does not exist in current working directory
('/home/travis/build/curso-r/pu.import').
dados_csv <- readr::read_csv(file = "data/mtcars.csv")
## Error: 'data/mtcars.csv' does not exist in current working directory
('/home/travis/build/curso-r/pu.import').
```

Também é possível salvar objetos, como data.frames em um tipo especial de arquivos, o .rds. A vantagem dessa extensão é guardar a estrutura dos dados salvos, como a classe das colunas de um data.frame. Além disso, é uma boa alternativa para lidar com grandes bancos de dados, já que arquivos .rds serão bem mais compactos do que arquivos Excel.

```
write_rds(mtcars, path = "data/mtcars.rds")
dados <- read_rds(path = "data/mtcars.rds")
```

A função read_excel() auto detecta a extensão do arquivo.

```
read_excel(path = "data/mtcars.xls")
## Error in read_excel(path = "data/mtcars.xls"): could not find function "read_excel"
read_excel(path = "data/mtcars.xlsx")
## Error in read_excel(path = "data/mtcars.xlsx"): could not find function "read_excel"
```

4. Referências

Comandos adaptados de http://material.curso-r.com/import/ e
https://bookdown.org/wevsena/curso_r_tce/curso_r_tce.html#preparando-o-ambiente-de-trabalho