Introdução à Análise de Dados com Linguagem R

1. Introdução

R é uma linguagem de programação gratuita de código aberto usada principalmente para computação estatística e gráficos. R é uma linguagem interpretada semelhante a Python, onde você não precisa compilar primeiro para executar seu programa. Depois de criar seu programa, você pode executá-lo em uma ampla variedade de plataformas UNIX, Windows e MacOS.

R é uma linguagem específica de domínio de código aberto, explicitamente projetada para ciência de dados. Muito popular em finanças e academia, R é uma linguagem perfeita para manipulação, processamento e visualização de dados, bem como computação estatística e aprendizado de máquina.

Como qualquer outra linguagem de programação, R também suporta extensão na forma de pacotes, portanto, os desenvolvedores podem criar seus próprios pacotes e reutilizá-los quando necessário.

2 Instalação do R

A instalação padrão da linguagem R é feita a partir do CRAN, uma rede formada de servidores espalhadod pelo mundo que armazena versões atualizadas do código fonte e executável (Windows), assim como a documentação da linguagem R.

3 Instalação do RStudio

RStudio é um ambiente de desenvolvimento integrado para linguagem R (IDE - *integrated development environment*), porém pode rodar *scripts* SQL, C, C++ e Python. A vantagem se se poder trabalhar com IDE é que ela disponibiliza ferramentas de apoio ao desenvolvimento de códigos em linguagem de programção. Para *download* do RStudio acesse o endereço https://www.rstudio.com/, e escolha a opção *RStudio Desktop*

3.1 Conhecendo a Interface Gráfica do RStudio

A interface do RStudio é divida em 4 painéis, e duas barras:

- 1 - Barra de *menu*
- 2 - Barra de ferramentas
- 3 - Painel de *scripts* e arquivos
- 4 - Painel de variáveis de Ambiente/Histórico/Conexões/Tutorial
- 5 - Painel de *Console/Terminal*
- 6 - Painel de Árvore de Arquivos/Gráficos/Pacotes/Ajuda/Visualizador

4. Obter e Configurar o ambiente de trabalho

Para obter o ambiente de trabalho atualmente em uso pelo RStudio utilizamos a função getwd(); esta função vem do termo em inglês: *Get Working Directory*, traduzido para o portugês como "Obter Ambiente de Trabalho". No sistema operacional Windows, por padrão, o RStudio configura o ambiente de trabalho em "C:/Usuários/Nome_do_Usuário/Documentos"

.

Para configurar um ambiente de trabalho diferente do padrão utilizamos a função setwd(). Esta função tem nome bem sugestivo na lingua inglesa, a saber: *Set Working Directory*, "Configurar Ambiente de Trabalho".

In [ ]:

Exercício Prático - Ambiente de trabalho

Instrução 1/2

- Use a janela console do RStudio para obter o atual diretório de trabalho em uso no seu computador.

In [ ]:

Instrução 2/2

- Configure seu ambiente de trabalho para `'C:/Users/Seu_nome_de_usario/Downloads'`

_Atenção: Em seu nome_deusuario *insira seu nome de usuário.*

In [3]:

In [ ]:

4.1 Operadores Aritméticos e de Atribuição em R

Operador	Função
+	Soma
-	Subtração
/	Divisão
*	Multiplicação
%%	Resto da divisão
%/%	Parte inteira divisão
^	Potenciação
**	Potenciação
<-	Atribuição
=	Atribuição

Exercício Prático - Operadores Aritméticos e de Atribuição

Instrução 1/3

- Obter o resto da divisão entre os números inteiros 10 e 3.

In [ ]:

Instrução 2/3

- Obter a parte interira da divisão de 10 por 3.

In [ ]:

Instrução 3/3

- Obter o quadrado de um número inteiro qualquer.

In [ ]:

5 Operadores de Comparação em R

Operador	Significado
==	igual a
!=	diferente de
>	maior que
<	menor que
>=	maior ou igual a
<=	menor ou igual a

6 Objeto em R

Um objeto é simplesmente qualquer variável que armazena um caractere numérico, alfabético ou uma cadeia de carateres (*string*). Em R temos objetos especiais para manipulação de grande volume de dados, a exemplo de vetores, listas e dataframes. Para criação de um objeto se utiliza o operador de atribuição `<-`. Nos exemplos a seguir são criados os objetos `x`, `y` e `z` para armazenar um número inteiro, uma letra e uma frase.

In [10]:
```
# definição de um objeto do tipo inteiro
int <- 42

# definição de um objeto do tipo caractere
letra <- "R"

# # definição de um objeto do tipo string (cadeia de caracteres)
string <- "R é massa"
```

Em ambos os trechos de código acima se observa o uso do caractere `#`, em R e python este caractere é utilizado para fazer comentário no código, portanto, toda linha que contém este caractere é ignorada na execução do programa.

A seguir é mostrado como fazer comentário com várias linhas.

In [ ]:
```
"
Este é um comentário em R utilizando
várias linhas para documentação de
códigos.
"
```

6.1 Conferindo o conteúdo de um objeto

Para conferir o conteúdo de um objeto em R, fazemos uma chamada diretamento pelo nome do objeto de interesse ou através do uso da função *`print()`*

In [11]:
```
int
print(int)
cat(int)
```

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

6.2 Descobrindo o tipo de dados armazenado em um objeto R

Para descobrir o tipo de dados armazenado em um objeto, podemos utilizar a função *`class`*

In [12]:
```
class(int)
class(letra)
class(string)
```

'numeric'
'character'
'character'

7 Estrutura de Dados em R

7.1 Vetores

Vetor em R é um objeto R que armazena um ou mais elementos de tipos de valores indexados, ou seja, cada elemento dentro do vetor possui uma posição específica. Para criação de um vetor basta colocar os valores dentro de `c()`. Vetor é uma estrutura de dados especialmente importante em análise de dados. A seguir temos um exemplo de como criar um vetor de inteiros.

In [13]:
```
inteiros <- c(42, 33, 0, -1, 5)
```

Para acessar o primeiro elemento do vetor `inteiros` usamos o comando *`vetor[x]`*, onde *vetor* é nome atribuido ao vetor e *x* é o índice do elemento a ser buscado no vetor. Se quisermos mostrar apenas o elemento de índice 1 do vetor `inteiros`, ou seja: filtar apenas o primeiro elemento, basta usar *`inteiros[1]`*

In [14]:
```
# acessar o primeiro elemento de um vetor
inteiros[1]
```

42

Ao usar um índice negativo para filtrar um vetor estamos pedindo que a saída dos daos do vetor não mostre o elemte relativo ao índice negativo. No código a seguir vamos mostrar todos os elementos do vetor `inteiros`, exceto o quarto elemento (elemento de índice 4).

In [15]:
```
# mostrar todos os elementos do vetor com exeção do elemento de índice 4
inteiros[-4]
```

42 · 33 · 0 · 5

7.1.2 Substituição de elementos de um vetor

In [16]:
```
# Substituir o primeiro elemento do vetor "inteiros" por 2
inteiros[1] <- 2
inteiros
```

2 · 33 · 0 · -1 · 5

7.2 Funções básicas aplicadas a vetores

- *`length()`* Retorna o tamanho de um vetor, ou seja, o número de elementos armazenados no vetor.

In [17]:
```
length(inteiros)
```

5

- *`names()`* Retorna os nomes atribuídos a cada elemento de um vetor

In [20]:
```
# Defini um vetor de números inteiros chamado "dias_semana"
dias_semana <- c(1:7)

# Mostrar na tela do console os dados do vetor
dias_semana

# Checar se o vetor possui nomes associados aos seus elementos
names(dias_semana)
```

1 · 2 · 3 · 4 · 5 · 6 · 7
```
NULL
```

In [21]:
```
# Atribuir nomes aos elementos do vetor utilizando a função names()
names(dias_semana) <- c('Domingo', 'Segunda', 'Terça', 'Quarta',
                        'Quinta', 'Sexta', 'Sábado')
```

In [22]:
```
# Checar se o vetor possui nomes associados aos seus elementos
names(dias_semana)
```

'Domingo' · 'Segunda' · 'Terça' · 'Quarta' · 'Quinta' · 'Sexta' · 'Sábado'

In [23]:
```
# Mostrar na tela do console os dados do vetor
dias_semana
```

Domingo: 1 **Segunda:** 2 **Terça:** 3 **Quarta:** 4 **Quinta:** 5 **Sexta:** 6 **Sábado:** 7

- *`attributes()`* Retorna uma lista com os atributos associados a um vetor.

In [24]:
```
# Verificar se os vetores "inteiros" e "dias_semana"
# possuem algum atributo
attributes(inteiros)
attributes(dias_semana)
```

```
NULL
```
$names =
'Domingo' · 'Segunda' · 'Terça' · 'Quarta' · 'Quinta' · 'Sexta' · 'Sábado'

In [ ]:

- *`seq()`* Cria uma sequência dentro de um vetor

In [25]:
```
# gerar uma sequência de 1 a 10, saltando 2 números
sequencia <- seq(0, 10, 2)
print(sequencia)
```

```
[1]  0  2  4  6  8 10
```

- *`rep()`* Cria uma repetição de um vetor

In [26]:
```
# Gerar uma repetição de três vezes o vetor formado pelos número de 1 a 4
rep(1:4, 3)
```

1 · 2 · 3 · 4 · 1 · 2 · 3 · 4 · 1 · 2 · 3 · 4

- *`duplicated()`* Mostra a localização de elementos duplicados.

In [27]:
```
# Vetor com números inteiros duplicados
vetor <- c(1, 2, 1, 3, 4, 5, 4)

# Mostrar os números duplicados
duplicated(vetor)
```

FALSE · FALSE · TRUE · FALSE · FALSE · FALSE · TRUE

Por padrão a saída da função `duplicated()` é um vetor lógico. O código abaixo mostra com seria a saída em forma de vetorial para os dados acima.

In [28]:
```
# Mostrar apenas os números repetidos
vetor[duplicated(vetor)]
```

1 · 4

In [33]:

- *`unique()`* Retorna apenas os valores distintos.

A seguir utilizaremos os dados dos vetores `dap` (diâmetro a altura do peito), categoria, altura e nomes_cientifocos, os quais são parte de um inventário florestal realizado na Unidade de Manejo Florestal 4 da Floresta Nacional de Altamira.

In [34]:
```
# Leitura de vetores de um inventário florestal
load('./dados/dados_modulo_1.rda')

# Mostrar os objetos atualmente disponíveis no ambiente R
ls()
```

'altura' · 'categoria' · 'dap' · 'dias_semana' · 'int' · 'inteiros' · 'letra' · 'nomes_cientificos' · 'sequencia' · 'string' · 'vetor'

In [ ]:

A seguir mostraremos quantas árvores foram inventariadas, ou seja, o tamnaho do nosso vetor.

In [35]:
```
length(nomes_cientificos)
```

18406

O a saída código acima mostrar que foram inventariadas 18.406 árvores.

A seguir mostraremos a quantas espécies estas árvores pertencem, e para tal as funções `length()` e `unique()`.

In [41]:
```
# Vetor apenas com a relação distinta de espécies inventariadas
especies <- unique(nomes_cientificos)

# somar o vetor especies
length(especies)
```

67

Portanto, temos a que para a área do inventário ocorrem 67 espécies de interesse comercial. O código acima poderia ser resumido em apenas uma linha, vejamos o exemplo a seguir:

In [42]:
```
length(unique(nomes_cientificos))
```

67

7.3 Funções Estatísticas Aplicadas a Vetores

- *`mean( )`* Retorna a média aritmética
- *`median( )`* Retorna a mediana
- *`min( )`* Retorna o menor valor
- *`max( )`* Retorna o maior valor
- *`sd( )`* Retorna o desvio padrão
- *`summary( )`* Retorna a estatistiva descritiva.
- *`cor( )`* Retorna a correlação entre dois vetores.

Calcular a média do vetor *dap* (diâmetro médio das árvores)

In [43]:
```
# média do vetor dap (diâmetro médio das árvores)
mean(dap)
```

72.2728159295882

Calcular a mediana do vetor *altura* (altura comercial das árvores)

In [44]:
```
### Calcular a mediana do vetor _altura_ (altura comercial das árvores)
median(altura)
```

18

Mostrar os valores mínimo e máximo do vetor dap

In [45]:
```
# valor mínimo de dap
min(dap)
```

40

In [46]:
```
# dap Máximo
max(dap)
```

312.26

Calcular o desvio padrão do vetor dap

In [47]:
```
# desvio padrão para o vetor dap
sd(dap)
```

24.0258756077791

Mostrar a estatística Descritiva do Vetor altura

In [48]:
```
summary(altura)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   7.00   16.00   18.00   18.14   20.00   40.00 
```

Calcular a correlação linear entre diâmetro e altura das árvores

In [49]:
```
cor(dap, altura)
```

0.273168903939106

7.4 Função `lapply()` aplicada a vetores

A função lapply, parte do pacote base do R, no caso específico de vetores, recebe 2 argumentos como parâmetro: o vetor contendo de dados e uma função a ser aplicada aos elementos do vetor.

In [51]:
```
nomes <- c('MASSARANDUBA', 'IPÊ', 'GARAPEIRA', 'JATOBÁ')
nomes
```

'MASSARANDUBA' · 'IPÊ' · 'GARAPEIRA' · 'JATOBÁ'

In [52]:
```
lapply(nomes, tolower)
```

1. 'massaranduba'
2. 'ipê'
3. 'garapeira'
4. 'jatobá'

7.5 Função `sapply()`

In [53]:
```
sapply(nomes, tolower)
```

MASSARANDUBA: 'massaranduba' **IPÊ:** 'ipê' **GARAPEIRA:** 'garapeira' **JATOBÁ:** 'jatobá'

7.6 Função `mapply()`

Versão multivariada das funções lapply e sapply, utilizada para iterar entre elementos de vetores ou listas.

In [56]:
```
# Definição dos Vetores a e b
a <- c(7, 12, 5, 2, 1)
b <- c(4, 2, 3, 5, 1)

# Nomes para os vetores
dias_semana <- c('Segunda', 'Terça', 'Quarta', 'Quinta', 'Sexta')

# Atribuir nome aos vetores
names(a) <- dias_semana
names(b) <- dias_semana

# Uso da função mapply() para retornar a soma
# entre os elementos dos vetores a e b
mapply(max, a, b)
```

Segunda: 7 **Terça:** 12 **Quarta:** 5 **Quinta:** 5 **Sexta:** 1

7.7 Função *`tapply()`*

Aplica uma função sobre um vetor com agrupamento em outro vetor categórico. Recebe como parâmetros: um vetor numérico, um vetor categórico e uma função. O código a seguir aplica a função média sobre o vetor `volume` agrupado ao vetor `ut` (unidades de trabalho)

In [57]:
```
# Calcular o volume médio por unidade de trabalho
tapply(dap, categoria, mean)
```

Explorar: 76.8872521591047 **Remanescente:** 67.9580670429674 **Substituta:** 71.9543683510638

8 Valores Ausentes (NA)

Em R valores ausentes são conhecidos como `NA`, uma sigla em inglês que significa *Not Available*, ou seja, valores nãop disponíceis. Na literatura técnica e também em outras linguagens de programação esta sigla é definida como `Nan` (*Not available number*)

In [58]:
```
# Definição do Vetor "num" com elemento "NA"
num <- c(2, 11, 25, NA, 45)
```

Para o caso do vetor acima se utilizarmos a função `mean` para calcular a média aritmética do vetor `num`, teriamos que dizer a função para desconsiderar o valor NA, o que se faz definindo o parâmetro `na.rm = TRUE`, vejamos o exemplo a seguir:

In [59]:
```
# Uso da função mean() sem desconsiderar valor ausente (NA).
mean(num)
```

<NA>

In [60]:
```
# Desconsiderar valores NA
mean(num, na.rm = TRUE)
```

20.75

Como Saber se Há Valores Ausentes (NA) nos Dados?

Para conferir a presença de valores NA nos dados utilizamos a função `is.na()`, a qual recebe como parâmetro de entrada apenas o vetor de dados.

- `is.na( )` Testa se o vetor contém valores ausentes (*Not Availables*)

In [63]:
```
vetor <- c(NA, 2, 3, 6)
is.na(vetor)
```

TRUE · FALSE · FALSE · FALSE

Vemos acima que o retorno da função `is.na()` retorna um vetor lógico, mostrando *TRUE* sempre que o elemento do vetor é do tipo `NA`. Podemos melhor a saída acima para uma forma tabular através do uso da função `summary`

In [64]:
```
summary(is.na(vetor))
```

```
   Mode   FALSE    TRUE 
logical       3       1 
```

Poderiamos ainda filtar o vetor de forma a não mostrar valores ausentes, usando o perador de negação ou "diferente", qual seja `!`. Este operador tem a mesma função da negação utilizada em lógica matemática porém nesta área utiliza-se os caracteres `¬` e `~`

In [65]:
```
vetor[!is.na(vetor)]
```

2 · 3 · 6

Exercício Prático - Vetores e Operadores de Comparação

Para este exercício, considere o vetor temperatura. Esse vetor possui dados de temperatura média mensal da Estação Meteorológica Manual INMET 82861, localizada no município de Conceição do Araguaia.

In [ ]:
```
temperatura <- c(26.38452, 26.90357, 27.04064, 27.42467,
                 28.53548, 28.90000, NA, 29.73818,
                 30.54667, 27.21652, 27.28800, 27.84000)
```

Instrução 1/4

- Obter a temperatura média do vetor `temperatura`

In [ ]:

Instrução 2/4

- Obter as temperaturas que estão acima da média do vetor `temperatura`

In [ ]:

Instrução 3/4

- Mostre quanto dos dados do vetor de temperaturas apresentam valores `NA`

In [ ]:

Instrução 4/4

- Mostre onde os dados do vetor de temperaturas apresenta valores `NA`

In [ ]: