2. R Intermediário

2.1 Pacotes

Para instalar um determinado pacote (biblioteca) em R, utilizamos a função *install.packages("nome_do_pacote")*, a qual recebe como parâmetro o nome do pacote a ser instalado. Utilizaremos como exemplo a instalação do pacote dplyr, o mais importante pacote para manipulação e transformação de *dataframes* em R.

Para carregar um pacote em R fazemos uso da função library(), fornecendo como parâmetro o nome do pacote

2.2 Instalar e Carregar Pacotes

Exercício Prático

Instrução 1/3

- Instale o pacote `dplyr` utilizando a janela do console

In [ ]:

Instrução 2/3

- Instale o pacote `readr` utilizando a janela Packages.

In [ ]:

Instrução 3/3

- Carregue o pacote `dplyr`

In [ ]:

3.3 Listar todos os Pacotes instalados

In [ ]:
```
available.packages()
```

3.4 Mostrar os pacotes carregados na sessão atual

In [ ]:
```
search()
```

4. Datas

Neste capítulo entenderemos como as datas e horas são criadas em linguagem R. Este capítulo vai lhe ensinar o suficiente para começar a trabalhar com datas. R tem muito a oferecer em termos de datas e horários. As duas principais classes de dados para isso são *Date* e *POSIXct*. Date é usado para objetos de data do calendário como "2015-01-22". *POSIXct* é uma maneira de representar objetos de data e hora como "2015-01-22 08:39:40 EST", o que significa que são 40 segundos após 8:39 AM Eastern Standard Time.

Na prática, a melhor estratégia é usar a classe mais simples que você precisa. Frequentemente, Date será a escolha mais simples. Este curso usará quase exclusivamente a classe Date, mas é importante estar ciente do *POSIXct* também para armazenar dados financeiros diários.

4.1 Obter a Data Atual

In [2]:
```
# Data Atual
Sys.Date()
```

2023-01-15

4.2 Obter a Data e Hora Atuais

In [3]:
```
# Data e Hora
Sys.time()
```

[1] "2023-01-15 17:49:45 -03"

4.3 Como Criar Data

Para criarmos um data a partir de um vetor de caracteres, utilizamos a função `as.Date()`

In [4]:
```
# caracter com dado referente a data da crise de 1929
crise_29 <- "1929-11-29"

data <- as.Date(crise_29)
```

In [ ]:
```

```

4.4 Várias Datas

In [ ]:
```

```

4.5 Formatos de Data

Como você viu anteriormente, R é exigente sobre como ele lê as datas. Para lembrá-lo, se executarmos o código `as.Date("28/09/2008")`, a saída no terminal será um erro porque não está no formato correto. A correção para isso é especificar o formato que você está usando por meio do argumento de `format`:

In [6]:
```
as.Date("09/28/2008", format = "%m/%d/%Y")
```

2008-09-28

Existem vários formatos diferentes que você pode especificar quando estiver trabalhando com data, aqui estão alguns deles:

- `%Y` : 4-digit year (1982)",
- `%y` : 2-digit year (82),
- `%m` : 2-digit month (01),
- `%d` : 2-digit day of the month (13),
- `%A` : weekday (Wednesday),
- `%a` : abbreviated weekday (Wed),
- `%B` : month (January),
- `%b` : abbreviated month (Jan)

4.6 Aritimética com Datas

Assim como com numérico, aritmética pode ser feita em datas. Em particular, você pode encontrar a diferença entre duas datas, em dias, usando a subtração:

In [7]:
```
hoje <- Sys.Date()
amanha <- hoje + 1
hoje
amanha
```

2023-01-15
2023-01-16

4.6.1 Função `difftime()`

In [ ]:
```

```

4.7 Meses, Semanas e Trimestre

In [ ]:
```

```

5. Operadores Lógicos

In [ ]:
```

```

6. Instruções if, else e else if

In [ ]:
```

```

7. Loops

8 Funções em R

A definição de uma função em R segue a seguinte sintaxe:

In [1]:
```
nome_da_função <- function (parâmetros) {
    return()
}
```

A função definida a seguir, denominada como `duplicar_valor`, recebe como parâmetro um valor `x`, definido por padrão como 2, e retorna o quadrado deste valor.

In [2]:
```
duplicar_valor <- function(x = 2) {
        return(x ^ 2)
}
```

Após a criação da função devemos invocá-la fornecendo o valor de seu parâmetro para recebermos seu retorno.

In [3]:
```
duplicar_valor()
```

4

In [4]:
```
duplicar_valor(2)
```

4

Exercício Prático 1 - Funções

O índice de massa corporal (IMC) é uma medida internacional usada para calcular se uma pessoa está no peso ideal. Neste exercício você deverá criar uma função para calcular o IMC e modificá-la para dizer ao usuário de ele está com o peso normal (entre 18,5 e 24,9), sobre peso (25,0 e 29,9), obeso (30,0 e 39,9) ou magro demais (menor que 18,5). O IMC é calculado a partir da seguinte fórmula:

$$IMC = \frac{Peso}{(Altura)^2}$$

Instrução 1/2

- Crie uma função chamada `imc` para calcular o índice de massa corporal. Esta função deve receber como parâmetros: `peso` (kg) e `altura` (m).

In [ ]:
```

```

Instrução 2/2

- Agora melhore a função de forma a informar ao usuário sua situação de acordo com o IMC calculado.

In [ ]:

Exercício Prático 2 - Funções

O diâmetro de árvores calculado a partir da circunferência do caule. utilizando-se da seguinte fórmula:

$$diametro = circumferencia \times \pi$$

In [ ]:

Instrução

- Crie uma função chamada `d` que recebe como parâmetro a circunferência de uma árovre e retorna o seu diâmetro.

In [ ]:

5.1 Escopo de variáveis

No código a seguir temos a definição da variável `x` recebendo o valor inteiro `4`

In [5]:
```
x <- 4
```

Agora ao chamarmos a função `duplicar_valor()`, que valor deverá nos retornar, 4 ou 16?

In [6]:
```
duplicar_valor()
```

4

A função retorna 4, pois o valor da variável `x` definida na função é uma variável local e com o valor padrão 2, a qual é restrita ao escopo da função em questão, apesar de ´x´ definido fora da função ser uma variável global. O conceito de variável global fica melhor compreedido com o uso da seguinte função:

In [7]:
```
txt <- 'Olá R'

saida_txt <- function () {
    return(txt)
}
```

Como a função acima não tem a variável `txt` localmente definida em seu escopo, retorna a variável global `txt` definida fora de seu escopo, pois esta é uma variável global a qual está disponível em todo o ambiente de variáveis do R. Podemos definir uma variável local de uma função como global, a qual passará a ser global após a chamada da função. Para tal devemos utilizar o operador de atribuição `<<-`

In [13]:
```
# variavel global fora da funcao
msg <- 'Eu sou global fora e estou fora de função!!'
```

```
# Funcao com variavel local definida como global
f <- function () {
        msg <<- 'Eu sou uma variável local e global!! kkkkk'
        return(msg)
}
```

In [14]:
```
msg
```

'Eu sou global fora e estou fora de função!!'

In [15]:
```
f()
```

'Eu sou uma variável local e global!! kkkkk'

In [16]:
```
# Funcao com variavel local definida como global
f <- function () {
        msg <<- 'Eu sou uma variável local e global!! kkkkk'
        return(msg)
}
```

In [17]:
```
msg
```

'Eu sou uma variável local e global!! kkkkk'

In [ ]: