
Aula 1: apresentando o R

Marcelo Prudente

9 de janeiro de 2020

Sumário

	Página
1 Objetivo do curso	4
1.1 Objetivo	4
1.2 Roteiro do curso	4
1.3 Material do curso	5
1.4 Leituras recomendadas	5
2 Porque usar o R?	6
2.1 Vantagens de utilizar o R	6
2.2 Desvantagens	6
3 R	7
3.1 R como linguagem de programação	7
3.2 O ambiente do R	7
3.3 Como é o ambiente do R?	8
3.4 O ambiente do R Studio	8
3.5 Console	9
3.6 Script	10
3.7 Arquivos, gráficos, pacotes e ajuda	11
4 Pacotes	12
4.1 Pacotes do CRAN	12
4.2 Utilizando os pacotes	13
4.3 Ambiente global	14
5 Obtendo Ajuda	14
5.1 Ajuda	14
5.2 Ajuda na internet	15
6 Criando Projetos no R	15
7 Estrutura de Dados	15
7.1 Introdução	15
7.2 Vetores Atômicos	16
7.3 Exemplo 1: criando vetores	16
7.4 Exemplo 1: criando vetores	17
7.5 Exemplo 1: testando a natureza dos vetores	17

7.6	Exercício 1	17
7.7	factor	18
7.8	Comandos matemáticos no R	19
7.9	Exemplos de comandos matemáticos	19
7.10	Outros comandos úteis	20
7.11	Aplicando multiplas funções de uma vez	21
7.12	Operadores Lógicos	22
7.13	Operações com Conjuntos	22
7.14	Sequências e repetições	24
8	Matrizes	25
8.1	Matrizes	25
8.2	Matrizes	25
8.3	Matrizes (exemplo 1)	25
8.4	Matrizes (exemplo 2)	25
9	Listas	27
9.1	Listas	27
10	Data Frame	27
10.1	Data Frame	27
10.2	Data Frame	27
10.3	Data Frame	28
10.4	Data Frame: acessando elementos com o \$	29
10.5	Data Frame: acessando seus elementos com os []	29
10.6	Data Frame: adicionando elementos	30
11	Do Data.Frame para Tibble	30
11.1	Tibble: um data.frame melhorado	30
11.2	Tibble: visualizando os dados	31
11.3	Tibble: criando dados reciclando vetores	31
11.4	Tibble: criando dados reciclando vetores	31
11.5	Exercício 5	32

1 Objetivo do curso

1.1 Objetivo

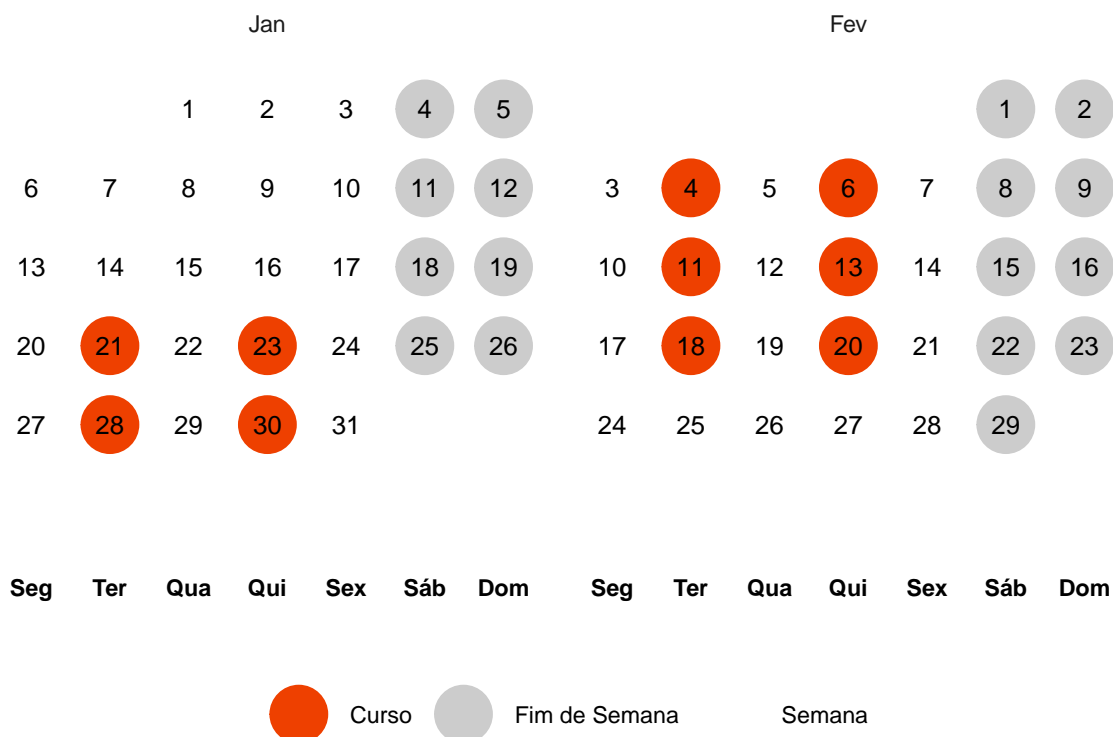
- Aprender pelo exemplo a manipulação de bases de dados no software **R**:
 - foco no domínio da sintaxe
 - exemplos e exercícios
- Migrar análises do excel para o R
 - reproducibilidade
 - transparência

1.2 Roteiro do curso

O curso terá duração de *4 semanas*. Nesse período, cobriremos os seguintes tópicos:

- Apresentação do ambiente R
- Estrutura dos dados
- Importação de arquivos em diferentes formatos
- Manipulação de dados
 - verbos do pacote *dplyr*
 - variáveis categóricas com o pacote *forcats*
 - organização dos dados com o pacote *janitor*
- Análise exploratória dos dados com o pacote *ggplot2*
- Análise da dados da PNAD Contínua com o pacote *PNADcIBGE*
- Acesso remoto a bases do IPEA Data com o *ipeadataR*
- Muitos exercícios

Calendário do curso – 2020



1.3 Material do curso

- **Acessar site do curso:** https://github.com/marceloprudente/curso_R_SOF/
 - dados básicos
 - exercícios
 - apresentações
 - scripts das aulas
 - *cheatsheets*

1.4 Leituras recomendadas

- Há diversos livros introdutórios e gratuitos ao **R**:
 - R for Data Science
 - Exploratory Data Analysis with R
 - R para cientistas sociais
 - Advanced R

- O site *curso-r* também é uma boa fonte de informações para o aprendizado da linguagem.

2 Porque usar o R?

Algumas razões para utilizar o **R** são elencadas *nesse post*.

2.1 Vantagens de utilizar o R

- É uma ferramenta aberta (de graça);
- Tem funcionalidades do Excel, porém é mais poderoso;
 - Além disso, é uma ferramenta estatística e de programação;
- **Gráficos e mapas** com excelente qualidade de publicação;
- Faz análises de dados e estatísticas facilmente.
- Ampla comunidade de contribuidores;
- Curadoria dos pacotes pelo CRAN
- É fácil reproduzir os comandos para outros bancos de dados (maior transparência nas análises);
- É possível abrir inúmeros bancos de dados ao mesmo tempo;
- **Rmarkdown**: linguagem de texto integrada ao R + elaboração de relatórios + elaboração de slides
- Ferramenta amplamente utilizada na análise dados em todo o mundo;

2.2 Desvantagens

- Por ser livre, não há garantias legais:
 - o CRAN atua como uma espécie de moderador do sistema
- Não lida bem com bases grandes (carrega dados na memória):
 - pacote `data.table` - desenhado para lidar com bases grandes
 - MonetDB e MonetDBLite() - soluções para conexão com bases grandes
- Curva de aprendizado acentuada (não será o caso para quem está aqui)

3 R

3.1 R como linguagem de programação

- O **R** é uma linguagem de programação voltada para a computação estatística;
- Acessamos o programa por meio de *linhas de comando*
- A abordagem de programação permite:
 - automatizar operações
 - documentar as operações realizadas (transparência e integralidade)

3.2 O ambiente do R

- O **R** é um conjunto integrado de instalações de softwares para manipulação de dados, cálculo e apresentação de gráficos.
 - não é apenas um programa estatístico
 - os pacotes do R podem expandir bastante as funcionalidades do programa
 - é possível escrever suas próprias funções

3.3 Como é o ambiente do R?

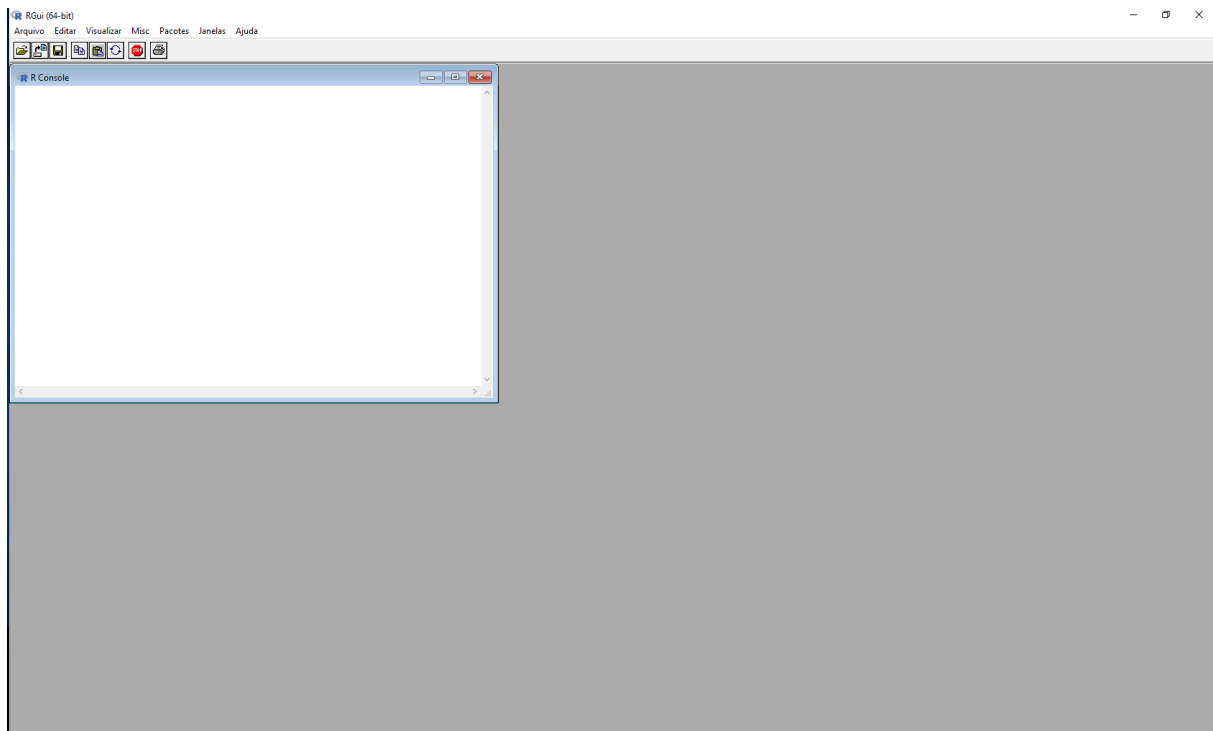


Figura 1: Ambiente do R

- Por isso, utilizaremos o **R Studio**

3.4 O ambiente do R Studio

O **R Studio** integra o **R** como um Ambiente de Desenvolvimento Integrado, que apresenta maior funcionalidade. Por exemplo:

- Tem ferramentas que realçam a sintaxe e completam códigos;
- Executa códigos diretamente do editor

Para utilizar o **R Studio** é necessário ter o **R** previamente instalado no computador.

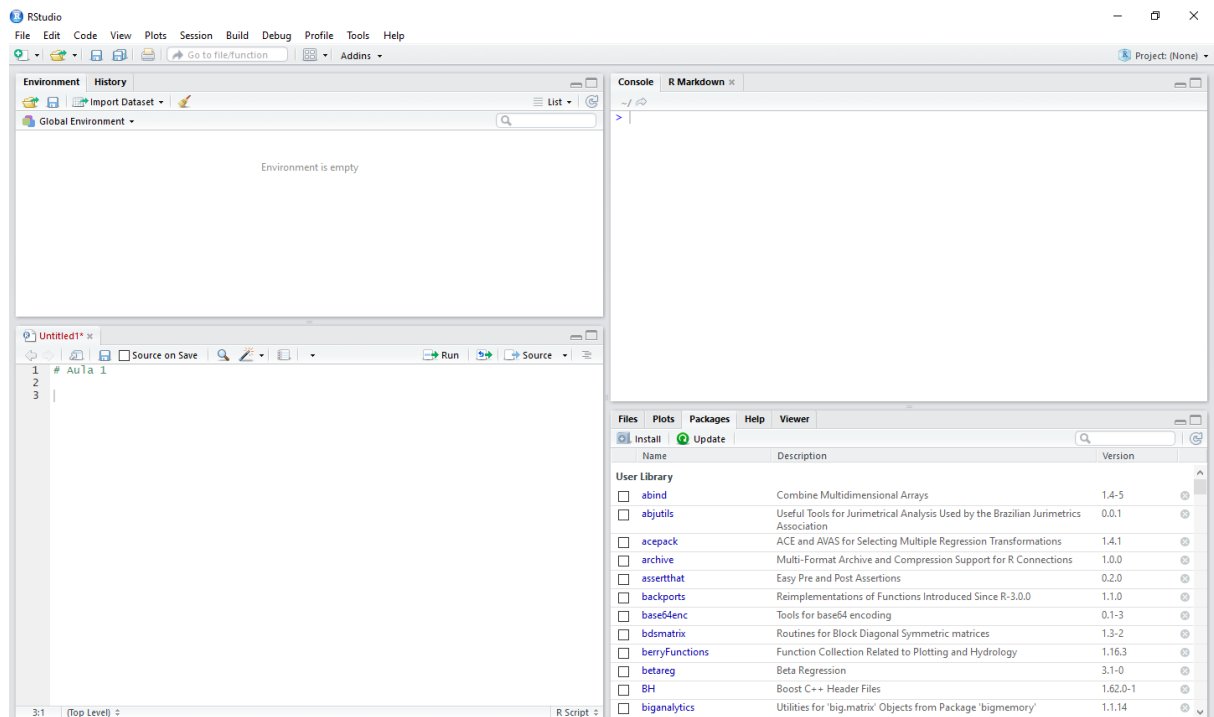


Figura 2: Ambiente do R Studio

3.5 Console

- No *console* digitamos as nossas linhas de comando para o programa. Podemos realizar operações matemáticas, estatísticas, descrever funções entre outros.
- O símbolo **>** indica que podemos digitar comandos ao programa.
- Em seguida, pressionamos **ENTER** para executá-los.

Podemos executar o seguinte comando no console:

```
50 * 15
```

```
## [1] 750
```

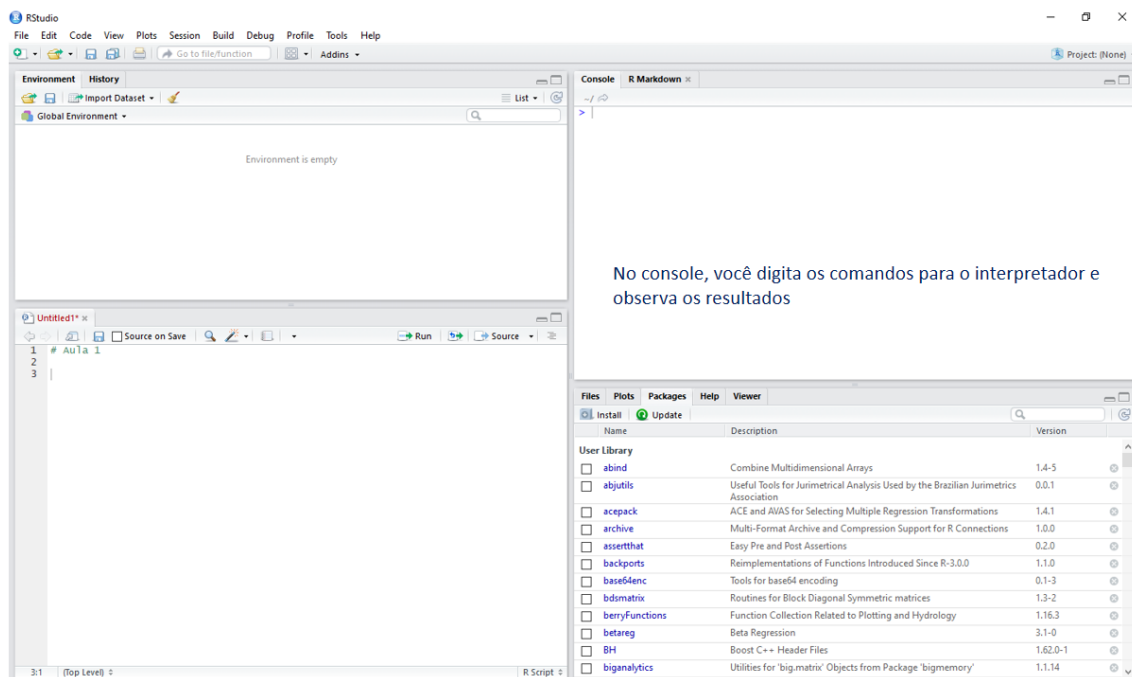


Figura 3: Console do R Studio

3.6 Script

- Nos *scripts* passamos instruções que serão processadas no *console*.
- Porém, nesse caso gravamos os comandos em um arquivo de texto (.txt) que pode ser executado a qualquer momento:
 - reprodutibilidade + transparência!
 - capacidade de executar diversas funções em projetos grandes
- Para executar um código do script, basta utilizar o atalho **Ctrl + Alt + ENTER**

Vamos digitar algumas linhas no *script* e, em seguida, selecionar os comandos e utilizar o atalho:

```
5 + 10
```

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

```
"Boa tarde, pessoal"
```

```
## [1] "Boa tarde, pessoal"
```

- Os scripts permitem comentar os códigos com a **#**

- reproducibilidade !

```
# Vamos escrever um texto
```

```
"Para escrevermos um texto no R, precisamos das aspas"
```

```
## [1] "Para escrevermos um texto no R, precisamos das aspas"
```

- Os comentários podem demarcar sessões com o uso de quatro -

```
# Sessão 1: soma ----
```

```
1 + 1
```

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

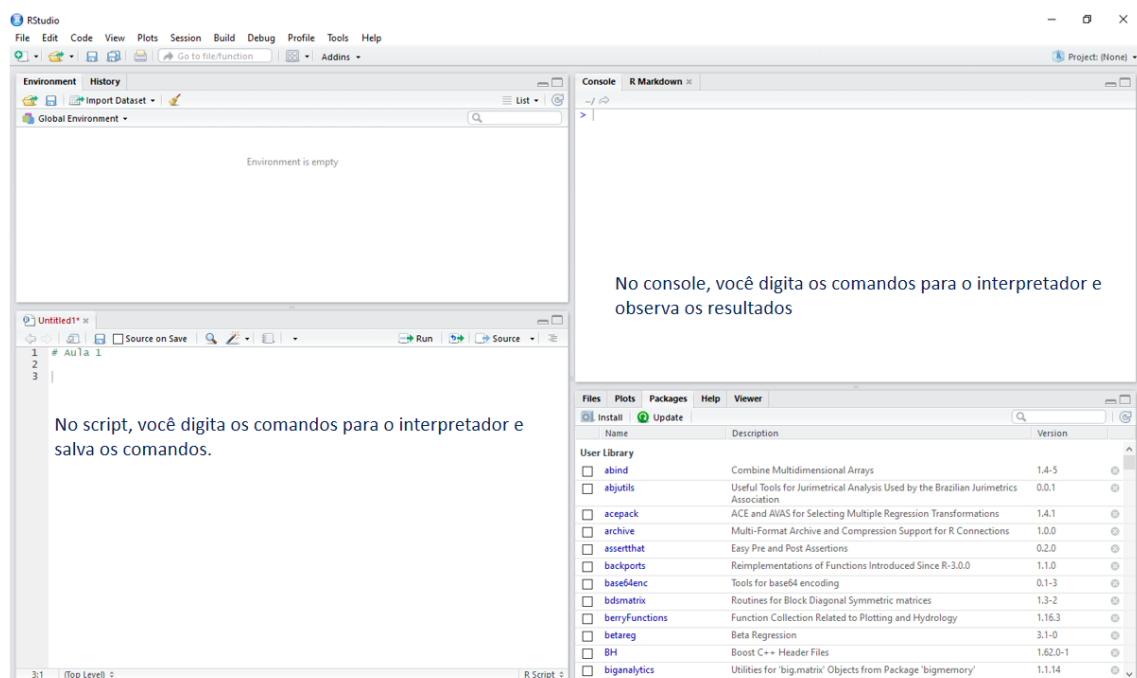


Figura 4: Console do R Studio

3.7 Arquivos, gráficos, pacotes e ajuda

- A última parte do ambiente do **R** apresenta:
 - **Arquivos:** da pasta que estiver especificada no sistema + getwd()
 - **Plots:** mostra os gráficos eventualmente feitos
 - **Pacotes:** apresenta os pacotes instalados em nosso **R**
 - **Ajuda:** mais informações sobre o sistema

4 Pacotes

4.1 Pacotes do CRAN

- Os pacotes do **R** empacotam(!!) códigos, dados, documentação e testes!
- Esses pacotes podem ser encontrados no CRAN ou no github
 - Há pelo menos 15140 pacotes só no CRAN (quando esse texto foi escrito)
- Para instalar pacotes do CRAN, basta ter conexão com a internet e digitar no *console* ou *script*:

```
# Instalando o pacote dplyr  
install.packages("dplyr")
```

- Ou ainda, ir para a aba *packages* e, em seguida, em *install*

4.1.1 Pacotes do GitHub

- Há também pacotes disponíveis no github. Alguns desses pacotes ainda não foram aprovados pelo CRAN. Em geral, recomenda-se a instalação de pacotes verificados, pois são mais estáveis. De qualquer modo, caso queira instalar os pacotes não validados, é necessário ter instalado o pacote *devtools* e seguir os comandos a seguir:

```
install.packages("devtools")  
devtools::install_github("tidyverse/readr")
```

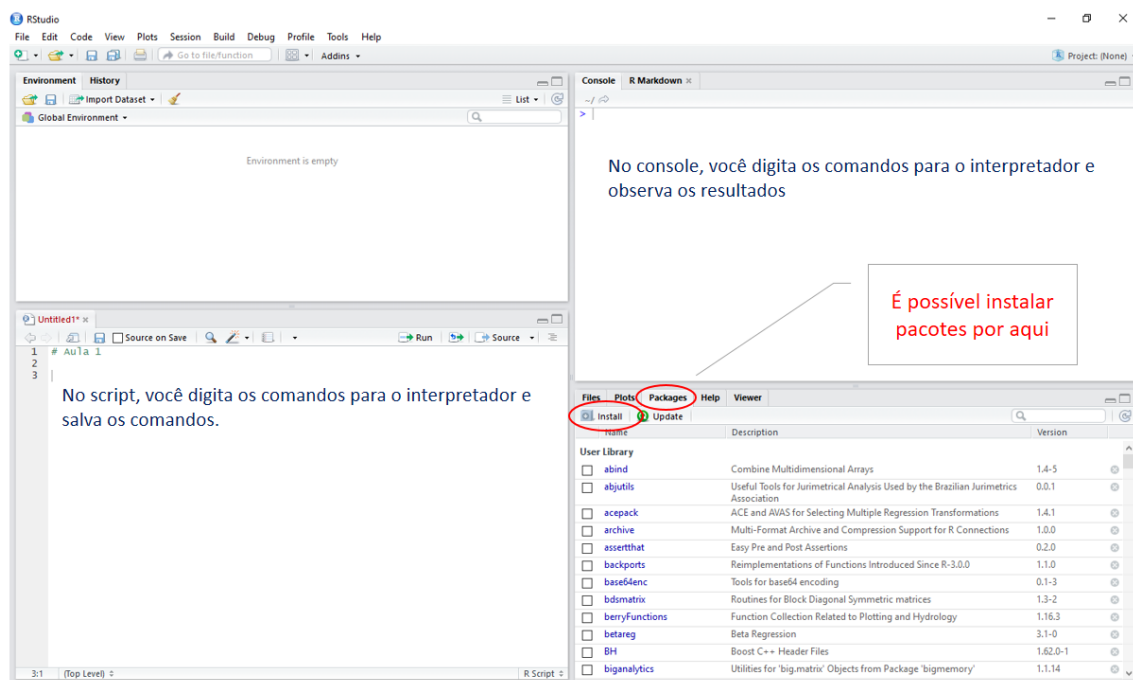


Figura 5: Console do R Studio

4.2 Utilizando os pacotes

- Há três formas de colocar os pacotes em uso no R:
 - Clicando diretamente na caixa ao lado do nome do pacote
 - Ou utilizando dois comandos similares:

```
# Primeira forma  
require(dplyr)
```

```
# Segunda Forma  
library(dplyr)
```

- Há também uma forma ágil de baixar diversos pacotes por meio do pacote xfun.

```
install.packages("xfun")  
library(xfun)  
pkg_attach("dplyr", "readr")
```

4.3 Ambiente global

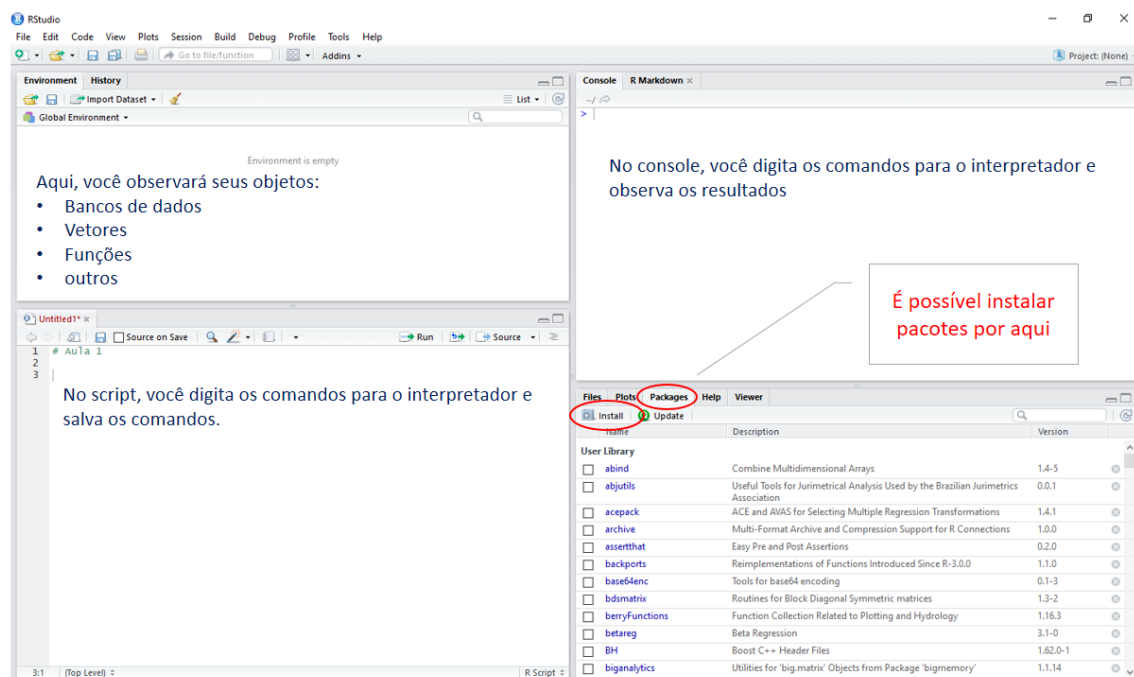


Figura 6: Console do R Studio

5 Obtendo Ajuda

5.1 Ajuda

- Em caso de dúvidas, o **R** possui uma extensiva documentação *offline* para auxiliá-lo a entender os comandos.

Por exemplo, como funciona a função log?

```
help(log)
```

```
?log
```

- Para descobrir quais os argumentos das funções:

```
args(log)
```

- Ainda, é possível ver os exemplos do sistema para uma determinada função:

```
example(log)
```

5.2 Ajuda na internet

- Se a ajuda do programa não esclarecer o problema, é possível encontrar fontes externas de ajuda na internet.
- Você pode procurar o seu problema no google, que provavelmente te retornará uma página:
 - **Stack Overflow**: o site tem uma extensa série de tópicos relacionados a problemas no **R**.
 - Como a comunidade é ativa, você encontrará respostas para a maior parte das suas dúvidas lá.

6 Criando Projetos no R

Uma boa maneira de começar o nosso trabalho no R é criando um projeto. Essa solução auxilia a manejar o trabalho nos diversos contextos - para cada um estabelecemos um diretório, história e conteúdo.

Para manejarmos os scripts e bases de dados desse curso, vamos criar um projeto. Para tanto, clique em **File**, no canto superior esquerdo e, em seguida, **New Project**.

Quando você cria um projeto, basta clicar no símbolo do projeto na pasta de preferência que a sua sessão do R estará salva, bem como todos scripts trabalhados em seus projetos.

7 Estrutura de Dados

7.1 Introdução

As estruturas de dados no **R** podem ser diferenciadas por sua dimensão e pelo seu conteúdo (heterogêneo ou homogêneo).

Dimensões	Homogêneo	Heterogêneo
1	Atomic Vector	List
2	Matrix	Data Frame
Nd	Array	

Figura 7: Estrutura Dados

7.2 Vetores Atômicos

Os vetores atômicos são as estruturas (objetos) mais simples de dados, com uma dimensão e um tipo.

- Há três tipos de armazenamento de objetos (os vetores assumem um desses tipos):
 - Numéricos - double (contínua) ou integer (discreto)
 - Lógicos
 - Texto

7.3 Exemplo 1: criando vetores

- Os vetores são criados por meio dos sinais de atribuição `<-` ou `=`. Sempre assumem um tipo de armazenamento.
- Podemos criar vetores por meio da função **combine**, `c(arg1, arg2, ..., arg_n)`.

```
# Numérico contínuo
num_dbl <- c(1, 2.8, 4.5, 6.3)
# Numérico discreto - coloca um L ao lado
num_int = c(1L, 3L, 5L, 7L)
```



```
# Lógico
log_vt <- c(TRUE, FALSE, TRUE, FALSE)
# Texto
chr_vt <- c("Jõao", "Zezinha", "Itamar", "Cristiane" )
```

7.4 Exemplo 1: criando vetores

- **Atenção:**

- os valores decimais são separados por .
- o uso do **L** ao lado dos números os torna discretos
- o uso das aspas “” marca a criação de textos
- os valores lógicos **TRUE** ou **FALSE** são escritos em maiúsculo. Numericamente, correspondem a 1 e 0.

7.5 Exemplo 1: testando a natureza dos vetores

- Podemos testar se os vetores assumem os tipos esperados. Note que o **R** traz uma grande gama de funções destinadas a verificar a natureza dos vetores e outros objetos.

```
length(num_dbl)    # comprimento do vetor
is.vector(num_dbl) # testa se objeto é vetor
is.numeric(num_dbl) # testa se objeto é numerico
is.integer(num_dbl) # testa se objeto é numerico discreto
is.atomic(num_dbl) # testa se vetor é atomico
is.double(num_dbl) # testa se vetor é numerico
```

- Vamos testar a natureza dos outros vetores criados?

7.6 Exercício 1

Exercício

- Crie seus próprios vetores:
 - numérico, contínuos e discretos
 - lógico
 - character

7.7 factor

- Os fatores são vetores gravados como números inteiros para representar variáveis categóricas em níveis.
 - podem ser variáveis contínuas transformadas em categorias
 - ou variáveis de texto transformadas em categorias.
- Para tanto, utiliza-se o comando `as.factor()`

```
letras = c("a", "b", "a", "b", "c", "c",  
           "d", "e", "e", "e", "e", "d")  
levels(letras) # se não é factor, não tem níveis  
  
## NULL  
  
letras<- as.factor(letras)  
levels(letras) # como factor, cada valor tem um nível  
  
## [1] "a" "b" "c" "d" "e"  
  
table(letras) # veja a quantidade de cada nível  
  
## letras  
## a b c d e  
## 2 2 2 2 4
```

As categorias tem muita utilidade nos modelos estatísticos e na criação de gráficos

7.8 Comandos matemáticos no R

Símbolo/Comando	Operação
+	Soma
-	Subtração
/	Divisão
*	Multiplicação
^	Potência
sqrt	Raiz Quadrada
log	Logarítmo
sum	Soma
cumsum	Soma cumulativa
max	Máximo
min	Mínimo
mean	Média
median	Mediana
sd	Desvio Padrão

7.9 Exemplos de comandos matemáticos

```
# Logaritmo dos números de um vetor
```

```
log(num_dbl)
```

```
## [1] 0.000000 1.029619 1.504077 1.840550
```

```
# Soma cumulativa
```

```
cumsum(num_dbl)
```

```
## [1] 1.0 3.8 8.3 14.6
```

```
# Reverter valores do vetor
```

```
rev(num_dbl)
```

```
## [1] 6.3 4.5 2.8 1.0
```

7.9.1 Exercício 2: operações matemáticas

Exercício:

- Crie um vetor com 10 observações.
- Realize algumas operações matemáticas entre números e vetores.
 - Utilize todas as operações descritas na tabela.
- Crie um vetor numérico e obtenha algumas estatísticas.
- É possível realizar operações matemáticas com vetores de texto? E com vetores lógicos? Teste!

7.10 Outros comandos úteis

Comando (Função)	O que faz
length	Comprimento do vetor
rev	Reverter elementos
sort	Ordenar
order	Retorna a ordem dos elementos
head	Apresentar os primeiros elementos do vetor
tail	Apresenta os últimos elementos do vetor
unique	Apresenta os valores únicos
round	Arredonda as observações
ceiling	Arredonda valores a maior
floor	Arredonda valores a menor

7.10.1 Exemplo de outros comandos úteis

Comumente, precisamos arredondar valores. O **R** oferece uma gama de funções para resolver esse tipo de problema.

```
# Arredondar valores  
round(num_dbl)
```

```
## [1] 1 3 4 6
```

```
# Arredondar a maior  
ceiling(num_dbl)
```

```
## [1] 1 3 5 7
```

```
# Arredondar a menor  
floor(num_dbl)
```

```
## [1] 1 2 4 6
```

Podemos também especificar o número de dígitos do nosso arredondamento.

```
# Arredondar valores
round(1.55222, digits = 2)
```

```
## [1] 1.55
```

```
round(1.55222, 3)
```

```
## [1] 1.552
```

7.11 Aplicando múltiplas funções de uma vez

No **R**, podemos encadear diversas operações, o que facilita muitos dos nossos cálculos. Observe:

```
num_dbl <- c(1, 2.3, 4.8, 5, 6.7, 8, 9, 10)
```

```
# Arredondar a média do vetor num_dbl multiplicado por 3
round(mean(num_dbl * 3))
```

```
## [1] 18
```

```
# Erro padrão do quadrado do vetor num_dbl
sd(num_dbl ^ 2)
```

```
## [1] 35.9484
```

```
# Logaritmo da soma dos quadrados do vetor num_dbl
log(sum(num_dbl^2))
```

```
## [1] 5.841281
```

```
# Variância da amostra
sum((num_dbl - mean(num_dbl))^2) / (length(num_dbl) - 1)
```

```
## [1] 10.06286
```

```
# Variância da amostra (mesmo que o anterior)
var(num_dbl)
```

```
## [1] 10.06286
```

7.12 Operadores Lógicos

- As operações lógicas são cruciais para a manipulação dos dados.
- É possível utilizar o seguinte conjunto de operadores lógicos:

Símbolo	Descrição
<	Menor
<=	Menor ou igual a
>	Maior
>=	Maior ou igual a
==	Exatamente Igual
!=	Não é igual
x y	x ou y
x & y	x e y

7.12.1 Exercício 3: operações lógicas

Exercício:

- Suponha que x representa o mês de nascimento de um indivíduo.

```
x <- c(1, 3, 1, 4, 5, 4, 8, 10, 11, 9, 7, 9)
mes_inicial <- c(1, 1, 1, 2, 6, 6, 6, 8, 8, 8, 8, 8)
mes_final <- c(12, 12, 12, 12, 10, 10, 10, 10, 9, 9, 9, 9)
```

- *Verifique:*
 - quantos casos de x são iguais ao mês inicial e quantos ao final?
 - em quais casos o valor de x é maior que o mês inicial e final?
 - em quais casos não são iguais?

7.13 Operações com Conjuntos

Ainda, é possível realizar operações lógicas com conjuntos. Nesses termos, a união de dois vetores, a interseção, a diferença de conjuntos ou a identificação se um número ou valor pertencem a um conjunto é facilitada.

Símbolo	Descrição
<code>union(x, y)</code>	União
<code>intersect(x, y)</code>	Interseção
<code>setdiff(x, y)</code>	Diferença
<code>setequal(x, y)</code>	Igualdade
<code>is.element(el, set)</code>	É elemento
<code>%in%</code>	Equivalente a é elemento

7.13.1 Exemplo operações conjuntos

```
# Relembre os exemplos da escola
a <- c("João", "Maria", "José", "Josefa")
b <- c("Mariana", "José", "Joana", "João")

# União
union(a, b)

## [1] "João"      "Maria"      "José"      "Josefa"    "Mariana"    "Joana"

# Diferença entre conjuntos
setdiff(a, b)

## [1] "Maria"      "Josefa"

# É elemento? Retorna valor lógico
a %in% b

## [1] TRUE FALSE TRUE FALSE

is.element(a, b)

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

7.13.2 Exercício 4: operações com conjuntos

Dado que:

```
x <- 1:15
y <- seq(5, 55, 2.5)
z <- 12:27
```

Determine:

- quais elementos comuns entre x e y ?
- qual a união entre y e z ?
- qual a interseção entre x e z ?

7.14 Sequências e repetições

- Uma forma importante de criar dados é por meio de sequências e repetições. Além do já conhecido : (**ex.**, **1:10**), há funções que refinam essa lógica:

```
# Criando outras sequências
seq(from = 1, to = 10, by = 2)
# De modo mais breve
seq(0, 100, 0.333)

# Você pode especificar um valor
n <- 1000
q <- 2.5
seq(1, n, q)

# Repetindo argumentos
rep("Olá", 2)

# Você pode criar um vetor com nomes
frutas <- c("laranja", "banana", "manga", "maracujá",
            "mamão", "ameixa")
# Repetir frutas 15 vezes
rep(frutas, 15)
# Repetir cada fruta duas vezes
rep(frutas, each = 2)

# Repetir cada argumento de modo distinto
rep(frutas, c(1, 2, 3, 4, 5, 6))
rep(frutas, 1:6) # mesma coisa
rep(frutas, rep(c(2,3), 3)) # mesma coisa
```

7.14.1 Exercício 5: repetições e sequências

Crie vetores por meio de sequências e repetições. Execute algumas operações lógicas e matemáticas à sua escolha.

8 Matrizes

8.1 Matrizes

As matrizes se diferenciam dos vetores atômicos por ter o atributo dimensão **dim()**.

Tente executar os comandos abaixo. O que você observa?

```
# Matriz
matriz_a <- matrix(1:12, nrow = 4, ncol = 3)

# Vetor transformado em matriz (adicionar dimensão)
vetor_b <- 1:12
dim(vetor_b) <- c(4, 3)
```

8.2 Matrizes

- Como a matriz tem duas dimensões, o comando **length()** se generaliza em:
 - **ncol()** - número de colunas
 - **nrow()** - número de linhas
- Do mesmo modo, o comando **names()** se generaliza em:
 - **rownames()**
 - **colnames()**

8.3 Matrizes (exemplo 1)

```
rownames(matriz_a) <- c("linha_1", "linha_2", "linha_3",  
                        "linha_4")  
colnames(matriz_a) <- c("coluna_A", "coluna_B", "coluna_C")  
  
matriz_a
```

8.4 Matrizes (exemplo 2)

O pode executar todas operações com matrizes. Por exemplo, transpor, inverter, multiplicar, extrair diagonal das matrizes, entre outros.

```
A <- matrix(1:12, nrow = 4, ncol = 3)
B <- matrix(1:6, nrow = 3, ncol = 2)
```

```
# Transpor matriz
t(A)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

```
# Multiplicar matrizes
```

```
A %*% B
```

```
##      [,1] [,2]
## [1,]   38   83
## [2,]   44   98
## [3,]   50  113
## [4,]   56  128
```

```
A * 3
```

```
##      [,1] [,2] [,3]
## [1,]    3   15   27
## [2,]    6   18   30
## [3,]    9   21   33
## [4,]   12   24   36
```

```
B * 2
```

```
##      [,1] [,2]
## [1,]    2    8
## [2,]    4   10
## [3,]    6   12
```

Para saber mais sobre operações com matrizes no **R**, acesse **este site**.

9 Listas

9.1 Listas

- As listas se diferenciam dos vetores atômicos, pois seus elementos podem ser de qualquer tipo, inclusive listas ou matrizes.
- Para criar lista, utiliza-se o comando **list()**

```
lista<- list(x = 1:5,  
            "Curso de R",  
            c(TRUE, FALSE),  
            c(1.5, 1.6, 1.7, 1.8),  
            matriz = matrix(1:12, 3, 4),  
            lista2 =list(letters, LETTERS))  
str(lista)
```

Embora as listas sejam muito importantes, sobretudo na construção de loops *for*, podem passar despercebidas por muitos usuários do **R**

10 Data Frame

10.1 Data Frame

- Essa é a forma mais comum de encontrar dados no **R**.
- É um tipo de lista com vetores de comprimento iguais e duas dimensões, do mesmo modo que as matrizes.

_ Em regra todos os quadros de dados que analisamos estão nesse formato. Por isso, o foco do curso a partir deste momento migrará para as aplicações da análise dos dados no formato `data.frame`.

10.2 Data Frame

Abaixo, criamos um objeto da classe `data.frame` ao qual chamamos sucintamente *df* . Você notará que compilamos vetores de diversas classes e mesmo comprimento em um `data.frame` com a função **`data.frame`**.

```
# Criando um data.frame na mão  
df <- data.frame(nome =c("Josias", "Joaldo", "Josefa",  
                        "Josie", "Josimar"),  
                 idade = c(42L, 28L, 34L, 27L, 55L),
```

```
tem_filhos= c(TRUE, FALSE, FALSE,
               TRUE, TRUE),
renda = c(1457.2, 954.7, 1600.8, 900.5, 600.4))
print(df)
```

```
##      nome idade tem_filhos  renda
## 1 Josias   42      TRUE 1457.2
## 2 Joaldo   28     FALSE  954.7
## 3 Josefa   34     FALSE 1600.8
## 4 Josie    27      TRUE  900.5
## 5 Josimar  55      TRUE  600.4
```

10.3 Data Frame

Apresentamos algumas funções muito importantes e bastante utilizadas na análise dos data.frames. Com elas, você terá o primeiro olhar sobre os seus dados.

```
# Estrutura do data.frame
str(df)
```

```
## 'data.frame':  5 obs. of  4 variables:
## $ nome      : Factor w/ 5 levels "Joaldo","Josefa",...: 3 1 2 4 5
## $ idade     : int  42 28 34 27 55
## $ tem_filhos: logi  TRUE FALSE FALSE TRUE TRUE
## $ renda     : num  1457 955 1601 900 600
```

```
# Nomes das colunas
names(df); colnames(df)
```

```
## [1] "nome"      "idade"     "tem_filhos" "renda"
```

```
## [1] "nome"      "idade"     "tem_filhos" "renda"
```

```
# Qual o número de linhas?
nrow(df)
```

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

```
# Número de colunas
ncol(df)
```

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

```
# Sumário dos dados  
summary(df)
```

```
##      nome      idade  tem_filhos      renda  
## Joaldo :1  Min.    :27.0  Mode :logical  Min.    : 600.4  
## Josefa :1  1st Qu.:28.0  FALSE:2    1st Qu.: 900.5  
## Josias :1  Median :34.0  TRUE :3    Median : 954.7  
## Josie  :1  Mean    :37.2          Mean    :1102.7  
## Josimar:1  3rd Qu.:42.0          3rd Qu.:1457.2  
##              Max.    :55.0          Max.    :1600.8
```

10.4 Data Frame: acessando elementos com o \$

- É possível acessar os elementos do data.frame por meio do \$. Essa é a forma de extrair informações de objetos com mais de uma dimensão.

```
# Observar a coluna idade  
df$idade
```

```
## [1] 42 28 34 27 55
```

- Com isso, é fácil extrair a média da coluna idade

```
# Qual a média da idade  
mean(df$idade)
```

```
## [1] 37.2
```

10.5 Data Frame: acessando seus elementos com os []

- Outra forma de acessar os elementos dos objetos (listas, vetores, data.frames ou matrizes), é por meio dos [].
 - Para o uso [,], o lado direito da vírgula é linha e o esquerdo coluna.

```
df[ , ] # todo data.frame
```

```
##      nome idade tem_filhos  renda
## 1  Josias   42      TRUE 1457.2
## 2  Joaldo   28     FALSE  954.7
## 3  Josefa   34     FALSE 1600.8
## 4   Josie   27      TRUE  900.5
## 5 Josimar   55      TRUE  600.4
```

```
df[1, ] # linha 1
```

```
##      nome idade tem_filhos  renda
## 1 Josias   42      TRUE 1457.2
```

```
df[, 2] # coluna 2
```

```
## [1] 42 28 34 27 55
```

```
df[ 1, 2]
```

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

10.6 Data Frame: adicionando elementos

- Do mesmo modo que é possível extrair informações de um data.frame, é possível adicionar.
- É possível, então, criar novas colunas:

```
df$qt_d_filhos <- c(3, 4, 1, 3, 6)
df[, "seis"] <- 6
df[, "cidade"] <- "Brasília"
```

- O uso do `$` extrai ou inclui exatamente o nome da variável.
- Já os `[]` extraem ou incluem a coluna a que se referem.

11 Do Data.Frame para Tibble

11.1 Tibble: um data.frame melhorado

- O tibble apresenta um formato moderno para o data.frame.
- É parte do *Tidyverse*.
- Traz as seguintes vantagens:

- Facilita a visualização dos dados
- Facilita a criação dos dados
- Facilita a reciclagem de vetores (!?)

11.2 Tibble: visualizando os dados

- É possível compelir um data.frame comum à classe tibble.

```
# Qual a classe atual do "df"
class(df)
# Transformar "df" em tibble
library(tidyverse)
# Compelir o data.frame para tibble
df <- as_tibble(df)
df
```

11.3 Tibble: criando dados reciclando vetores

- É possível criar dados com o tibble de forma mais fácil:

```
# Tente criar esse data frame
df <- data.frame(nomes = c("João", "Pedro", "Maria", "Érica"),
                 idade = c(18, 25, 49, 38),
                 idade_q = idade^2,
                 idade_media = mean(idade))
```

11.4 Tibble: criando dados reciclando vetores

- É possível criar dados com o tibble de forma mais fácil:

```
# Tente criar esse data frame
df <- data.frame(nomes = c("João", "Pedro", "Maria", "Érica"),
                 idade = c(18, 25, 49, 38),
                 idade_q = idade^2,
                 idade_media = mean(idade))

# Com o tibble - compare
dfti <- tibble(nomes = c("João", "Pedro", "Maria", "Érica"),
               idade = c(18, 25, 49, 38),
               idade_q = idade^2,
               idade_media = mean(idade) )
```

11.5 Exercício 5

Exercício

- Crie um `data.frame` com 15 linhas e 5 colunas em que:
 - a primeira coluna seja uma sequência de 1 a 15;
 - a segunda seja uma coluna de letras maiúsculas;
 - a terceira compreenda os números 1, 2 e 3;
 - a quarta seja a multiplicação da soma da coluna 1 e 3 por 5;
 - a quinta seja um vetor lógico que indique se os valores da quarta coluna sejam maiores do que sua média.