



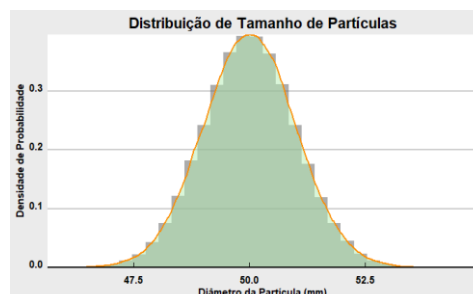
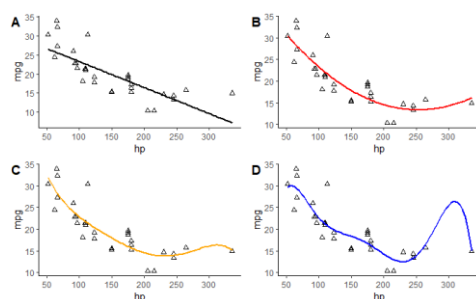
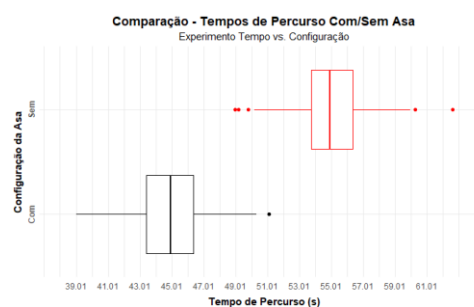
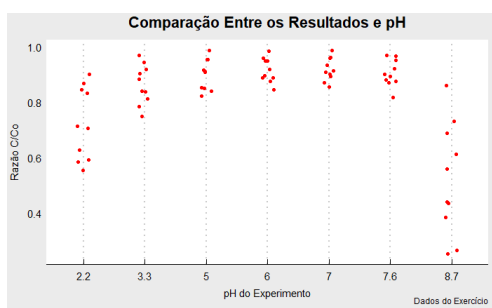
Escola de Engenharia de Lorena (EEL) – Dezembro de 2020

Treinamento – Curso de R (1ª Edição)

Henrique Azank dos Santos (PowerTrain 2020)

Grupo EEL Racing

Minicurso em Programação e Análise de Dados usando a Linguagem R e o Ambiente RStudio com Ênfase em Estatística



Lorena (SP), 06 de dezembro de 2020

I. Prelúdio:

Olá caro leitor(a). Se você está lendo isso é porque fez parte da 1ª Edição do curso de programação R da EEL Racing ou está interessado em aprender um pouco sobre programação em R.

Inicialmente, é importante destacar o porque a programação em R é uma ferramenta útil nos dias de hoje. Esta linguagem de programação é um ambiente especialmente projetado para realizar computações estatísticas, manipular dados, realizar computações e expressar resultados em formas gráficas. Em seu estado padrão, muitas técnicas estatísticas e métodos de cálculo numérico podem ser realizadas como: Modelagem linear e não linear, testes de hipóteses estatísticas, classificação, agrupamento/*clustering*, análise de séries temporais e muitas outras. Entretanto, as características de maior destaque desta linguagem são sua capacidade de estender sua funcionalidade com inúmeros pacotes, possuir uma sintaxe acessível à novos usuários, gerar gráficos de alta qualidade altamente customizáveis e sua acessibilidade, afinal, se trata de um software livre e universalmente disponível.

A linguagem de programação conta com uma crescente comunidade de desenvolvedores e usuários, que projetam e compartilham milhares de funções todos os dias em inúmeros sites voltados à programação (como o *Stack Overflow* - <https://stackoverflow.com/>) ou à ciência de dados (como o *STHDA* - <http://www.sthda.com/english/>). Por se tratar de uma linguagem extremamente popular e versátil, muitos dos problemas que novos usuários encontram são facilmente encontrados na internet, sendo que o número de materiais *online* e cursos voltados à programação crescem todos os dias.

Pessoalmente, só entrei em contato com o R na metade de 2019 quando li um artigo muito interessante de engenharia bioquímica que tinha sido inteiramente feito utilizando R. Desde aquele dia, essa linguagem se tornou meu principal ambiente de cálculos e projetos, abrindo meus horizontes sobre o que é possível se fazer com apenas algumas linhas de código e força de vontade.

Com o intuito de apresenta-los ao mundo de programação em R voltada à engenharia, desenvolvi este pequeno curso (ainda em sua versão *Beta*) para familiarizá-los com o ambiente do RStudio (ambiente para trabalho com R), apresentar as sintaxes das principais funções para a análise de dados e ilustrar a funcionalidade dos principais pacotes anexos ao R, principalmente o conjunto denominado de *Tidyverse*.

Espero que aproveitem o curso e este material, grato desde já pela oportunidade,

Henrique Azank dos Santos

Índice:

I. Prelúdio 2

01. Instalando R e RStudio 3

02. Introdução à Análise de Dados 6

03. Estruturas de Controle em R – Operando Loops 13

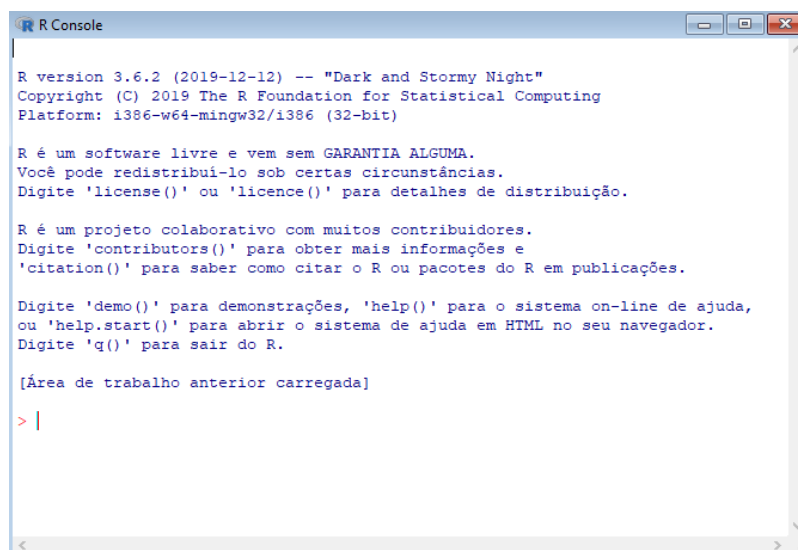
04. Introdução à Data Frames e o Pacote DPLYR 15

1. Instalando R e RStudio e seus primeiros passos:

Antes de apresentarmos quaisquer sintaxes de funções e metodologias estatísticas, é importante saber como fazer o download e instalar os componentes necessários do R e do RStudio. Inicialmente, o console do R, que consiste essencialmente no *software* em si que realizará todas as operações de cálculo, chamada de funções, leitura de dados e etc. é livremente disponível no endereço <https://www.r-project.org/>, site oficial do projeto R e do projeto CRAN. De uma forma simples, os seguintes passos devem ser realizados:

01. Abra o endereço <https://www.r-project.org/> em seu navegador padrão;
02. Clique na aba “CRAN”, localizada à esquerda. Nessa aba, você encontrará uma lista de links subdivididos em países. Estes links representam os servidores que patrocinam o programa CRAN e disponibilizam seus servidores para este propósito. Em nosso caso, clicar em quaisquer uns dos links do “Brazil” irá leva-lo à página de download;
03. Uma vez na página, pode-se realizar o download do R (o arquivo executável *.exe*) para sua respectiva plataforma (Linux, Mac ou Windows);
04. Quando o Download é finalizado, execute o arquivo e siga os procedimentos de instalação. Quando acabado, o console do R pode ser aberto em seu computador.

Figura 1: Ilustração do console R – disponível para download no site do projeto CRAN.



```
R Console

R version 3.6.2 (2019-12-12) -- "Dark and Stormy Night"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R é um software livre e vem sem GARANTIA ALGUMA.
Você pode redistribuí-lo sob certas circunstâncias.
Digite 'license()' ou 'licence()' para detalhes de distribuição.

R é um projeto colaborativo com muitos contribuidores.
Digite 'contributors()' para obter mais informações e
'citation()' para saber como citar o R ou pacotes do R em publicações.

Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.

[Área de trabalho anterior carregada]

> |
```

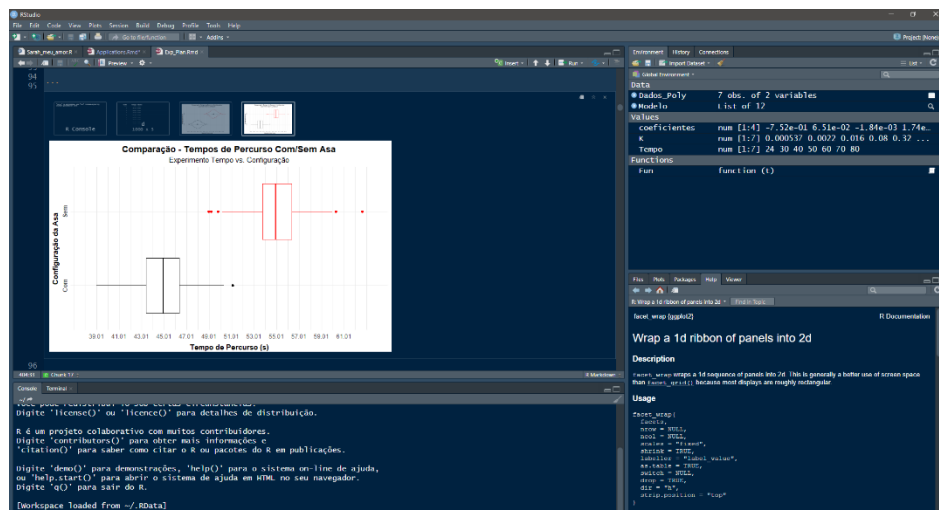
Apesar do console poder ser utilizado para quaisquer práticas de programação, análise de dados e quaisquer outras aplicações de R, encontram-se alguns problemas em sua utilização no dia a dia.

Primeiramente, quando definimos muitas variáveis e trabalhamos com algoritmos longos ou scripts longos, o acompanhamento, documentação e utilização se tornam mais difíceis.

Felizmente, existem maneiras alternativas de se utilizar o R. Assim como a maioria das linguagens de programação modernas, o R conta com um ambiente especializado para programação e confecção de projetos. Este ambiente é o denominado *RStudio*. Formalmente, o RStudio é um ambiente integrado de desenvolvimento para R, que conta com as mesmas ferramentas do console (que também está presente no RStudio), um editor de código que facilita muito a sintaxe de programação, ferramentas de execução de código (*Scripts*, *Markdowns*, *Notebooks* e etc.), ferramentas de plotagem, histórico e *debugging* além de providenciar uma forma muito mais organizada para realizar nossas análises.

Com o RStudio, o usuário consegue, utilizando a linguagem R, construir e executar scripts para a análise de dados, criar Web aplicativos interativos, confeccionar documentos, relatórios, gráficos e muito mais. Trata-se basicamente de um grande “caderno” para nós, onde podemos importar dados, exportar dados, realizar nossas análises, documentar nossos achados, criar visualizações e finalmente compartilhar o que achamos.

Figura 2: Ilustração do RStudio – disponível para download no site do RStudio. Nessa figura podemos ver o navegador de variáveis, o Notebook de projetos, o console e a aba de ajuda.



Assim como todo bom caderno, o RStudio é altamente customizável. Em sua versão padrão, o RStudio conta com um fundo branco por exemplo, e as janelas do navegador possuem tamanhos diferentes. Essas definições são facilmente editáveis e adaptáveis para as necessidades do usuário.

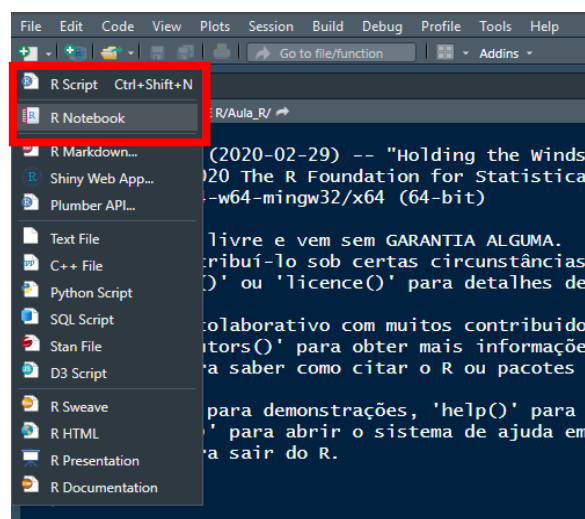
O RStudio possui muitos outros serviços (como servidores para seus aplicativos e dashboards web, compartilhamento de projetos simultâneos e etc.) mas a versão *desktop* é grátis e segue o espírito do open-source software. Para realizar o download do RStudio, basta seguir para o link <https://rstudio.com/products/rstudio/>.

2. Introdução à Sintaxe de R e a Análise de Dados:

Vamos agora começar a entender os fundamentos básicos para se trabalhar com dados em R. Para trabalharmos com dados primeiramente precisamos aprender como introduzir dados, armazená-los e manipulá-los no ambiente do RStudio. A linguagem pode trabalhar com números, matrizes, textos, *data sets*, gráficos, funções e muitos outros tipos de objetos.

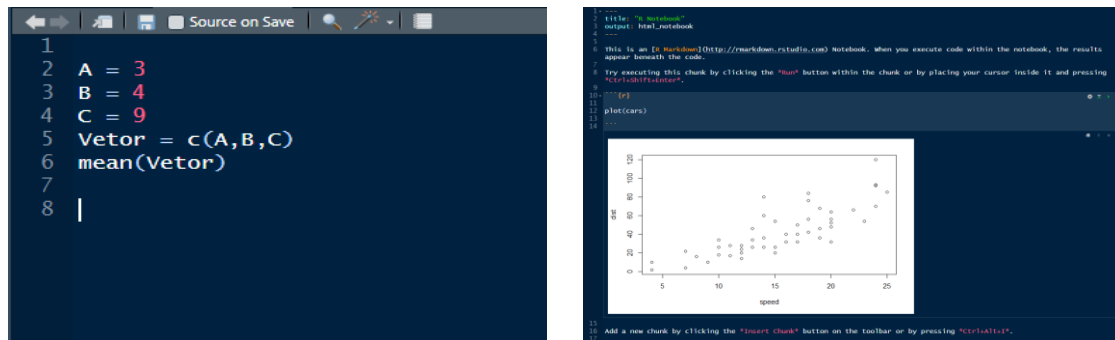
Para começarmos nosso trabalho, devemos ter em mente que podemos construir nossos códigos em basicamente 2 tipos de documentos: Os *Scripts* e os *Notebooks*. Os *Scripts* são documentos onde cada uma das linhas aloja códigos e funções, sendo que quando “rodamos” o *script* (ou seja, executamos o arquivo) o console do R realiza as instruções de acordo com a ordem que foram programadas. Enquanto isso, os *Notebooks* atuam como verdadeiros cadernos para nossos projetos, sendo um híbrido entre um documento de texto, visualização gráfica e console. Nos *Notebooks*, a informação digitada é considerada como um texto, ou seja, não como código. Se desejamos inserir código, incluímos os chamados “*Chunks*”, que essencialmente são “mini” *Scripts*, que se portam como tais e podem ser executados separadamente. Para criarmos um novo documento, clicamos no ícone “novo” localizado no canto superior esquerdo:

Figura 3: Localização das opções para criar novos arquivos no RStudio. Destaca-se as opções *Notebook* e *Script*.



Nota-se também que o RStudio também pode criar arquivos que utilizem outras linguagens de programação, como *Python*, *C++* e *SQL*. Nos *Notebooks*, podemos incluir inúmeros *Chunks*, incluindo *Chunks* de linguagens diferentes, o que se torna muito útil quando realizando projetos integrados com outras linguagens.

Figura 4 e 5: Comparação entre um *Script* de R (Esquerda) e um *Notebook* (Direita).



Vamos agora discutir sobre os principais tipos de objetos que podemos introduzir no R. Para designarmos dados e armazená-los em uma variável em R podemos utilizar dois operadores: o operador de igual (=) ou o operador de “seta para a esquerda” (<-). Os dois operadores atingem o mesmo objetivo, que é designar à variável a esquerda os valores (dados, ou resultados) apresentados à direita, ou seja, você pode escolher qualquer um dos dois e depende inteiramente de sua preferência.

Assim como na imensa maioria das linguagens de programação, todos os objetos que trabalhamos em R possuem uma *classe*. A classe do objeto / variável define o tipo de dado que se trata o mesmo. Vale a pena destacar que em R, podemos nomear variáveis de infinitas maneiras, desde que se obedecem a algumas pequenas condições (como por exemplo, não possuir “espaços” em seu nome). Geralmente, nomeiam-se variáveis de forma que facilmente identifiquemos o que elas representam. Algumas sugestões e exemplos são dados a seguir:

- Se desejamos armazenar as notas de uma turma, podemos definir o vetor que armazena essas informações como “Notas” ou ainda “Notas_Turma_1”. A variável não pode ser definida como “Notas Turma 1”, já que o R não reconhece os espaços entre os caracteres;
- O R reconhece as diferenças de caracteres maiúsculos e minúsculos, ou seja, a variável “A” e a variável “a” são diferentes;
- Não se pode atribuir duas variáveis com o mesmo nome;

Assim como exposto anteriormente, as variáveis que definimos podem ser de múltiplas classes¹, dependendo do tipo de dados ou informações que atribuímos a elas. As principais classes de objetos que podemos utilizar são:

- ***Numeric*** (Numérico): Tratam-se de números inteiros e não inteiros. Existe também a classe ***Integer***, que se limita para números inteiros. Trata-se de uma das classes mais utilizadas no cotidiano, já que a grande maioria das aplicações de estatística, cálculo e análise de dados obrigatoriamente necessita de análises numéricas;

Um exemplo de objeto de classe numérica é qualquer variável que definimos como números (como $a = 3$ ou $b = 987$). Quando atribuímos o resultado de funções que tem retorno numérico (por exemplo média, mediana, variância e etc.) a variável também possuirá classe numérica, como por exemplo $Media = mean(x)$, onde x é um vetor de números do qual desejamos conhecer a média.

NOTA: Iremos mencionar vetores e matrizes em instantes, mas de forma a prosseguirmos com nossos exemplos a seguir, devemos entender como podemos concatenar diferentes variáveis, elementos ou dados utilizando o operador $c(..., ...)$. Este operador permite criar vetores de um mesmo tipo de dado, ou seja, consegue criar vetores de números, palavras (caracteres), fatores e etc.

$A = c(2,4,5) \rightarrow$ Vetor com os elementos 2,4 e 5

NOTA IMPORTANTE: Em R, números decimais seguem o padrão americano (ponto e não vírgula)

- ***Character*** (Caracteres): Em muitas linguagens de programação, este tipo de objeto é denominado de classe “String”. Basicamente, consistem de informações na forma de frases, palavras ou letras;

Para construirmos variáveis desta classe, atribuímos o que desejamos digitar entre aspas. Um bom exemplo seria especificar qual seu nome em uma variável ($Nome = "Henrique Santos"$). Essas variáveis podem ser meras palavras, frases, parágrafos ou até textos inteiros. Um exemplo deste tipo de variáveis são os nomes de alunos em uma escola, que podem ser dados por:

$Nomes_Alunos = c("Carol","Guilherme","Henrique", ...)$

- ***Factor*** (Fatores): Trata-se de uma das classes mais importantes quando analisamos dados. Apesar de se parecerem com os caracteres, os fatores atuam de forma diferente, apesar de serem

¹ Mais detalhes disponíveis em https://www.brodrigues.co/blog/2018-12-24-modern_objects/

representados por palavras e símbolos, atuam como classificadores. Oficialmente, fatores são variáveis que são limitadas a um conjunto de valores (funcionando assim como variáveis categóricas)²;

Os fatores são estruturados da seguinte forma: Digamos que temos um grupo de 10 alunos da EEL. Esses 10 alunos podem pertencer a um dos 6 cursos disponíveis na unidade. Digamos que designamos os cursos de 1 a 6. Podemos criar uma variável categórica “Cursos_Alunos” a partir de:

```
Cursos_Alunos = c(1,5,6,3,2,4,5,4,1,5)
```

```
Cursos_Alunos = factor(Cursos_Alunos, labels = c("EB", "EQ", "EM", "EF", "EA", "EP"))
```

A partir desta sintaxe, o vetor contendo os códigos dos cursos é transformado em um conjunto de fatores, cujos níveis agora são as iniciais das disciplinas. Os fatores são de suma importância, principalmente quando analisando variáveis categóricas. Muitos algoritmos de análise de dados são construídos assumindo-se que variáveis categóricas (como sexo/gênero, tratamento, dia/noite, tipo de produto, e etc.) sejam codificadas na forma de fatores. Para analisarmos rapidamente a quantidade de membros de cada nível do fator (em nosso exemplo o curso) podemos utilizar a função `table()`: `table(Cursos_Alunos)`.

- **Logical** (Classe dos operadores lógicos): Operadores lógicos, também chamadas de variáveis Booleanas, são variáveis que adotam apenas dois valores possíveis: *Verdadeiro* ou *Falso*. Essas variáveis se originam quando se realizam comparações entre variáveis ou números. As variáveis lógicas em R podem ser especificadas manualmente por TRUE ou FALSE (ou ainda por T ou F).

As variáveis Booleanas possuem especial aplicação em análise de dados. Quando desejamos avaliar uma determinada condição em um vetor, matriz ou *data set*, os operadores Booleanos permitem identificar *onde* essas condições são atendidas ou desrespeitadas. Suponhamos que possuímos um vetor que contenha as notas de 10 alunos em uma determinada disciplina:

```
Notas_Alunos = c(5,7,8,6,9,10,7,9,8,10)
```

Suponhamos agora que desejamos verificar quais dessas notas são maiores, por exemplo, que 7 podemos utilizar operadores de comparação: (`>`, `<`, `<=`, `>=`). Uma vez definida a variável

² Mais informações acerca deste tópico em <https://www.stat.berkeley.edu/~s133/factors.html>

“Notas_Alunos”, podemos utilizar estes comparadores para testar uma condição. Para averiguar quais notas são maiores (e não iguais a 7) escrevemos:

Notas_Alunos > 7

[1] FALSE FALSE TRUE FALSE TRUE TRUE FALSE TRUE TRUE TRUE

Cada uma das entradas desse resultado representa se o respectivo número do vetor atende à condição especificada, ou seja, se a nota é maior do que 7. Mas qual a utilidade destas variáveis lógicas? Podemos utilizar esses operadores para filtrar dados e obter algumas informações acerca deles. Um exemplo de utilização é determinar *quais* dados obedecem a uma certa condição.

NOTA: Várias estruturas de dados são possíveis de serem construídas em R. Podemos construir matrizes, vetores, *data frames*, listas e muitas outras estruturas. Naturalmente, muitos dados são armazenados nessas estruturas, e devemos estar militarizados em como extrair subsecções destas estruturas. No caso anterior, estamos tratando de um vetor linha. Para extrairmos uma entrada desse vetor, utilizamos a seguinte sintaxe:

Notas_Alunos[x]

onde *x* indica os índices das entradas que desejamos extrair. Dessa maneira, por exemplo, se especificarmos Notas_Aluno[3] o resultado que obteremos no console é 8. Essa forma de obter subsecções é bem similar à estrutura de matrizes, ou seja, se “Notas_Alunos” fosse uma matriz, a forma de obtermos um elemento seria:

Notas_Alunos[linha, coluna]

Podemos utilizar essa forma de secção aliado às variáveis lógicas para obtermos os termos de um vetor que satisfaçam uma determinada condição. No caso das notas dos alunos acima de 7, podemos obter as notas passando o vetor lógico obtido anteriormente como argumento da subsecção:

Index = Notas_Aluno > 7

Notas_Alunos[Index]

8 9 10 9 8 10

Podemos também ver qual a proporção de notas foram acima de 7 através da função mean(x). Normalmente, utilizamos a função mean(x) para determinar a média de um vetor numérico (poderíamos usar **mean(Notas_Alunos)** para encontrar a média das notas, nesse caso 7.9), mas quando passamos um vetor logico (como por exemplo Notas_Alunos > 7) a função resulta na

proporção de resultados VERDADEIROS, ou seja, qual a porcentagem de entradas verdadeiras. No caso da nota dos alunos, temos 6 notas acima de 7, logo:

`mean(Notas_Alunos > 7)` resulta em 0.6

Outros operadores também são úteis. Quando designamos uma variável, podemos utilizar o sinal de igual (=). Este sinal de igual, entretanto não é utilizado para comparações, ou seja, quando desejamos averiguar se uma variável é igual a um determinado valor ou se duas variáveis são iguais, utilizamos dois sinais de igual seguidos (==). O sinal de desigualdade é dado por (!=) e também é utilizado para realizar comparações. Outros operadores lógicos tradicionais também estão presentes, como o operador **E** (&) e o operador **OU** (|). O resultado dessas comparações também resulta em variáveis lógicas.

Um exemplo simples desses operadores é verificar se um número é igual a ele mesmo. Por exemplo, 3 é obviamente igual a 3. Dessa forma o resultado de `(3 == 3)` é TRUE. Da mesma maneira, se digitamos `(3 != 3)` o resultado é FALSE, uma vez que 3 é igual a 3.

Vetores, Matrizes e Listas: Podemos criar vetores de diferentes formas. Em linguagens de programação, “vetores” correspondem a nada mais do que uma “lista” de coisas. Essas coisas podem ser números, palavras, diferentes tipos de dados e até outros vetores. Vetores em R podem armazenar somente um tipo de elementos, ou seja, não se pode misturar diferentes tipos de dados dentro de um vetor. As listas, *data frames* e *tibbles*, entretanto, possuem maior flexibilidade, já que admitem diferentes tipos de dados.

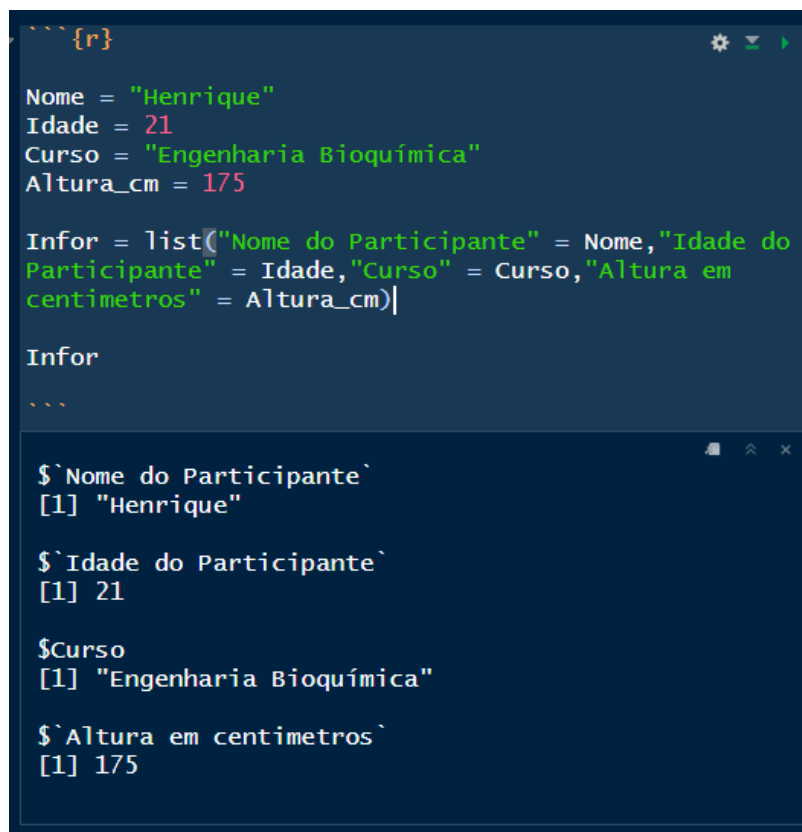
01. Vetores: Assim como exposto anteriormente, podemos facilmente construir vetores através da função `c(...)`. Essa função cria vetores, mas não no sentido matemático. Por exemplo, quando criamos um vetor do tipo:

`Vetor = c(x,y,z)`

Espera-se que esse vetor possua 3 dimensões correto? A função `dim(Objeto)` retorna a dimensão do objeto. O resultado de `dim(Vetor)` é na verdade `## NULL`. Isso significa que o “Vetor” não possui dimensões, e se trata de um **vetor atômico**. Para criar vetores matemáticos, devem-se utilizar as funções `cbind(...)` e `rbind(...)`. Por definição, vetores são considerados vetores linhas ou colunas. `cbind(...)` cria vetores linhas e `rbind(...)` cria matrizes “colando” matrizes linha.

Para criarmos as chamadas listas, utilizamos a função `list(...)`. As listas são muito menos discriminatórias em questão dos tipos de dados que elas aceitam. Podemos agrupar múltiplos tipos de informações dentro de uma lista. Dessa forma, listas são particularmente úteis para agrupar dados diferentes referentes à um mesmo tópico. Podemos por exemplo criar uma lista contendo as informações para cada participante do curso:

Figura 6: Criando nossa primeira lista – Informações individuais de cada participante.



```
## {r}

Nome = "Henrique"
Idade = 21
Curso = "Engenharia Bioquímica"
Altura_cm = 175

Infor = list("Nome do Participante" = Nome, "Idade do Participante" = Idade, "Curso" = Curso, "Altura em centímetros" = Altura_cm)

Infor

##

$`Nome do Participante`
[1] "Henrique"

$`Idade do Participante`
[1] 21

$Curso
[1] "Engenharia Bioquímica"

$`Altura em centímetros`
[1] 175
```

Quando criamos uma estrutura de dados que pode acomodar diversas estruturas e resultados, é de interesse saber como selecionamos um dado em específico. No caso de listas, podemos selecionar uma informação pela seguinte sintaxe:

Nome_da_Lista\$Nome_da_Variável

Quando colocamos (\$) ao lado do nome da lista, um pequeno menu de opções se abre e podemos selecionar qual informação estamos desejando. No caso da lista anterior, se especificarmos `Infor$Curso` o resultado do console será “Engenharia Bioquímica”.

O exemplo de utilização mais relevante que podemos pensar é como o R divulga os resultados de uma regressão linear (tópico abordado mais a frente). Quando realizamos uma regressão linear, os resultados são armazenados na forma de uma lista. A seguir vemos um exemplo utilizando um *data set* famoso denominado de *mtcars*:

Figura 7: Exemplo de aplicação de listas: Comunicar resultados de uma regressão linear. Neste caso, trata-se de uma regressão correlacionando o consumo de combustível de um carro (*Miles per Galon*) com seu peso (*Weight*).

```
Call:
lm(formula = mpg ~ wt, data = mtcars)

Residuals:
    Min       1Q   Median       3Q      Max
-4.5432 -2.3647 -0.1252  1.4096  6.8727

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851     1.8776   19.858  < 2e-16
wt          -5.3445     0.5591   -9.559 1.29e-10

(Intercept) ***
wt          ***
---
Signif. codes:
0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.046 on 30 degrees of
freedom
Multiple R-squared:  0.7528,    Adjusted R-squared:
0.7446
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-
10
```

03. Estruturas de Controle em R – Operando Loops:

Antes de entrarmos em uma maior discussão sobre estruturas de dados e mais mecanismos acerca de modelos estatísticos, devemos primeiro nos familiarizar com os conceitos de estruturas de controle em R. Estas estruturas, também chamadas de “estruturas de fluxo”, são basicamente obrigatórias em linguagens de programação, sendo que o R não é exceção. Elas são as peças chaves para criarmos programas e/ou automatizar nossas análises.

Existem 8 tipos de estruturas de controle em R, sendo algumas delas:

- IF
- IF-ELSE
- FOR
- WHILE

- REPEAT AND BREAK
- NEXT
- RETURN

Essas estruturas de controle geralmente requerem operadores lógicos, ou seja, quando uma determinada condição é respeitada (ou desrespeitada), os comandos dentro de sua estrutura são executados ou não. Elas são utilizadas para formar os famosos *Loops* de programação, onde um determinado conjunto de comandos é realizado até que a condição fornecida seja desrespeitada ou atingida. Dessa forma, elas não só são fundamentais para programar funções, mas também para desenvolvermos nossos próprios algoritmos.

A. Estrutura IF (SE):

Essa estrutura realiza as instruções especificadas se a condição que colocarmos no interior dos parênteses ser respeitada. Ou Seja:

```
if (Expressão Teste){Instruções}
```

Assim como já observamos nos operadores lógicos, a “Expressão Teste” deve ser da classe lógica e ter valor TRUE (Verdadeiro). Quando isso se concretiza, as operações dentro de {Instruções} são conduzidas.

B. Estruturas IF-ELSE):

Essa estrutura é mais utilizada quando desejamos estabelecer dois conjuntos de instruções/códigos, a primeira sendo (Instruções 01) executada quando a condição é verdadeira e a segunda (Instruções 02) sendo executada quando a condição não é atendida:

```
if (Expressão Teste){
    Instruções 01} else{
    Instruções 02}
```

C. Estruturas de LOOP FOR:

Um LOOP é definido como um conjunto de instruções que é repetida até que certa condição seja atingida. As principais formas de LOOPS em R (e em linguagens de programação no geral) são o FOR, WHILE e REPEAT, que podem ser complementadas por BREAK ou NEXT.

A estrutura FOR é utilizada para casos em que desejamos especificar um valor para uma determinada variável, realizar um cálculo ou um conjunto de instruções com a variável naquele valor, registrar os resultados e então repetir esse processo com um outro valor para a variável:

for(Variável **in** Vetor){Instruções}

D. Estruturas WHILE:

As estruturas WHILE funcionam de forma a repetir o loop sempre que a expressão especificada seja verdadeira. Dessa forma, se a expressão sempre for verdadeira, o loop será repetido *infinitamente*. Assim, esse tipo de estrutura é muito útil quando desejamos repetir um determinado conjunto de instruções (ou cálculos) até que uma condição (ou variável) atinja um determinado valor:

while(Condição){Instruções 01}

E. Estruturas REPEAT, BREAK e NEXT:

As estruturas REPEAT constroem loops infinitos, ou seja, todas as instruções dentro do REPEAT são repetidas infinitamente. Mas qual a utilidade de loops infinitos? Podemos utilizar essas estruturas em cálculo numérico para realizar as mesmas instruções indeterminadamente. Mas como saímos desse LOOP infinito? Para sairmos deste loop infinito (ou qualquer loop) podemos utilizar a estrutura BREAK. Quando BREAK é detectado em uma iteração, o loop é automaticamente interrompido e o computador não realiza mais nenhuma iteração. Quando utilizamos REPEAT, associamos BREAK com uma estrutura IF:

repeat{instruções 01

if(condição){BREAK} }

