Tipos básicos de objeto no R: vetores e listas

Ronaldo Baltar[1] and Claudia Siqueira Baltar[1]

[1]Affiliation not available

April 12, 2024

Resumo

Esse conteúdo digital foi criado como parte da atividade de capacitação "Introdução ao R e RStudio para ciência social computacional". É uma iniciativa de formação e difusão científica do Observatório de Populações e Políticas Púbicas (ObPPP) e do Programa de Formação Complementar de Graduação em Pesquisa Social Computacional (InfoSoc), ambos vinculados ao Dept.º C. Soc. / CLCH, Universidade Estadual de Londrina (UEL), Paraná - Brasil

1 As características do R

Para usar o **R** como linguagem computacional para pesquisa social, é importante conhecer a estrutura do programa. Não é suficiente aprender alguns comandos e reproduzir os *scripts* como receitas prontas. A grande vantagem da informática está na capacidade quase que ilimitada de tornar pensamentos e ideias em resultados replicáveis. E para que a imaginação não se limite aos exemplos de códigos inicialmente disponíveis, mas leve os recursos computacionais ao máximo do potencial criativo de quem está fazendo a pesquisa, se faz necessário conhecer mais a fundo a lógica de programação com o **R** .

As linguagens de programação, desde que começaram a ser popularizadas na década de 50 do século XX, se desenvolveram a partir de diferentes paradigmas. Primeiro foram as linguagens de máquina, depois os pseudocódigos em Assembler e então as linguagens de maior abstração (mais próximas da linguagem humana, principalmente o inglês, do que da linguagem de máquina), com os paradigmas procedural, funcional, orientado à objetos e paradigma lógico. Nessas linguagens não há o termo "objeto," apenas variáveis, procedimento e funções (que são procedimentos que retornam algum valor).

O **R** , criado na década de 90, tem por princípio os paradigmas funcional e orientado a objeto. Aqui abre-se uma grande discussão na área de computação sobre esses termos, variáveis, funções, objetos, métodos, classes. O importante a se considerar, nesse caso, não são as definições conceituais em termos puros, mas a forma como se pode utilizar o **R** para otimizar os *scripts* que serão escritos para analisar dados.

O **R** está construído em cima das linguagens C++ e Fortran. Logo, do ponto de vista da execução do programa, tudo é executado como variáveis e funções. Contudo, a estrutura lógica do **R** foi erguida a partir da concepção de uma linguagem funcional e orientada à objeto. Uma linguagem funcional, grosso modo, reduz a necessidade de construção de passos procedimentais.

De modo geral, a linguagem orientada a objetos, reduz a necessidade de funções, uma vez que cada objeto teria as definições de classes (tipos de dados e outros atributos) e métodos (funções) correspondentes. Assim, a intenção era que, com poucas linhas de código e poucos comandos, se pudesse executar as análises em **R**.

Por exemplo, a função `plot()` é utilizada para gerar gráficos. Os usuários não precisam memorizar diferentes comandos para gerar diferentes tipos de gráficos. A função `plot()` "saberá" reconhecer qual tipo de gráfico

deve ser criado a partir dos atributos do objeto que se está passando como parâmetro. Na execução da instrução `plot(notas)`, se o objeto notas for do tipo `numeric vector`, a saída do gráfico será um diagrama de pontos. Se for, por exemplo, for do tipo `factor` (fator, ou dados categóricos), a função `plot(notas)` irá gerar um gráfico de barras.

Como a função `plot()` "sabe" qual o gráfico mais apropriado a ser gerado? Internamente, a função lê as informações do objeto notas (`type`, `mode`, `length` e `attributes`) e executa o subconjunto de procedimentos (chamados métodos na linguagem de programação orientada à objeto) necessários para gerar o gráfico adequado.

Essa grande vantagem do **R** é também uma das grandes fontes de confusão para usuários iniciantes. Vamos supor que o objeto notas seja um `data.frame`. O console retornará um erro, porque a função`plot()`não tem como gerar um gráfico de um data.frame (ou de uma lista) no qual não estão identificado os valores para as coordenadas x e y.

Conhecer os diferentes tipos de objetos e como as funções atuam sobre os objetos é importante para ter um uso mais produtivo do **R**. Uma das principais fontes de erro e da percepção de dificuldade no uso do **R** está justamente na estrutura orientada à objetos, que foi criada para facilitar a interação com os usuários. Por isso, o uso da função de ajuda deve ser feito sempre que se tiver dúvida ou erro no uso de algum procedimento, por exemplo: `help(plot)` ou simplesmente `?plot`.

2 Variáveis e objetos

Em qualquer linguagem de programação, as variáveis são utilizadas para guardar valores que serão utilizados no processamento de alguma função. Como uma analogia, pode-se dizer que uma variável é um nome que identifica uma área da memória do computador, reservada para armazenar um valor. Esse valor alocado na memória permanecerá o mesmo até que seja modificado por alguma função ou procedimento do programa.

Quando uma variável é criada, por exemplo `nota <- 8.5`, um espaço na memória do computador passa a ter como endereço o identificador nota, no qual é alocado o valor 8.5. Quando se digita o no *prompt* do console a palavra `nota` o **R** procura na memória pelo endereço correspondente da variável e retorna o valor que lá estava guardado: `8.5`.

Diferente de algumas outras linguagens de programação, toda variável é armazenada como um **objeto** no **R** . Pode-se pensar em um **objeto** como um agrupamento de elementos que possuem atributos em comum. Esses atributos permitem que objetos diferentes sejam manipulados de forma específica pelas mesmas funções. Ter elementos agrupados com atributos comuns faz sentido em uma linguagem com o **R** (herdeira do **S**), que foi criada primeiramente para o tratamento e análise estatística de dados. Hoje em dia, o **R** com seus mais de 15 mil pacotes tem funções que vão muito além da estatística. Mas o tratamento de dados continua sendo a característica principal do **R** .

3 Vetores: os objetos básicos

Para o **R** , uma variável é um objeto do tipo vector (vetor). O uso do termo vetor e escalar é bastante abrangente em várias áreas do conhecimento. De modo geral, escalares (termo que origem em escala ou graus de magnitude) sempre se referem à valores singulares, vetores referem-se a um conjunto de escalares. Na matemática e na física, o sentido é semelhante, mas possuem propriedades diferentes, dependendo do campo do conhecimento que se está aplicando o conceito (campos de força, álgebra linear, etc.).

Em computação, de modo geral, uma variável com um valor singular é um escalar. Uma variável que faz referência a um conjunto de outras variáveis singulares é um vetor. No **R** , de acordo com Patrick Burns (Burns, 2011) a palavra "vetor" refere-se primeiramente ao **vetor atômico** (`atomic vector`), que é um objeto no

qual todos os elementos são do mesmo tipo, em oposição à **lista** (`list`), que é um objeto no qual os elementos podem ser de tipos diferentes.

Um vetor pode ser entendido também como objeto básico do **R,** a partir do qual são construídos os demais tipos de objetos, com tamanho (`length`) arbitrário, e sem nenhum outro atributo (`attribute`) além do nome (name), conforme avaliado pelas funções `is.vector()` e `as.vector()`.

Para facilitar a compreensão, de início, vamos adotar aqui uma definição mais simples: um vetor é o tipo de objeto mais básico do R, só tem uma dimensão e todos os elementos devem ser do mesmo tipo.

Para ilustrar essa definição, podemos comparar com uma planilha, como Excel ou Google Planilhas. Cada célula individual da planilha, no **R** se denominaria como um valor escalar (scalar). Um vetor seria uma única coluna da planilha (ou seja, uma dimensão) com todos os elementos (células na planilha ou escalares no **R**) do mesmo tipo, podendo ser números, caracteres ou do tipo lógico (verdadeiro ou falso).

Os vetores são os tipos fundamentais de objetos de dados no **R** [3]. Podem ser classificados em dois grupos: vetores atômicos (`atomic vectors`), no qual todos os elementos devem ser do mesmo tipo, e as listas (`list`), que podem ter elementos de diferentes tipos (Wickman, 2019).

Os vetores possuem três propriedades:

1. type (tipo) - pode ser um dos seguintes formatos:
2. logical (lógico) - pode assumir os valores TRUE (verdadeiro) ou FALSE (falso)[4];
3. numeric - qualquer valor numérico[5];
4. character - qualquer letra ou caractere representado entre aspas duplas ou simples[6];
5. complex (complexo) - armazena um número complexo[7].
6. raw - armazena valores em bytes[8].
7. length (tamanho) - informa a quantidade de elementos contidas no objeto[9].
8. attributes (atributos) - definem as informações que permitem diferentes formas de manipulação dos objetos[10].

As informações sobre objetos criados no **R** são mostradas na Aba *Environment* (Ambiente) do **RStudio** , conforme pode se observar na **Figura 1** :

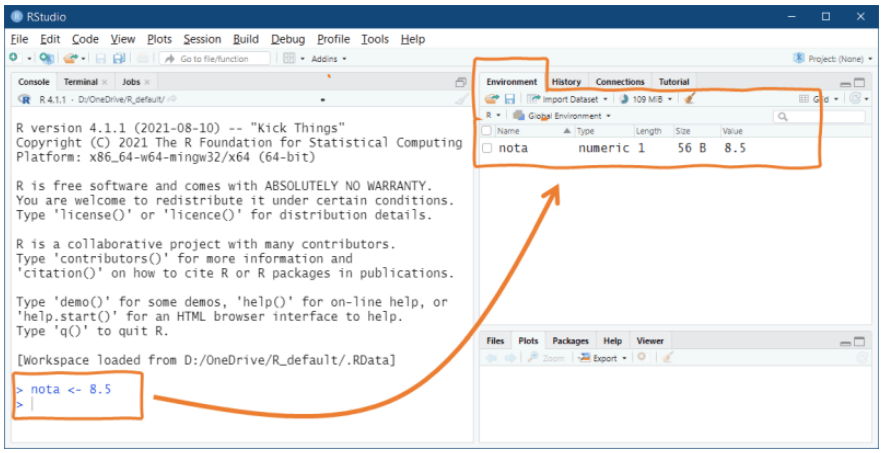

Figura 1: Detalhamento dos objetos na aba Environment (Ambiente) do RStudio

No caso, o objeto nota é um vetor do tipo (`type`) numérico (`numeric`), com um único elemento (`length`). O objeto nota possui o tamanho (size) na memória de 56 bytes, e o único valor (`value`) armazenado é 8.5 .

A função `c()` faz a combinação dos valores individuais para serem inseridos em um vetor. Vamos supor que tenhamos as notas de dez estudantes e queiramos armazená-las em uma variável do **R** , ou seja, em um objeto do tipo vetor. Vamos inserir as dez notas com o seguinte código:

```
notas <- c(7.5,10,8.5,7,9,8,4,6,8,6.5)
```

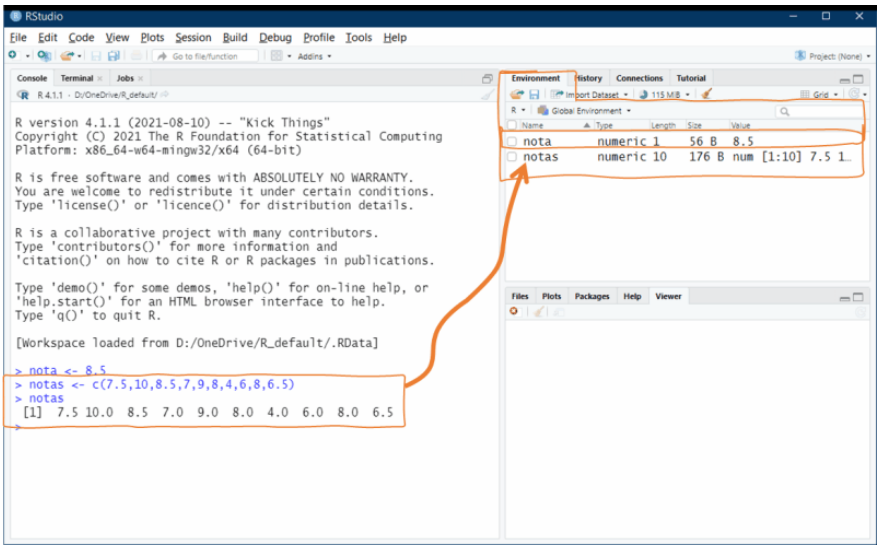

Figura 2: Detalhamento de um Vetor com mais de um elemento na aba Environment (Ambiente)

Podemos observar na **Figura 2** , que na Aba *Environment* as informações da variável (objeto) notas aparece como do tipo numeric, com length de 10 elementos e size de 176 bytes. O objeto notas, como era de se esperar, ocupa mais espaço na memória do computador do que o objeto nota. O campo value da Aba *Environment* mostra parcialmente o conteúdo do objeto notas. Para mostrar todos os valores, basta digitar o nome do objeto no *prompt* do **R** .

4 A força dos vetores

Uma das grandes vantagens do **R** está na performance resultante da vetorização das operações com dados. Vetorização em computação significa que as operações são aplicadas aos conjuntos de dados e não a cada valor em separado.

Vamos tomar como exemplo o cálculo da média das notas dos alunos.

$$média = \sum_{i=1}^{n} \frac{nota_i}{n}$$

A fórmula(ou o algoritmo) de cálculo da média é bastante conhecido. Primeiro passo soma-se todos as notas, sendo o i valor indexador, que vai de 1 (primeiro elemento escalar, ou seja, a primeira nota) - até o total n de elementos no vetor (n representa o tamanho do vetor length, o seja, o total de notas armazenadas no vetor). Segundo passo, divide-se o valor da soma pelo total de elementos n .

Vamos escrever um script que faz o cálculo da média passo a passo.

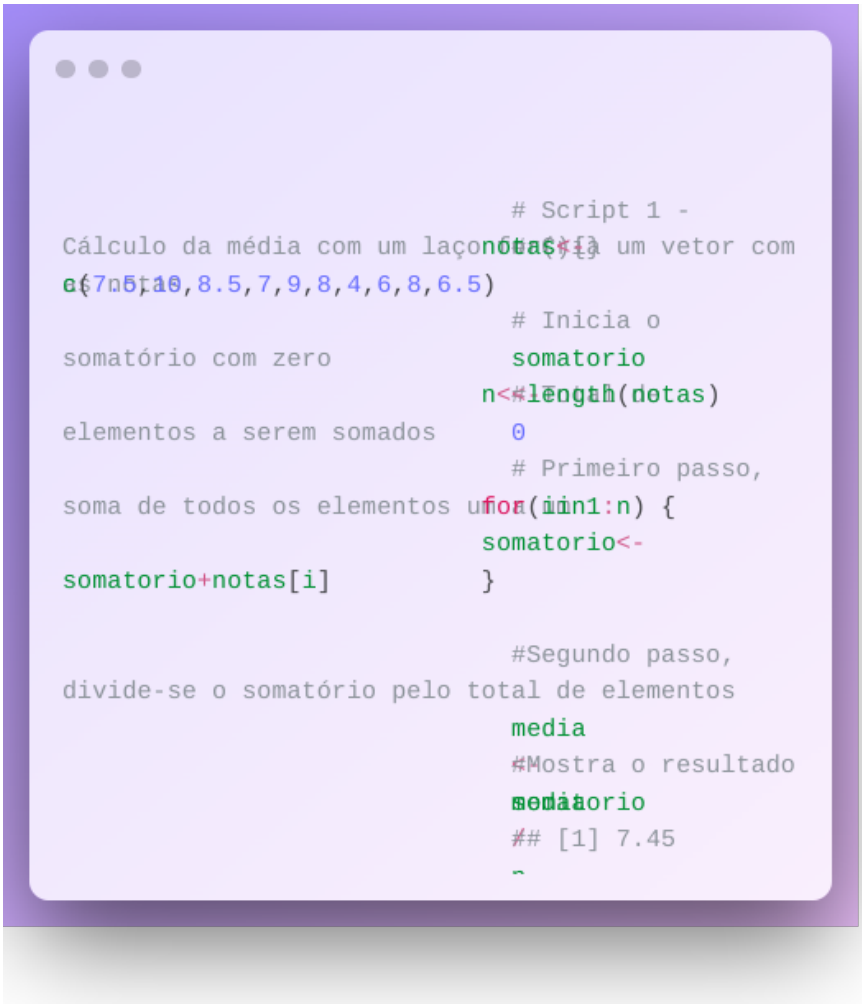

Figura 3: Cálculo da média com um laço for()

Na fórmula da média, o símbolo $\sum$ (letra grega sigma maiúscula, utilizada para indicar uma operação de somatório) representa um laço de repetição (*loop*)[12], como mostrado no *script* da Fig. 3. No **R**, Os colchetes `[ ]` são operadores utilizados para a seleção (*subsetting*) de valores dentro de um objeto, como no exemplo `notas[i]`. As chaves { } tem uma função bem diferente. São utilizadas para delimitar o início e o fim de uma função, como no caso de `for(i in 1:n) {...}`. Os parênteses servem para delimitar o escopo ou os parâmetros para a execução de uma função, como no exemplo da função `for()` ou no caso da média das notas mean(notas). Em uma expressão matemática, os parênteses também servem para definir a ordem das operações, conforme as regras da álgebra.

Na décima primeira linha da Fig. 3 , inicia-se o *loop* com a função `for(i in 1:n){`[13]. Ou seja, com o índice *i,* iniciando com o valor 1, até o total de elementos no vetor, representado por *n,* deve-se somar o valor de cada nota no vetor e guardar o resultado em um outro vetor, o que é feito na linha seguinte com o código de acumulação de valores `somatorio <- somatorio + nota[i]`.

Na décima terceira linha, a chave } fecha o loop, indicando que o computador irá repetir essa operação até o que o índice i chegue ao número 10, que corresponde ao total de elementos n no vetor notas. O valor de n foi definido na linha 8 com a instrução n <- length(notas). Vale lembrar que a função length() retorna o número de elementos contidos no vetor, no caso 10.

Esse procedimento pode ser realizado no **R** com uma única linha de código, conforme a **Fig**. 4**.**

Figura 4: Cálculo da média com a função base do R

Ambos os *scripts* mostram o resultado da média das notas, que é 7.5. O segundo código exemplifica a intenção dos desenvolvedores do **R** com a criação de uma linguagem funcional e orientada à objetos.

O script da **Fig**. 3 , procurar ilustrar um paradigma computacional procedural. O script da **Fig.** 4 ilustra o paradigma funcional, os códigos são embutidos em funções que retornam algum valor. O usuário não precisaria se preocupar com os detalhes do procedimento de execução do algoritmo. Os códigos seriam escritos próximo a como se escreve uma instrução matemática, como no caso `mean(notas)`.

Também representa a otimização para operações vetorizadas, isto é, com o conjunto de dados e não com cada elemento em separado. Quando o vetor notas é colocado como parâmetro da função `mean()`, o procedimento é realizado como um conjunto dos dados de uma vez, e não com cada valor, um por um, como na **Fig**. 3.

Essa é força da linguagem **R** , a realização de operações vetorizadas com códigos escritos, na medida do possível, de forma funcional e valendo-se do paradigma orientado a objetos. As operações vetorizadas são bem mais rápidas do que operações elemento por elemento.

Mas essas características infelizmente também representam boa parte da dificuldade que iniciantes têm com o **R** . Na verdade, o **R** é bastante versátil. O que pode, de início, se parecer mais com um enigma, com o avanço da aprendizagem, se perceberá que são características facilitadoras de programação. Com o **R** , pode-se construir códigos que representam o modelo de análise dos dados. E isso facilita a comunicação científica e a reprodutibilidade dos resultados.

5 Extraindo subconjuntos de dados em um vetor

Armazenar valores em vetores permite a aplicação de funções sobre os conjuntos dos dados de uma só vez, ao invés de forçar sempre o tratamento de cada elemento individualmente, como visto no exemplo da função que calcula a média das notas mean(notas). Essa concepção da linguagem de programação **R** pretende tornar mais eficiente tanto a execução, quanto a elaboração dos *scripts* .

A vetorização facilita também a manipulação de subconjuntos de valores em um vetor. Quase sempre, em análise de dados, procura-se informações comparando grupos de valores ou identificando casos específicos. Como os valores em um vetor são tratados como um conjunto de dados (*set*), a operação de identificar partes dos valores armazenados é chamada de manipulação de subconjuntos (*subsetting*).

No exemplo do **Script 1** foi mostrado que para acessar cada valor armazenado no vetor notas utilizou-se os colchetes`[ ]`[14]como operador. Vamos ver na **Figura 5** uma ilustração de como esse procedimento ocorre.

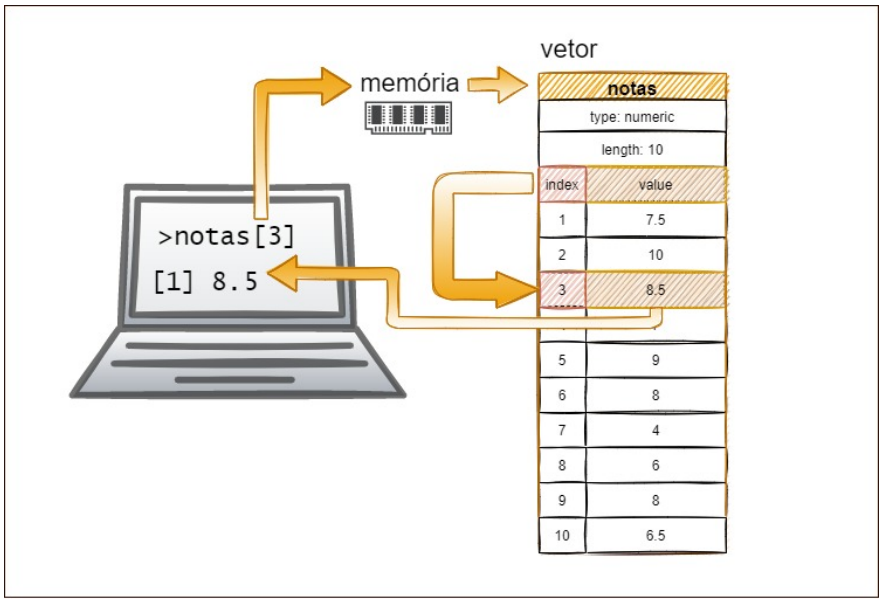

Figura 5: Valores indexados em um vetor

Quando se executa a instrução `notas[3]`, o **R** busca na memória do computador a localização do vetor que tem o nome notas. Como foi visto, um vetor é um objeto na memória do **R** que pode ser imaginado como uma coluna empilhada com valores, com cada linha identificada por um índice numérico. Vetores possuem propriedades como tipo (`numeric`) e tamanho (`length = 10`). Ao encontrar o vetor notas, o **R** segue à procura do índice igual à 3, que é o que está indicado pela notação `[3]`. Ao encontrar o valor correspondente ao índice, o **R** devolve o valor armazenado correspondente: `8.5`.

O valor do índice pode ser passado através de um outro objeto, como caso `notas[i]`. Pode-se recuperar mais de um valor de uma só vez, passando um vetor como parâmetro.

Por exemplo, `notas[1:5]` retorna as notas do índice `1` até o número `5: 7.5 10.0 8.5 7.0 9.0.`

Já a instrução `notas[c(1,5)]` retorna a primeira e a quinta nota: `7.5 9.0`. A função `c()` foi necessária porque o operador de subconjuntos `[]` requer um objeto do tipo vetor como parâmetro. A função `c()` combina uma sequência de números em um objeto do tipo vetor. Essa função será muito utilizada na seleção de valores no **R** . No primeiro exemplo `notas[1:5]` os números `1:5` não precisam estar dentro da função `c()` porque o operador de sequência : já retorna um vetor como resultado. Ou seja, `1:5` é o mesmo que `c(1,2,3,4,5)`.

Podemos também usar um operador lógico para extrair os subconjuntos em um vetor. Por exemplo, vamos verificar quais foram as notas menores do que 7

Figura 6: Subconjunto de notas menores do que 7

Um erro comum quando se usa comparativos é presumir que o interpretador do **R** reconhece o contexto em que o objeto está sendo usado. Embora possa parecer intuitivo digitar a expressão como `notas[<7]`, isso faria o **R** retornar um erro. Os operadores de comparação precisam ter um termo antes e um depois do comparador. É necessário indicar explicitamente o que se está comparando. A instrução `notas[notas < 7]` pode ser lida da seguinte forma:

- localiza o vetor **notas** na memória
- [localiza o índice do vetor notas
- identifica no índice os valores do vetor **notas < 7**
-] encerra a consulta e mostre o resultado.

Note que a instrução `notas < 7` retorna o seguinte resultado: `FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE FALSE TRUE`

Os operadores de comparação retornam valores lógicos, não os valores contidos no vetor.

Os operadores de comparação são os seguintes no **R** :

- notas[notas == 7] **igualdade** [15] *notas iguais a 7*
- notas[notas != 7] **diferença** *notas diferentes de 7*
- notas[notas <= 7] **menor ou igual** *notas menores ou iguais a 7*
- notas[notas >= 7] **maior ou igual** *notas maiores ou iguais a 7*
- notas[notas > 7] **maior** *notas maiores do que 7*
- notas[notas < 7] **menor** *notas menores do que 7*

6 Listas

Vimos que um vetor é um conjunto de valores escalares (escalares são valores individuais). Uma nota seria um valor escalar. O conjunto de notas de uma turma seria um vetor. Os vetores devem ser todos do mesmo tipo, numérico (`numeric`), lógico (`logic`), caracteres de texto (character), número complexo (complex) ou binário (`raw`). Se um vetor for construído mais de um tipo, por exemplo numérico e caractere, o**R** alocará o vetor com o tipo mais genérico, no caso, caractere.

O **R** foi desenvolvido como linguagem de análise de dados, procurando explorar a força computacional dos vetores. No **R** o tipo básico é um vetor. Mesmo a variável `nota <- 8.5` é um vetor, só que com uma única linha de dado. Se for dada a instrução nota, aparecerá no console do **R** o valor `8,5`, que é o primeiro (e único) valor armazenado no vetor. Já o vetor notas possui dez valores armazenados: `7.5 10.0 8.5 7.0 9.0 8.0 4.0 6.0 8.0 6.5`. Independentemente da quantidade de elementos[16], ambos são vetores e podem ser manipulados pelas mesmas funções.

Os vetores podem ser combinados em conjuntos maiores, para oferecer mais versatilidade para manipulação das informações. Um agrupamento de vetores no **R** é chamado de lista (list). Lista podem conter qualquer tipo de vetores (numéricos, caracteres, lógicos, complexos, binários), incluindo outras listas.

Quando a lista é criada na memória do computador, cada elemento recebe um número de ordem, como em uma lista numerada:

```
turma_l
1. matutino
    1.1. primeira_nota
    1.2. segunda_nota
    1.3. resultado_final
2. noturno
    2.1. primeira_nota
    2.1. segunda_nota
    2.3. resultado_final
```

:Para criar essa lista, utiliza-se uma cadeia de instruções que segue a mesma lógica. Deve-se lembrar que o operador <- significa atribuição de valor da direita para a esquerda, e que os elementos a serem inseridos não precisam estar na mesma linha. Uma lista é criada com a função `listt()`

Vamos criar uma lista de fato com o código do **R** , contendo as notas de duas turmas, matutino e vespertino. Cada turma terá duas notas e a avaliação final, que poderá ser "A" para aprovado e "RN" para reprovado por nota, caso a média da primeira e da segunda nota seja menor do que sete.

O **Script 4** cria a lista e armazena em **turmas_l**[17]:

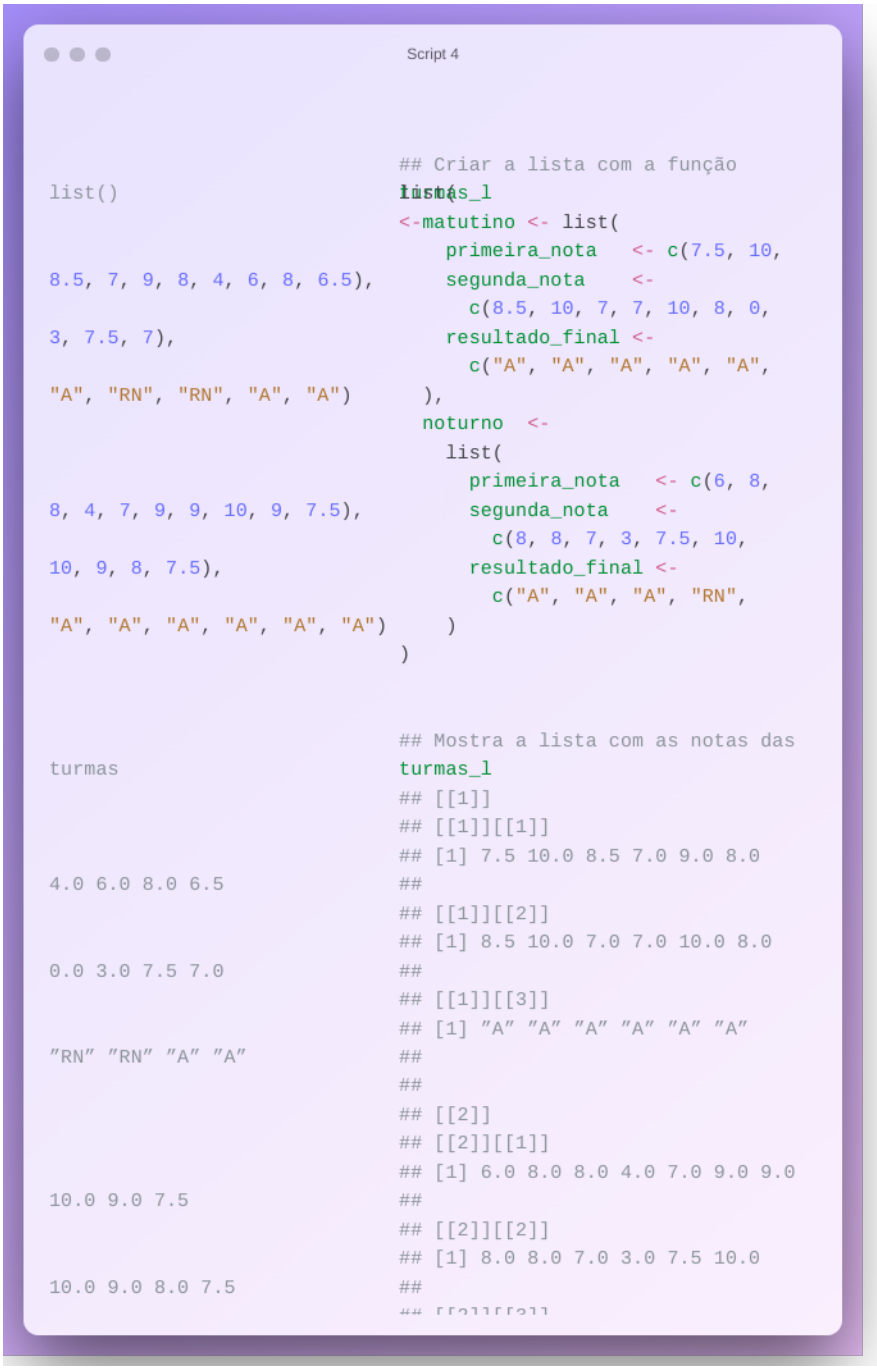

Figura 7: Cria uma lista com a função list()

Pode-se observar, no **Script 4** , que são criadas duas listas dentro da lista geral. A lista **1** , na linha 4, que recebe o nome de **matutino** , recebe três vetores: `primeira_nota` ,`segunda_nota` e `resultado_final` . A lista **2** , na linha 8, que recebe o nome de `noturno`, também recebe três vetores com os mesmos nomes da lista 1, mas com valores diferentes. Essas duas listas, `matutino` e `noturno`, cada qual com três vetores, na linha 4, são atribuídas à lista geral, chamada de `turmas_l`

Para recuperar o conteúdo pela indexação numerada (números 1.1. ou 1.2. por exemplo), utiliza-se o operador `[[ ]]`[18], conforme é mostrado no resultado do **Script 4** , ou o pode-se recuperar os valores pelo nome dos elementos, utilizando-se o operador `$`.

Por exemplo, vamos relembrar como a lista está estruturada:

"‘ turma_l 1. matutino 1.1. primeira_nota 1.2. segunda_nota 1.3. resultado_final 2. noturno 2.1. primeira_nota 2.1. segunda_nota 2.3. resultado_final

Pode ser representada no **R** da seguinte forma:

```
turma_l

[[1]]  $matutino
    [[1]][[1]] $primeira_nota
    [[1]][[2]] $segunda_nota
    [[1]][[3]] $resultado_final
[[2]]  $noturno
    [[2]][[1]] $primeira_nota
    [[2]][[1]] $segunda_nota
    [[2]][[3]] $resultado_final
```

A expressão `turmas_l$noturno$resultado_final` retornará o mesmo resultado. Pode-se extrair subconjuntos de uma lista com índices ou nomes.

Listas dão um poder muito grande aos programas em **R** para o tratamento de qualquer forma de dado. Há pacotes especializados em tratamento de listas, como {`rlist`}. Vetores e listas formam a base da manipulação de dados no **R** . Em análise de dados sociais, vamos trabalhar na maior parte do tempo com uma classe de objeto que é um tipo especial de lista, o `data.frame`.

7 Notas

1. As linguagens de programação possuem diferentes paradigmas.-*[?]*
2. Vetor e um tipo basico do R .-*[?]*
3. Os valores TRUE e FALSE podem ser substituidos por T ou F, ou 1 ou 0. As palavras em minuscula nao sao reconhecidas como **verdadeiro**e **falso** pelo **R** . Assim, *True*, *true*, *False*, *false* , *t* ou *f* retornarao um erro se forem usados.-*[?]*
4. Se nao houver especificacao em contrario, o **R** armanezara os valores todos numericos como double, isto com e, com espaco para alocar as casas decimais. O **R** possui diferentes denominacoes para os mesmos tipos de dados. Isso ocorre para manter a referencia aos diferentes padroes que contribuiram para a formulacao do programa ao longo dos anos. Originalmente, o **R** se baseou na linguagem **S** (padroes S3 e S4), que define o **modo** numeric como opcao basica. A funcao typeof() retorna o tipo do objeto conforme esta de fato armazenado na memoria do computador. Por exemplo: typeof(nota) retornara "double". A funcao mode(nota) retornara a mesma informacao, so que com a nomenclatura numeric. Um valor integer contem apenas numeros inteiros, sem fracoes. Para forcar a declaracao de um objeto como literalmente integer (e nao double ou numeric) deve-se adicionar um L (de Literal) ao final do numero. Exemplo: nota <- 10L . Forcar a definicao de numero inteiro tem utilidade em situacoes especiais, quando, por exemplo, se necessita de precisao numerica de calculo, dado que computadores fazem arredondamos nos valores decimais (conhecido como erro de ponto flutuante) que podem, em alguns casos, trazer resultados inesperados. Pode-se forcar os valores numeric ou double a serem armazenados como inteiros tambem com a funcao as.integer(). Quando um valor double e transformado em integer, a parte decimal e perdida. Por exemplo, as.integer(8.5) resultara em 8 . Para forcar que valores integer sejam tratados como double usa-se a funcao as.double() ou as.numeric().

Na maioria das vezes nao e necessario se preocupar com essas diferencas. O **R** convertera os valores integere double automaticamente quando necessario para a realizacao dos calculos matematicos, sem que o usuario precise se preocupar com isso. Na Aba Environment do **RStudio** o tipo do objeto e mostrado como o modo, por exemplo: um valor numerico como 10 ou 8.5, sera armazenado como double e sera mostrado como do tipo numeric. Um valor forcado com a funcao as.integer() ou como 10L sera armazenado e mostrado como integer. Um valor integer convertido com a funcao as.double() ou as.numeric() sera armazenado como double e mostrado como tipo numeric na Aba *Environment* do**RStudio** .-*[?]*

5. Um elemento do tipo character e passado para um vetor como valores entre aspas " duplas ou ' simples . Por exemplo, turma <- c("matutino","vespertino"). Em algumas linguagens, ha diferenca entre o uso de aspas simples e duplas. Em **R** ,nao existe diferenca. As aspas duplas sao mais comumente utilizadas. As aspas simples sao utilizadas quando se quer incluir as aspas duplas na cadeia de caracteres (string). Um numero pode ser inserido como character. Por exemplo: cod_ibge <- "310567". Nesse caso, o numero esta armazenado como simbolo e nao como tipo numeric. Nao sera possivel fazer calculos com esse valor. Um vetor e um objeto no qual todos os elementos sao do mesmo tipo. Quando se cria o vetor com um valor character entre outros numericos, por exemplo: sequencia <- c(1,2,3,"4",5,6), todos os demais elementos sao convertidos tambem em character. A funcao typeof(sequencia) informara que o tipo do vetor sequencia e character.-*[?]*
6. Numeros complexos sao pares ordenados, escritos na forma normal com uma parte real e uma parte imaginaria representada pela letra i, como por exemplo 5+2i . Um numero imaginario e igual a raiz quadrada de -1. Numeros complexos nao sao muito comuns em analise estatisticas de dados e nao utilizaremos esse tipo de dados nessa fase de aprendizagem do R. Esses numeros tem uma utilizacao maior em engenharia, sobretudo em engenharia eletrica, e matematica, por exemplo na resolucao de equacoes polinomiais. Para ciencia dos dados, uma aplicacao dos numeros imaginarios esta na algebra matricial. Em ciencia social computacional sao utilizados na compreensao de fenomenos complexos, a partir da base matematica dos fractais. Mas, numeros complexos nao devem ser vistos como algo exotico ou dificil de ser aprendido. Na verdade, adolescentes no Brasil aprendem essa materia como parte do Curriculo Minimo de Matematica do Ensino Fundamental. Geralmente, esse conteudo e ofertado na terceira serie do ensino medio. Caso tenha curiosidade e queira rememorar seus numeros imaginarios, assista as videoaulas sobre numeros complexos da Khan Academy em portugues, disponivel em: https://pt.khanacademy.org/math/algebra-home/alg-complex-numbers-*[?]*
7. Por exemplo, o codigo infosoc_raw <- charToRaw("INFOSOC") converte a palavra "INFOSOC" em *bytes* e armazena no objeto infosoc_raw. O valor armazenado retorna sete elementos: 49 4e 46 4f 53 4f 43 que correspondem as letras I N F O S O C . O tipo raw raramente e aplicado em analise de dados e nao vamos fazer uso desse recurso nessa fase de aprendizagem do **R** .-*[?]*
8. No caso de um objeto atomic vector e de uma matrix a propriedade length retorna o numero de elementos. No caso de um objeto do tipo list a propriedade length retorna o numero de componentes da lista. Para um objeto do tipo data.frame a propriedade length retorna o numero de colunas, isto e, o numero de variaveis.-*[?]*
9. Os atributos sao metadados, ou seja, informacoes que definem como um objeto devera ser manipulado por uma funcao. Por exemplo, uma matriz e um vetor que tem o atributo dimensoes (dimensions). Todos os principais tipos de objetos usados em analise de dados com o**R** , como matrizes (matrix) e fatores (factor) sao vetores com atributos especificos. A funcao que mostra e altera os atributos de um objeto e attr(). A funcao attributes() tambem permite recuperar ou visualizar os atributos de um ojbeto. A funcao structure() e usada para atribuir ou modificar os atributos de um objeto.-*[?]*
10. Se voce tiver duvida ou nao conhece a notacao simbolica do somatorio em matematica, assista o seguinte video da Khan academy em portugues: Notacao de somatorio (video) | Series | Khan Academy-*[?]*
11. Lacos ou loops sao estruturas fundamentais em logica de programacao. Veremos mais sobre isso adiante quando for abordado o conteudo sobre a criacao de funcoes no **R** . Caso queira conhecer mais detalhes sobre os lacos com a funcao for() e os lacos de repeticao, confira em Loops no R: usando o for() | Analise Real (analisereal.com). Confira tambem o Livro de Jackson Aquino R para cientistas sociais

(uesc.br) (de Aquino, 2019) , capitulo 10.7*Loops* for e *While* , p. 131.-*[?]*

12. O simbolo de dois pontos : e um operador de sequencia. Pode ser ler a instrucao 1:n da seguinte forma: *de 1 ate n* .-*[?]*
13. Os colchetes [ ] sao operadores utilizados para a selecao (subsetting) de valores dentro de um objeto, como no exemplo notas[i]. As chaves { } tem uma funcao bem diferente. Sao utilizadas para delimitar o inicio e o fim de uma funcao, como no caso de for(i in 1:n) {. . . }. Os parenteses servem para delimitar o escopo ou os parametros para a execucao de uma funcao, como no exemplo da funcao for() ou no caso da media das notas mean(notas). Em uma expressao matematica, os parenteses tambem servem para definir a ordem das operacoes, conforme as regras da algebra.-*[?]*
14. O operador logico de igualde e um sinal de igual duplo ==O sinal de igualdade simples = indica atribuicao de valor a uma variavel. i = 7 atribuicao - o vetor i recebe o valor 7 i == 7 comparacao - retorna TRUE porque i tem o valor 7.-*[?]*
15. O limite maximo de elmentos em um vetor e determinado pela disponibilidade de memoria RAM (Random Access Memory) do computador. Mas, nao pode ultrapassar 2^{31} - 1 elementos (mais de dois bilhoes, ou, precisamente, 2.147.483.647 elementos).-*[?]*
16. Vamos acrescentar o sufixo _l ao nome turmas para indicar que se trata de uma lista. Isso nao e necessario, mas ajuda a diferenciar os tipos de objetos com os quais estamos trabalhando.-*[?]*
17. Quando se extrai um subconjunto de um vetor, usa-se o operador [], como nos exemplos anteriores com vetores. Pode-se usar o mesmo operador em uma lista. A principal diferenca entre [] e [[]], segundo o manual do R, e que o primeiro permite selecionar varios elementos de uma so vez, enquanto o segundo so permite um elemento. Outra diferenca e que [] retorna uma lista com a mesma estrutura de nomes da lista original. O operador [[]] nao preserva os nomes, apenas os valores dos elementos. De modo pratico, usa-se com mais frequencia []para vetores, matrizes e data.frames. Utiliza-se [[]] para selecionar valores unicos aninhados (recursivamente) em listas. Para mais informacoes, consulte a ajuda do R, digitando o comando ¿‘[” no prompt do**R.** -*[?]*

Referências

More R numbers. (2011). https://www.burns-stat.com/documents/tutorials/impatient-r/more-r-key-objects/more-r-numbers/. https://www.burns-stat.com/documents/tutorials/impatient-r/more-r-key-objects/more-r-numbers/

Advanced R. (2019). Chapman Hall/CRC. http://adv-r.had.co.nz/

R para Cientistas Sociais. (2019). UESC.