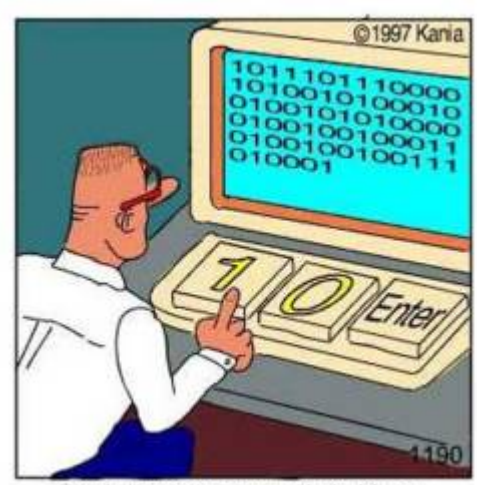
- Tutorial
- Exercícios
- Apostila

1a. Introdução ao R: bases da linguagem



Real programmers code in binary.

Entre as várias características que definem uma linguagem computacional está a forma como o código é implementado pelos sistema operacional, ou seja, como a linguagem do programa é transformada em linguagem de máquina do sistema operacional. Há dois tipos básicos de implementação: compilação e interpretação. O R faz parte do segundo grupo e por isso podemos conversar com o programa a cada linha de comando. Além disso, nossa conversa com o R é baseada em uma linguagem de alto nível, significando que nossa comunicação com o programa é similar à linguagem humana e se distancia da linguagem da máquina que é binária, só contendo zeros e uns.

Outra característica do R é que ele é uma linguagem orientada a objetos, ou seja, manipulamos objetos com procedimentos inerentes à classe a que eles pertencem. Essas características do R fazem com que esse ambiente de programação seja similar uma oficina com matéria-prima (objetos) e ferramentas que manipulam esses objetos [1] e com isso podemos construir outros objetos, ou '*obras de arte virtual*' em nosso ateliê. Vamos entrar nessa oficina!

ateliêR

Clique no botão RUN abaixo!

```
<iframe width='100%' height='400'
src='https://rdrr.io/snippets/embed/?code=print(%22Hello%2C%20world!%22)'
frameborder='0'></iframe>
```

Os computeiros dizem que a primeira coisa que deve ser feita ao aprender uma linguagem computacional é faze-la dizer Hello, world!. Pronto! Já fizemos nossa primeira tarefa na linguagem R!

No código acima, ao clicar em **RUN** enviamos o comando `print("Hello, world!")` para o interpretador do R. Apesar da simplicidade desse exemplo temos alguns conceitos básicos da sintaxe do R que são importantes.

Note que temos no comando acima caracteres (letras, símbolos e espaços em branco) que estão entre aspas "`Hello, world!`". Essa é nossa primeira lição: **O que está entre aspas o R interpreta como sendo caracteres**. Parece óbvio, mas veja o que acontece ao rodar o código abaixo:

```
<iframe width='100%' height='400'
```

```
src='https://rdrr.io/snippets/embed/?code=print(Hello)'
frameborder='0'></iframe>
```

A mensagem `Error: object 'Hello' not found`, significa que o R não encontrou o objeto com o nome `Hello`. Nossa segunda lição: **caracteres que não estão entre aspas o R intrepreta como sendo objetos**. No caso, o objeto `Hello` não foi encontrado!

Atribuição

Ok! E como fazemos para criar um objeto no R?

ATRIBUIÇÃO NO R

- junção dos caracteres < e - : atribuição à esquerda;
- caracter = : o mesmo que acima, atribuição à esquerda;
- junção de - e > : atribuição à direita;

Nas regras de boas práticas de estilo da linguagem, em geral, se diz que deve-se usar a primeira forma, que a segunda é aceitável, mas que não devemos usar a terceira!

Vamos criar nosso primeiro objeto no R!

```
<iframe width='100%' height='400'
src='https://rdrr.io/snippets/embed/?code=Hello%20<-%20%22Hello%2C%20world!%22' frameborder='0'></iframe>
```

Parece que nada aconteceu, mas de fato, atribuímos ao objeto chamado `Hello` os caracteres que compoem a frase `Hello, world!`. Após criar o objeto podemos manipulá-lo ou apenas chamá-lo para exibir o que foi atribuído a ele.

```
  <iframe width='100%' height='400'
src='https://rdrr.io/snippets/embed/?code=Hello%20%3C-%20%22Hello%2C%20world!%22%0AHello' frameborder='0'></iframe>
```

Agora temos novamente o retorno de “`Hello, world!`”, mas dessa vez a frase vem do objeto `Hello`. Quando chamamos um objeto que existe no R ele nos retorna o que está armazenado nele.

Sintaxe básica

No nosso primeiro código do R havia um objeto chamado `print`. Vamos visualizar o seu conteúdo chamando-o no R.

```
<iframe width='100%' height='400'
src='https://rdrr.io/snippets/embed/?code=print' frameborder='0'></iframe>
```

O retorno parece estranho:

```
function (x, ...)
UseMethod("print")
<bytecode: 0x23d3468>
<environment: namespace:base>
```

Vamos nos ater apenas na primeira linha, caso queira mais informação veja o comentário [2)]. Ela nos diz que esse objeto é uma função e nos mostra o código que é executado pela funcão . Ou seja, a função `print` é um objeto da classe `function`. Os objeto da classe `function` em geral estão associados a uma documentação que nos ajudam a entender como usar essa ferramenta! Para acessar a documentação no R, utilizamos outra ferramenta que é a função `help`[3)].

```
<iframe width='100%' height='400'
src='https://rdrr.io/snippets/embed/?code=help(print)'
frameborder='0'></iframe>
```

Uma questão interessante aqui é que estamos usando uma ferramenta, a função `help`, para manipular o objeto `print`, que por sua vez também é uma função! Será que o `help` tem help?

```
<iframe width='100%' height='400'
src='https://rdrr.io/snippets/embed/?code=help(help)'
frameborder='0'></iframe>
```

É muito importante diferenciar o objeto que contém o código, que é a função, do procedimento ao executar essa função. A diferença entre um e outro está em um detalhe pequeno que são os parênteses (...) que acompanham o nome da função. O nome da função acompanhado dos parênteses fazem com que o procedimento associado a esse objeto seja executado! Caso não seja acompanhada dos parênteses, o objeto da classe função irá retornar aquilo que está atribuído a ele: o texto de código que a funçao contém.

Na documentação da função `print` há a descrição de argumentos que, entre outras coisas, flexibilizam o procedimento da função. O primeiro argumento, chamado `x` é o objeto que será manipulado, um outro argumento dessa função é o `digits`. Vamos usá-lo:

```
<iframe width='100%' height='400'
src='https://rdrr.io/snippets/embed/?code=print(x= 1.23456789, digits = 3)'
frameborder='0'></iframe>
```

Para explicitar que estamos manipulando objetos, podemos fazer o procedimento em duas estapas, primeiro atribuíndo o valor `1,23456789` a um objeto e depois solicitando para que ele seja mostrado na tela com apenas 3 digitos.

```
<iframe width='100%' height='500'
src='https://rdrr.io/snippets/embed/?code=numero%20%3C-%201.23456789%20%20%0A
print(x%20%3D%20numero%2C%20digits%20%3D%203)%0A%20'
frameborder='0'></iframe>
```

Agora vamos ver a diferença na manipulação que o `print` faz, dependendo da classe do objeto:

```
<iframe width='100%' height='500'
src='https://rdrr.io/snippets/embed/?code=numero%20%3C-%201.23456789%20%20%0A
palavra%20%3C-
%20%221.23456789%22%0Aprint(x%20%3D%20numero%2C%20digits%20%3D%203)%0Aprint(x
```

%20%3D%20palavra%2C%20digits%20%3D%203)%20' frameborder='0'></iframe>

Porque o objeto `numero` é manipulado diferentemente do objeto `palavra`? Por que são objetos de classes diferentes e a função `print` reconhece essa diferença e trata eles de forma diferente. Quanto manipula números o argumento `digits` faz sentido, quando o objeto é da classe `characters` esse argumento é desprezado. Aqui tem um conceito avançado da linguagem, a função `print` chama um método que executa diferentes procedimentos dependendo da classe do objeto que ela manipula. Podemos dizer que o método é um conjunto de funções. Para acessar a classe a que um objeto pertence usamos a função `class`.

<iframe width='100%' height='500'
src='https://rdrr.io/snippets/embed/?code=numero%20%3C-%201.23456789%20%20%0A
palavra%20%3C-%20%221.23456789%22%0Aclass(numero)%0Aclass(palavra)'
frameborder='0'></iframe>

Vamos agora usar uma outra função para exemplificar a sintaxe básica do R. A função em questão é `round`, que arredonda um valor numérico até a casa decimal solicitada, diferentemente do `print` que não modifica o valor, apenas imprime ele com o número de casa decimais solicitado. O `round`, por sua vez, faz a transformação do valor arredondando o valor.

A primeira ação que deve ter ao utilizar uma função no R é: **SEMPRE LER A DOCUMENTAÇÃO**. A documentação do `round` descreve que ele também tem um argumento chamado `digits`.

  <iframe width='100%' height='500'
src='https://rdrr.io/snippets/embed/?code=numero%20%3C-%201.23456789%20%20%0A
outro%20%3C-%20round(numero%2C%20digits%20%3D%204)%0Aoutro'
frameborder='0'></iframe>

A sintaxe básica do R pode ser definida como:

```
object <- tool(x, arg2 = y, arg3 = z)
```

Podemos ler o comando acima como sendo: “utilize a ferramenta `tool` para manipular o objeto `x` tendo o argumento `arg2` com o atributo `y` e a opção `arq3` como `z`. O resultado dessa manipulação é armazenado no objeto `object`. Note que o R, nesse caso, não devolveria nada na tela, pois o resultado da manipulação é atribuído a um objeto.

Estrutura e tipos de dados

Até aqui vimos dois tipos de informação que podem ser manipuladas no R: caracteres e números. Os números, por sua vez, podem ser de dois tipos: números com decimais (`numeric`) e inteiros (`integer`). Essa distinção é importante para a maneira como o R armazena essa informação na memória do computador, de resto elas funcionam como números racionais na matemática clássica. No capítulo seguinte vamos tratar das funções matemáticas mais a fundo. Aqui vamos apenas ver as bases conceituais dos tipos de dados básicos e qual a estrutura básica de armazenamento em objetos. As operações da algebra básicas no R usam os mesmos simbolos que na matemática tradicional: `+`, `-`, `/` e `*`.

<iframe width='100%' height='500' src='https://rdrr.io/snippets/embed/?code=10%20%2B%2010%0A10%20-%2010%0A10%2F10%0A10%20*%2010' frameborder='0'></iframe>

Como vimos na sessão anterior podemos atribuir valores numéricos a um objeto. Depois disso, podemos manipular os valores indiretamente por intermédio do objeto.

<iframe width='100%' height='500' src='https://rdrr.io/snippets/embed/?code=dez%20%3C-%2010%20%0Adez%20-%20dez%0Adez%20%2F%20dez%0Adez%20*%20dez' frameborder='0'></iframe>

Atribuimos o valor `10` ao objeto `dez` e depois manipulamos o objeto `dez`. Isso não parece ser uma vantagem. Estamos trocando dois digitos, o valor `10`, por um objeto que contem 3 letras, o `dez`. A vantagem começa quando atribuímos o resultado de operações a um outro objeto.

<iframe width='100%' height='500' src='https://rdrr.io/snippets/embed/?code=dez%20%3C-%2010%20%0Acem%20%3C-%20dez%20*%20dez%0Adezmil%20%3C-%20cem%20*%20cem%0Acem%20%0Adezmil' frameborder='0'></iframe>

Vetores

Ficaria ainda melhor se pudessemos operar mais de um valor de uma vez. Como armazenar mais de um valor em um objeto? Usamos uma função `c` que significa **c**oncatenar ou **c**ombinar. Os elementos combinados são a estrutura básica de dados no R, que é o objecto da classe `vector`. Esse é o elemento básico dos objetos no R. Mesmo que o objeto só tenha um elemento, trata-se de um vetor com uma posição.

  <iframe width='100%' height='500' src='https://rdrr.io/snippets/embed/?code=contadez%20%3C-%20c(1%2C%202%2C%203%2C%204%2C%205%2C%206%2C%207%2C%208%2C%209%2C%2010)%0Acontadez' frameborder='0'></iframe>

Indexação de Vetores

Note que, antes de iniciar a apresentação dos valores que estão no vetor `contadez` o R apresenta o valor 1 entre colchetes `[1]`. Caso o nosso vetor fosse longo e tivesse que ser apresentado em várias linhas, outros valores em colchetes iriam iniciar essa outras linhas de apresentação dos dados. Esses valores representam a indexação do elemento que inicia a linha de apresentação do conteúdo do vetor. Ou seja, o elemento na posição 1, no nosso caso é o valor 1. Vamos inverter esse vetor, em seguida combiná-lo com o vector anterior!

<iframe width='100%' height='500' src='https://rdrr.io/snippets/embed/?code=contadez%20%3C-%20c(1%2C%202%2C%203%2C%204%2C%205%2C%206%2C%207%2C%208%2C%209%2C%2010)%0Ainvertedez%20%3C-%20rev(contadez)%0Adescontadez%20%3C-%20c(invertedez%2C%20contadez)%0Adescontadez' frameborder='0'></iframe>

Agora, como fazemos para acessar algum elemento dentro desse vetor? Para isso usamos a

indexação de posição.

Por padrão no R o primeiro elemento está na posição 1.

Essa frase pouco informativa é uma detalhe importante. Em muitas linguagem computacionais, diria até que a maioria das linguagens mais populares, a indexação começar pela posição definida como (zero)! Mais a frente vamos usar outras indexações de vetores e de outras classes de objetos de dados. Abaixo temos alguns exemplos, simples para vetores:

```
<iframe width='100%' height='500'
src='https://rdrr.io/snippets/embed/?code=contadez%20%3C-%20c(1%2C%202%2C%203
%2C%204%2C%205%2C%206%2C%207%2C%208%2C%209%2C%2010)%0Ainvertedez%20%3C-
%20rev(contadez)%0Adescontadez%20%3C-
%20c(invertedez%2C%20contadez)%0Adescontadez%0Adescontadez%5B7%5D%0Adescontad
ez%5Bc(1%2C5%2C10%2C20)%5D' frameborder='0'></iframe>
```

Classes Date

Crie objetos com as datas do tri e tetracampeonatos mundiais do Brasil[4]:

```
copa70 <- "21/06/70"
copa94 <- "17/07/94"
```

Qual a diferença em dias entre estas datas? A subtração retorna um erro (verifique):

```
copa94 - copa70
```

```
<iframe width='100%' height='500'
src='https://rdrr.io/snippets/embed/?code=copa70%20%3C-%20%2221%2F06%2F70%22%
0Acopa94%20%3C-%20%2217%2F07%2F94%22%20%0Acopa94%20-%20copa70'
frameborder='0'></iframe>
```

Isto acontece porque os objetos são caracteres, uma classe que obviamente não permite operações aritméticas. Já sabemos verificar a classe de um objeto, digitando o código:

```
class(copa70)
class(copa94)
```

O resultado seria `character` para ambos!

Mas o R tem uma classe para datas, que é `Date`. Vamos fazer a coerção [5] dos objetos para esta classe, verificar se a coerção foi bem sucedida, e repitir a subtração. O código para isso está descrito abaixo:

```
copa70 <- as.Date(copa70,format="%d/%m/%y")
copa94 <- as.Date(copa94,format="%d/%m/%y")
class(copa70)
class(copa94)
copa94 - copa70
```

NOTA: o argumento `format` da função `as.Date` informa o formato em que está o conjunto de caracteres que deve ser transformado em data, no caso dia/mês/ano (`%d/%m/%y`), todos com dois algarismos. Veja a ajuda da função para outros formatos.

Inclua na janela do R online abaixo o código que gera os objetos `copa70` e `cop94`, em seguida verifique a classe a que pertencem, e depois faça a a tranformação para a classe `Date` e a subtração entre eles. Não coloque linhas de comando que geram mensagens de erro ou, pode comentá-as iniciando-as com # . O hagtag ou mais comumente com ## , indica ao R que essa linha é um comentario e não deve ser interpretada.

```
<iframe width='100%' height='500'
src='https://rdrr.io/snippets/embed/?code=%23%23%23%23%23%23%23%23%23%23%23%2
3%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%
23%23%23%23%23%23%23%23%23%0A%23%23%20Digite%20seu%20c%C3%B3digo%20abaixo%20d
esse%20coment%C3%A1rio!%0A%23%23%20O%20R%20n%C3%A3o%20interpreta%20linhas%20c
ome%C3%A7ando%20com%20%23%0A%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%
23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23
%23%23%23%23%0A' frameborder='0'></iframe>
```

Dado Lógico

Até o momento, vimos algumas naturezas de informação que podemos armazenar e manipular no R: caracteres, datas e números. Uma outra natureza importante de dado básico no R é chamada de lógica. As palavras TRUE e FALSE e também as abreviações T e F são reservadas para esse fim. Uma questão importante dos dados lógicos é que a eles também são associadas os valores 0 e 1, para FALSE e TRUE, respectivamente. Veja abaixo como podemos operá-los algebricamente:

```
<iframe width='100%' height='500'
src='https://rdrr.io/snippets/embed/?code=TRUE%20%2B%20TRUE%0ATRUE%2FFALSE%0A
TRUE%20*%20FALSE' frameborder='0'></iframe>
```

Além disso, o R retorna TRUE ou FALSE quando fazemos alguma procedimento utilizando operadores lógica.

Operadores Lógicos

- == : igual
- != : diferente
- > : maior que
- < : menor que
- >= : maior ou igual
- < =: menor ou igual

Alguns exemplos de operações lógicas no R:

```
<iframe width='100%' height='500'
src='https://rdrr.io/snippets/embed/?code=numero%20%3C-%201.23456789%0Anumero
```

%20%3C%201%0Anumero%20%3E%201%0Aprint(numero%2C%20digits%20%3D%204)%20%3D%3D%20numero%0Around(numero%2C%20digits%20%3D%204)%20%3D%3D%20numero%0A' frameborder='0'></iframe>

A operação lógica também funciona com vetores, obedecendo a posição dos elementos:

  <iframe width='100%' height='500' src='https://rdrr.io/snippets/embed/?code=contadez%20%3C-%20c(1%2C2%2C3%2C4%2C5%2C6%2C7%2C8%2C9%2C10)%0Ainvertedez%20%3C-%20rev(contadez)%0Ainvertedez%0Ainvertedez%20%3E%20contadez' frameborder='0'></iframe>

Classe Fator

Imagine um experimento em que classificamos as plantas em uma escala de herbivoria com os níveis: “alto”, “médio”, “baixo” e “nulo”. Vamos criar um objeto que representa o valor desta medida de herbivoria em uma amostra de 14 plantas:

```
herb <- c("A","M","M","A","A","M","M","B","A","A","A","A","B","A")
```

E então criar um objeto da classe fator com estes valores:

```
herbFactor <- factor(herb)
```

Usamos a função `table` para contar o número de observações em cada nível do fator, cujo resultado atribuímos a um outro objeto. Os valores são exibidos se digitamos o nome do objeto.

```
herbTable <- table(herbFactor)
herbTable
```

A função para gerar gráficos `plot` pode ser aplicada diretamente ao objeto desta tabela:

```
plot(herbTable)
```

Rode o código abaixo e avalie o que está sendo produzido em cada linha de comando . Caso fique com dúvidas a primeira coisa a fazer é consultar o `help()` da função. O quadro onde temos o código abaixo, pode ser editado e pode rodar novamente com outro código. Fique a vontade para explorar a documentação das funções que estamos apresentando.

  <iframe width='100%' height='600' src='https://rdrr.io/snippets/embed/?code=herb%20%3C-%20c(%22A%22%2C%22M%22%2C%22M%22%2C%22A%22%2C%22A%22%2C%22M%22%2C%22M%22%2C%22B%22%2C%22A%22%2C%22A%22%2C%22A%22%2C%22A%22%2C%22B%22%2C%22A%22)%0AherbFactor%20%3C-%20factor(herb)%0AherbTable%20%3C-%20table(herb)%0AherbTable%0Aplot(herbTable)' frameborder='0'></iframe>

Note que na tabela e na figura os níveis não estão ordenados da forma como deveriam e falta o nível de herbivoria nula. Isto acontece porque, ao criar uma variável de fator a partir de um vetor de valores, o R cria níveis apenas para os valores presentes, e ordena estes níveis alfabeticamente. Caso

um nível não tenha sido observado nos dados, ele fica de fora da variável, mas o correto seria ele ter a contagem como `0` .

Para ordenar o fator e incluir um nível que não foi representado na amostra usamos o argumento `levels` da função `fator`:

```
herbFactor <- factor(herb, levels=c("N","B","M","A"))
```

Modifique o código da janela acima, incluíndo o argumento `levels` na função `factor` e rode novamente o código todo na janela abai

```
<iframe width='100%' height='500'
src='https://rdrr.io/snippets/embed/?code=%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%0A%23%23%20Digite%20seu%20c%C3%B3digo%20abaixo%20desse%20coment%C3%A1rio!%0A%23%23%20O%20R%20n%C3%A3o%20interpreta%20linhas%20come%C3%A7ando%20com%20%23%0A%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%23%0A' frameborder='0'></iframe>
```

NOTA: há uma classe para fatores ordenados que poderia se aplicar aqui, mas seu uso tem implicações importantes nos resultados de algumas análises, que no momento não vêm ao caso. Mais informações a respeito na ajuda da função factor.

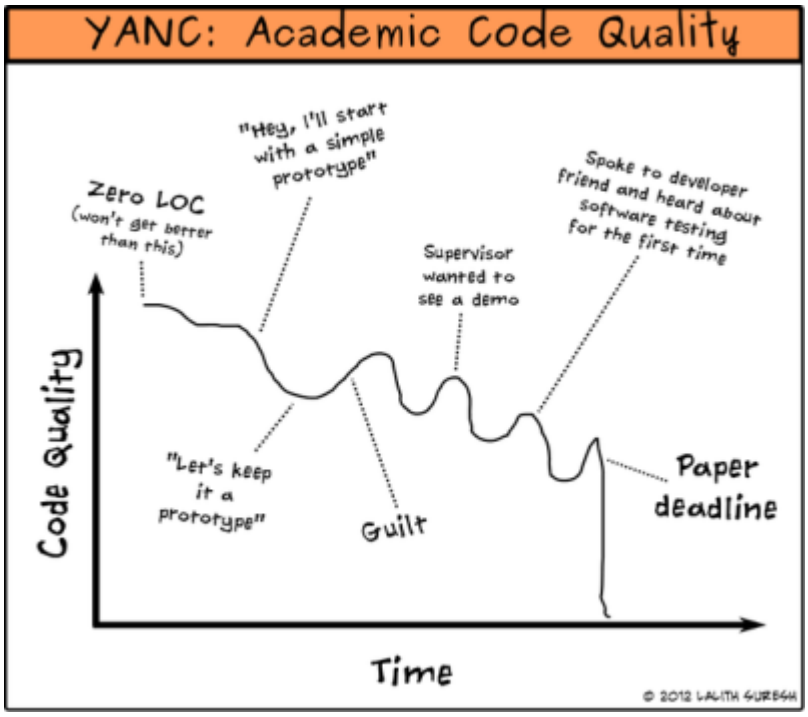
O Código

Antes de continuar a introdução aos conceitos básicos do R, vamos enteder uma conduta importante em programação. Um dos primeiros hábitos que você deve adquirir para trabalhar com o R é **não digitar os comandos diretamente no console do R**[6], e sim em um arquivo texto, que chamamos de ***script*** ou ***código***. Essa intermediação entre o texto do comando e o interpretador, feita pelo script, é importante pois garante que o que está sendo direcionado ao R é armazenado no arquivo texto, que por fim, pode ser salvo e armazenado no computador, como um registro do procedimento executado e para ser utilizar novamente quando necessário.

Reprodutibilidade do procedimento

Quando trabalhamos em uma planilha eletrônica, a partir de dados brutos, podemos salvar os gráficos ou os dados modificados após manipulados. Entretanto, o procedimento não é salvo. Se precisar fazer o mesmo procedimento para outro conjunto de dados precisará lembrar todas as etapas e a ordem em que foram executadas. Em programação, o script é nosso roteiro do procedimento que foi executado. Para repetir um procedimento é só executar novamente o script. Isso incrementa muito a reprodutibilidade do nosso procedimento, uma qualidade muito importante para a ciência de um modo geral, mas também para o dia a dia. Por isso, a partir desse momento no curso, iremos abondonar a interface do R online que estavamos usando para rodar o código e vamos, a patir de agora, produzir script ou códigos!

Editor de Código



Um editor de código nada mais é do que um editor de texto puro como o bloco de notas do Windows. Algumas funcionalidades são bem vindas, como por exemplo, enviar a linha de código diretamente para o console do R sem a necessidade de copiar e colar.

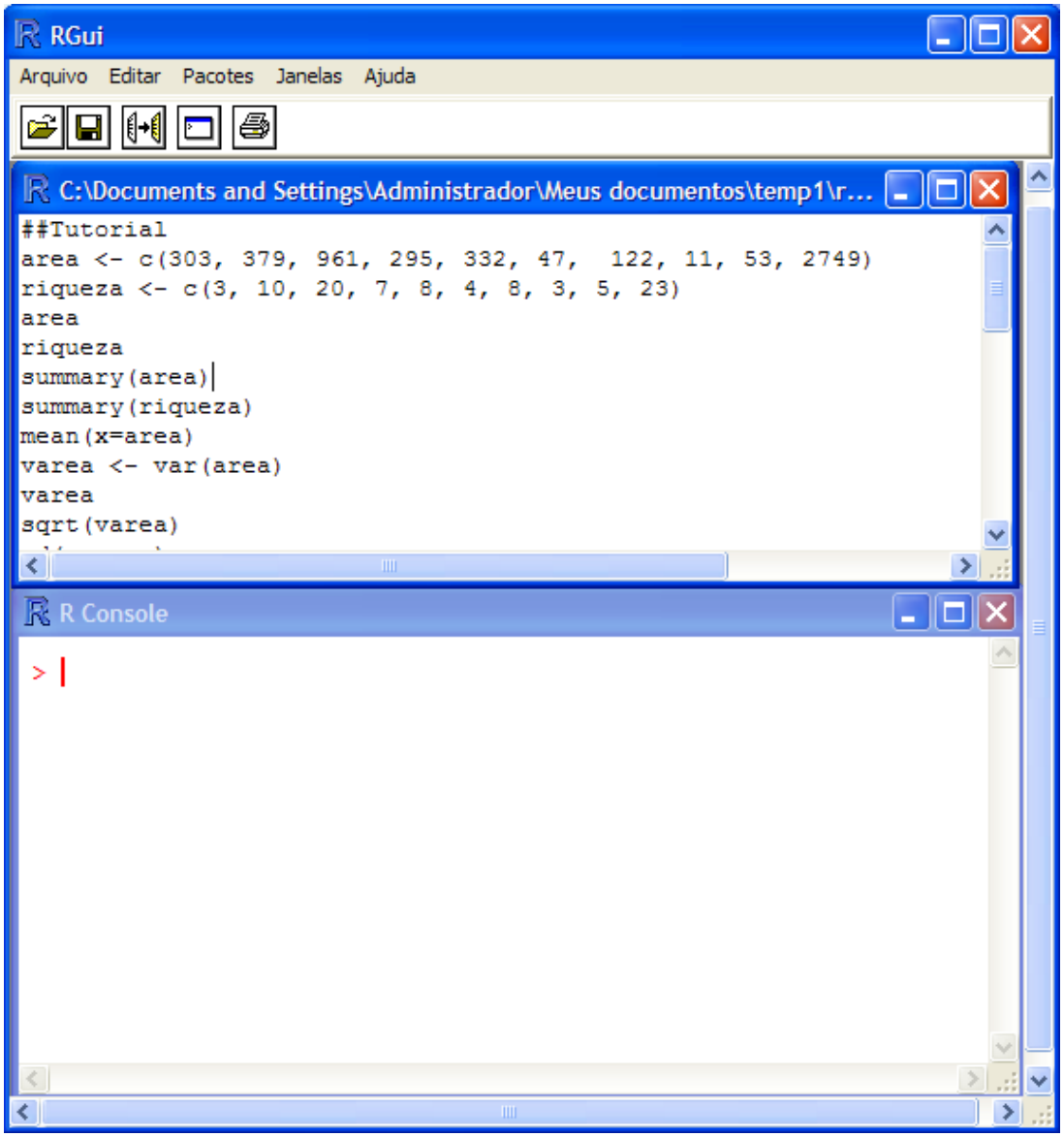
A instalação básica do R contém uma interface gráfica de usuário (R-GUI) simples, tanto no Windows como no IOS, que acompanha um editor de códigos.

O editor de códigos do R-GUI no Windows e no Mac é bastante simples e costuma ser uma boa opção inicial para usuários deste sistema. Para esta disciplina ele é suficiente.

No Linux não uma há uma GUI padrão para o R, e esta escolha deve ser feita logo no início.

Na página de material de apoio há uma seção com várias dicas sobre interfaces para o R para lhe ajudar.

A figura abaixo é uma captura de tela do R-GUI do Windows, mas no MAC o editor é similar, e você pode manter a mesma lógica. Deixe sempre uma janela de código aberta acima da janela do R, como na imagem abaixo:



Interface de usuário R-GUI

Na figura acima há duas janelas com funcionamentos e objetivos muito distintos.

1. a janela da parte superior apresenta um arquivo de texto puro que pode ser editado e salvo como texto. Por padrão salvamos esses arquivos com a extensão `.r` ou `.R` para reconhecermos que é um script da linguagem R. O sistema operacional deve reconhecer a extensão com sendo do R automaticamente.
2. a janela na parte inferior é o console do R, ou seja o programa propriamente dito. Essa janela recebe os comandos de código e envia ao interpretador do R, que por sua vez, retorna o resultado final do processamento[7)].

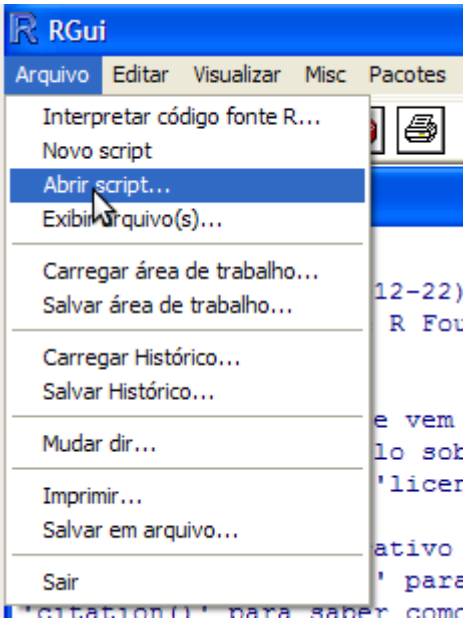
Para evitar confusão e perda de trabalho é importante digitar as informações que serão transmitidas ao R (linhas de código) no arquivo texto e ir passando esses comandos ao R. Uma boa prática também é comentar as linhas de código para que outras pessoas, ou mesmo a pessoa que criou o código, possam entender ou lembrar o que o código executa.

É imprescindível aprender a se organizar dentro da lógica do ambiente de

programação, com o risco de perder trabalho ou ficar completamente perdido entre as tarefas que executa.

O primeiro Script

- Copie todas as linhas de códigos que foram processados nesse tutorial até o momento em arquivo texto simples no bloco de nota do Windows ou algum outro programa simples de texto (TextEdit no macOS);
- Salve o arquivo em uma pasta [8] conhecida do seu computador, associada a essa disciplina, com o nome e extensão `tutorial01.r` [9];
- Execute o R e abra o *script* que salvou, utilizando a opção do menu "Arquivo/Abrir script":



- Vá para a janela do *script*, coloque o cursor na primeira linha e tecle `Ctrl-r`. Faça o mesmo com as linhas seguintes;

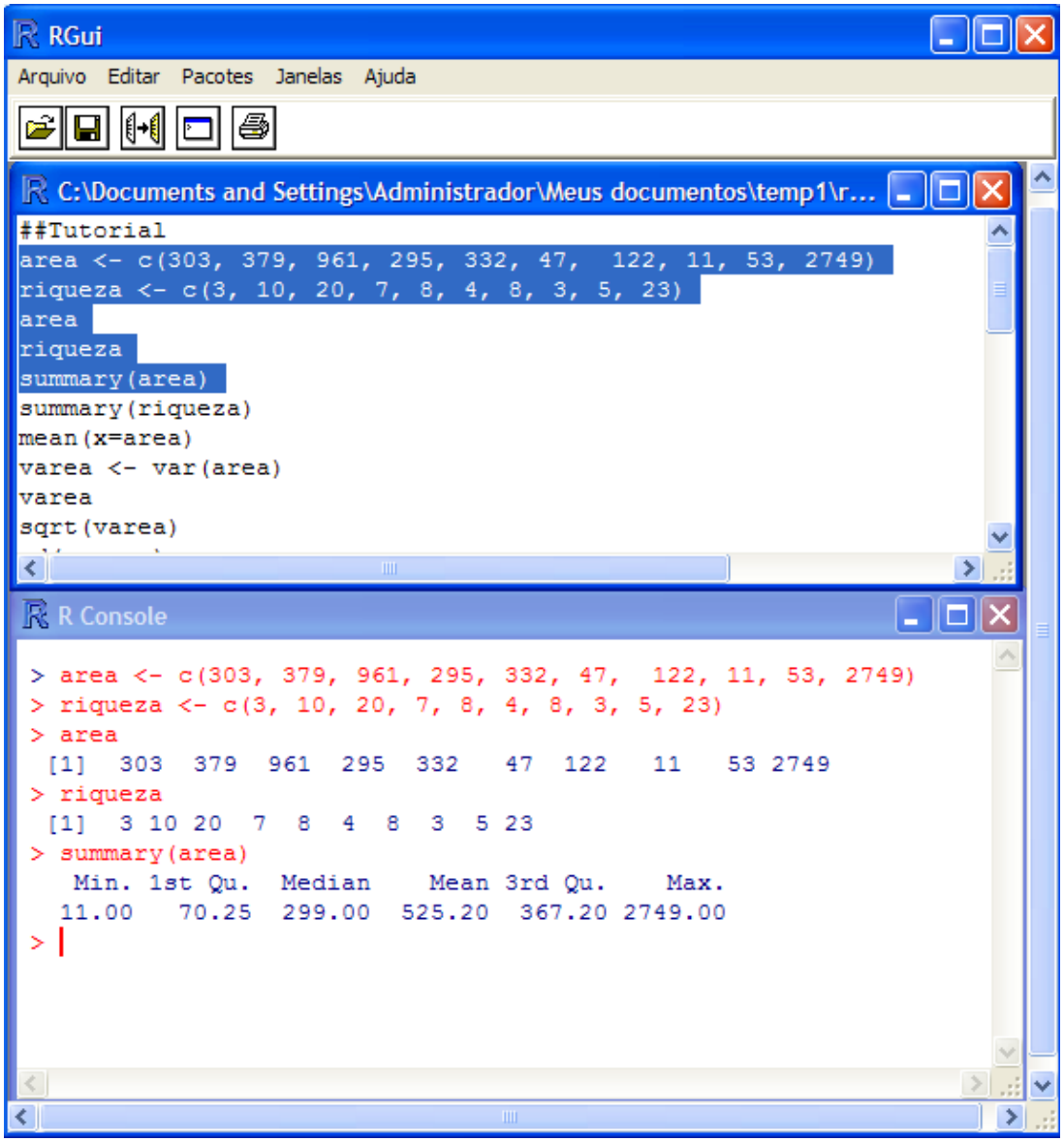
Para Usuários de MAC

Para enviar comandos do editor de código do R-GUI para o R utilize *Command+Enter* ou *Command+Return*.

Veja o material RMacOSX

- Coloque o título no arquivo como sendo `## Tutorial Introducao ao R`
- Na linha seguinte coloque a data como comentário: `## Data: ….`;
- Comente cada bloco de código com o nome do tópico que o código está associado no roteiro
- Comente ou retire qualquer linha de código que tenha gerado erros durante o processamento;
- Retire a redundância na atribuição dos abjetos, mas cuidado com objetos que são sobrescritos, veja o `copa70`, por exemplo;
- Ao final selecione todas as linhas do script, inclusive comentários e e tecle `Ctrl-r` para submeter tudo ao interpretador do R;

- Garanta que não há mesagens de erro ao longo do processamento do script.



Comentários no código

Para fazer comentários no código, usamos o simbolo de # . Qualquer conteúdo na linha de comando depois do # não é interpretado pelo R. Utilizamos os comentários, em geral, para tornar o código autoexplicativo.

- Salve o *script* com estas modificações.

Agora vamos submeter o *script* salvo no nosso sistema de correção automática de código, chamado **notaR**.

Agora que já fez seu primeiro exercício no notaR. Siga para a aba de exercícios para seguir os execícios desse tópico. Os exercícios ficarão embutidos nesse wiki, mas deixaremos sempre o link para o notaR caso prefiram abrir a plataforma

diretamente. **Lembre-se de logar no sitema notaR** antes de fazer os exercícios e não deixe de passar pela aba da apostila, ela e complementar aos tutoriais, apesar de alguma redundância desejável.

1)

funções também são uma classe de objetos

2)

é uma função que chama um método que é um generalização de ferramentas. Podemos dizer que o método é, por exemplo, uma faca. Quando esse método é aplicado a algum objeto, por exemplo, um pão, ele chama a função especifica para aquele objeto: a faca de pão!

3)

O caracter ? funciona como um atalho para essa função

4)

fonte: FIFA

5)

procedimento de transformar uma classe em outra

6)

Console é a interface de interação com o interpretador da linguagem: recebe o comando, envia ao interpretador e retorna a resposta. O que vinhamos usando no início desse tutorial é um interpretador online do R

7)

quando a tarefa solicitada é a representação de um gráfico, uma nova janela é aberta, um dispositivo gráfico.

8)

diretório de trabalho é o nome técnico desta pasta para o R

9)

quando o sistema operacional não mostra a extensão dos arquivos é preciso configura-lo para que seja apresentado