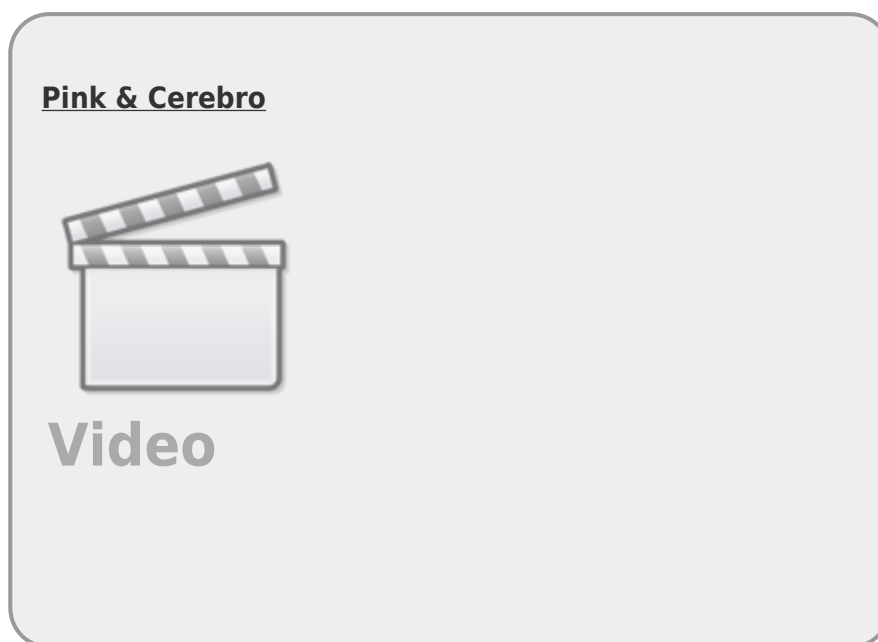


- [Tutorial](#)
- [Exercícios](#)
- [Apostila](#)

9. Criando Funções Básicas

Nossos Personagens



Pinky and the Brain, no Brasil, **Pinky e o Cérebro**, são personagens de uma série animada de televisão norte-americana. São dois ratos brancos de laboratório que utilizam os Laboratórios Acme como base para seus planos mirabolantes para dominar o mundo (sob razão nunca revelada). Pink é um rato totalmente estúpido e ingênuo enquanto o Cérebro é o gênio perverso que comanda os planos de conquista do mundo.

Cada episódio é caracterizado, tanto no início quanto no final, pela famosa tirada onde Pinky pergunta: “*Cérebro, o que faremos amanhã à noite?*” e Cérebro responde: “*A mesma coisa que fazemos todas as noites, Pinky... Tentar conquistar o mundo!*”

O paralelo entre os personagens e os usuários do **R**, é que estamos em uma constante transformação entre PINK e Cérebro, tentando vencer nossa ignorância. A resposta do nosso *Cérebro* à pergunta do *Pink* que habita dentro de nós é: ***A mesma coisa que fazemos todas as noites, Pink... terminar essa função para conquistar o mundo!***

O *PINK* que habita dentro de nós



Classe Function

Um objeto de função é uma classe especial de objetos no R que encapsulam algum tarefa. A função `function()` recebe algumas informações através dos seus argumentos e usa essas informações internamente para a realização do procedimento desejado. O procedimento ou tarefa realizada por uma função é conduzido por um algoritmo, que nada mais é que um sequência de comandos concatenados. Quando o objeto de função é chamado, ele retorna o que os outros objetos retornam no console do R, o que foi atribuído a ele, ou seja essas linhas de comando no formato de texto. Entretanto, quando o objeto da classe `function` é chamada com os parênteses `()` ele realiza a tarefa conduzido pelo conjunto concatenados de comandos que foram atribuídos a ele. Ao final, normalmente, a função retorna um **único** objeto que contém o resultado do procedimento. Para o retorno deste objeto, normalmente usamos uma outra função chamada `return()`.

FAZENDO VERSÕES PIORADAS DE FUNÇÕES EXISTENTES!!!!

Vamos construir algumas funções simples. A primeira é uma função que calcula a média de um conjunto de valores. A primeira etapa é definir qual o tipo de objetos que a função irá manipular e designar um nome a esse objeto como um argumento. No caso da média podemos definir esse objeto como um vetor numérico `x`. Em seguida precisamos definir o algoritmo que será executado. Uma forma de fazer isso é através de um pseudocódigo que é a descrição literal do algoritmo. No nosso caso:

Pseudocodigo media - recebe um vetor x - soma os valores do vetor no objeto soma - guarda o tamanho do vetor x em nobs - divide soma por nobs e guarda no objeto media - retorna o objeto med </WRAP> Depois de definir o que a função conterà, precisamos abrir um bloco de código para conter as linhas de comando que definem o algoritmo com as chaves `{ }`. Em seguida colocamos as linhas de comando descritas no pseudocodigo e rodamos todo o bloco para construir a

função, como abaixo: Uma funções muito simples
<code rsplus> media ←function(x) { soma ← sum(x)
nobs ← length(x) med ← soma/nobs return(med) }
</code>

Note que a função nada mais é do que um conjunto de linhas de comando concatenadas para executar uma tarefa. A princípio quem usa as funções básicas do R já está qualificado a fazer funções mais complexas a partir delas.

==== Testando a função ==== **<code rsplus> ls()**
media media() dados ← rnorm(20, 2, 1) media(dados)
dados1 ← rnorm(200, 2, 1) media(dados1) dados2 ←
(rnorm(10000, 2, 1)) media(dados2) sd(dados) dados3 ←
rnorm(20, 2, 0.01) media(dados3) dados4 ←
rnorm(200,2,0.01) media(dados4) dados[2] ← NA dados
media(dados) </code> ===== Uma função mais elaborada
 ===== A função padrão do R (**mean ()**) não calcula a média quando há NA no vetor de dados, a menos que o usuário utilize o argumento **na.rm = TRUE**. Vamos construir uma função que diferente da função padrão, calcule a média na presença de NA e imprima na tela uma mensagem sobre o número de NA removidos do cálculo. Note que é uma função com dois argumentos, que permite ao usuário tomar a decisão de remover ou não NA. Diferente da função **mean ()** o padrão é invertido, nossa função remove NA se nenhum argumento for mencionado. Note que vamos sobrepor o objeto anterior da classe função, chamado **media**. **<code rsplus> media ← function(x, rmNA = TRUE) { if(rmNA == TRUE) { dados ← (na.omit(x)) n.NA ← length(x) - length(dados) cat("\t", n.NA," valores NA excluídos\n") } else { dados ← x } soma ← sum(dados) nobs ← length(dados) media ← soma/nobs return(media) } </code>** Calcular a média do vetor dados **media(dados) ===== Função para calcular variância**
 ===== **<code rsplus> var.curso ← function(x) { media ←media(x) dados ← na.omit(x) disvquad ← (dados - media)^2 var.curso ← sum(disvquad)/(length(dados)-1) return(var.curso) } </code>** Calcular a variância de dados e comparando com a função do R! **var.curso(dados) var(dados) ### dica: veja o help dessa função "help(var)" var(dados, na.rm = TRUE) var(dados,na.rm = FALSE) =====**
Função para calcular o Índice de Dispersão ===== Os
índices de dispersão nos ajudam a avaliar se

contagens por amostras estão distribuídas de modo aleatório, agregado ou uniforme. Veja o material de aula para entender como a relação variância por média pode dar uma idéia do tipo de distribuição espacial, quando temos contagens de indivíduos em várias parcelas de igual tamanho.

```
<code rsplus>
ID.curso ← function(x) { id ← var.curso(x)/media(x)
return(id) } </code> ===== Simulando dados com
parâmetros conhecidos ===== Tomando dados
simulados de contagem de uma espécie em uma
amostra de 20 parcelas de 20x20m, podemos verificar
o padrão de dispersão dessa espécie, utilizando o
Índice de Dispersão (razão variância / média) Vamos
simular dados com diferentes características
conhecidas: * Simulando Aleatório <code rsplus>
aleat ← rpois(200, 2) aleat </code> * Uniforme <code
rsplus> unif ← runif(200, 0, 4) unif unif ← round(unif,
0) unif </code> * Agregado <code rsplus> agreg ←
round(c(runif(100, 0, 1), runif(100, 5, 10))) agreg
</code> Calcular o coeficiente de dispersão <code
rsplus> ID.curso(aleat) ID.curso(unif) ID.curso(agreg)
</code>
```

Quando o valor é próximo a 1 a distribuição é considerada aleatória. Isto quer dizer que a ocorrência de cada indivíduo na parcela é independente da ocorrência das demais. Neste caso, o número de indivíduos por parcela é descrito por uma variável Poisson, que tem exatamente a média igual à variância. Podemos então fazer um teste de hipótese simulando uma distribuição Poisson com a mesma média dos dados.

```
===== Função para criar o teste de hipótese do ID =====
<code rsplus> test.ID ← function(x, nsim=1000) { ID.curso ←
function(x){var(x)/mean(x)} # essa função precisa das
funcoes media e ID.curso dados ← na.omit(x) ndados ←
length(dados) med ← mean(dados) id ← var(dados)/med
simula.aleat ← rpois(length(dados)*nsim, lambda=med)
sim.dados ← matrix(simula.aleat,ncol= ndados) sim.ID ←
apply(sim.dados,1,ID.curso) quant.ID ← quantile(sim.ID,
probs=c(0.025,0.975)) if(id >= quant.ID[1] & id <=
quant.ID[2]) { cat("\n\n\t distribuição aleatória para alfa =
0.05 \n\t ID= ",id,"\n\n") } if(id < quant.ID[1]) {
cat("\n\n\t distribuição uniforme, p<0.025 \n\t ID=
",id,"\n\n") } if(id>quant.ID[2]) { cat("\n\n\t distribuição
agregado, p>0.975 \n\t ID= ",id,"\n\n") }
```

```

resulta=c(id,quant.ID) names(resulta)←c("Índice de
Dispersão", "crítico 0.025", "crítico 0.975") return(resulta) }
</code> Testando os dados simulados test.ID(aleat)
test.ID(agreg) test.ID(unif) ===== Outra função =====
eda.shape <code rsplus> eda.shape ← function(x) { x11()
par(mfrow = c(2,2)) ## muda o dispositivo gráfico para 2x2
hist(x) ## produz histograma de x boxplot(x) iqd ←
summary(x)[5] - summary(x)[2] ## faz a diferença entre o
quinto elemento x e o segundo
plot(density(x,width=2*iqd),xlab="x",ylab="",type="l")
qqnorm(x) qqline(x) par(mfrow=c(1,1)) } </code> Criando
um vetor de dados com 20 valores simulando a densidade
de árvores por parcelas <code rsplus> set.seed(22) ##
estabelece uma semente aleatória
dados.pois20←rpois(20,lambda=6) ## sorteia dados
aleatórios de uma função poisson com média 6
sum(dados.pois20) ## a somatória aqui sempre dará 131,
somente porque a semente é a mesma set.seed(22)
dados.norm20←rnorm(20,mean=6, sd=2) ## sorteia 20
dados de uma função normal com média 6 e dp = 1 sum
(dados.norm20) ### aqui o resultado dará sempre 130.48
###aplicar eda.shape para dados.dens
eda.shape(dados.pois20) eda.shape(dados.norm20)
###aumentando a amostra eda.shape(rpois(500,6))
eda.shape(rnorm(500,6)) </code> ===== Modificando uma
função ===== <code rsplus> eda.shape1 ← function(x) {
x11() par(mfrow = c(2,2)) hist(x,main="Histograma de x")
boxplot(x, main="BoxPlot de x") iqd ← summary(x)[5] -
summary(x)[2]
plot(density(x,width=2*iqd),xlab="x",ylab="",type="l",
main="Distribuição de densidade de x")
qqnorm(x,col="red",main="Gráfico Quantil x
Quantil",xlab="Quantil Teórico",ylab="Quantil da Amostra")
qqline(x) par(mfrow=c(1,1)) } </code> ===== Executando a
função modificada ===== eda.shape1(rnorm(500,6))
===== Fazendo ciclos de operações ===== Um outro
instrumento importante para programar em R é o loop ou
ciclos. Ele permite a aplicação de uma função ou tarefa a
uma sequência pré determinada de dados. Ou seja, repete a
mesma sequência de comandos um número determinado de
vezes. Simulando dados de novo! <code rsplus> x1 ←
rpois(20, 1) x2 ← rpois(20, 2) x3 ← rpois(20, 3) x4 ← rpois(20,
1) sp.oc ← matrix(c(x1, x2, x3, x4), ncol=4) colnames(sp.oc)
← c("plot A", "plot B", "plot C", "plot D") rownames(sp.oc) ←
paste("sp", c(1:20)) str(sp.oc) dim(sp.oc) head(sp.oc)
</code> Uma função para contar espécies por parcelas.
Mais uma vez uma função já existente em versão piorada!!
<code rsplus> n.spp ←function(dados) { nplot ←
dim(dados)[2] resultados ← rep(0,nplot) names(resultados) ←
paste("n.spp", c(1:nplot)) dados[dados>0] = 1 for(i in
1:(dim(dados)[2])) { cont.sp ← sum(dados[,i]) resultados[i] ←

```

```
cont.sp } return(resultados) } ##### Aplicando a função
n.spp(sp.oc) </code>
```

Uma dica para entender qualquer função é rodar cada uma das linhas separadamente no console do R, na mesma sequência que aparecem e verificar os objetos intermediários criados. Quando chegar a um ciclo, pule a linha do *for()* e rode as linhas subsequentes, porém designe antes algum valor para o contador, no nosso exemplo *i* (tente *i=2*). A lógica da função *for()* é que o contador (*i*) terá um valor diferente a cada ciclo, no exemplo entre 1 até o número de colunas do objeto *dados*. Além disso, o contador pode ser usado para indexar a posição onde o resultado de cada ciclo será colocado no objeto final (*resultados*)

```
===== Mais função!! SIMILARIDADE ===== <code
rsplus> sim<-function(dados) { nplot <- dim(dados)[2] similar
<- matrix(1,ncol=nplot,nrow=nplot) rownames(similar) <-
paste("plot", c(1:nplot)) colnames(similar) <- paste("plot",
c(1:nplot)) dados[dados>0] = 1 for(i in 1:nplot-1) { m=i+1
for(m in m:nplot) { co.oc <- sum(dados[,i]>0 & dados[,m]>0)
total.sp <- sum(dados[,i]) + sum(dados[,m]) - co.oc
similar[i,m] <- co.oc/total.sp similar[m,i] <- co.oc/total.sp } }
return(similar) } </code> ===== Aplicando a função SIM
===== <code rsplus> sim(sp.oc) debug(sim) sim(sp.oc)
undebug(sim) </code> MUITO BEM VC. JÁ ESTÁ SE
TRANSFORMANDO, NÃO PARECE MAIS UM PINK**
```



Agora faça os [Exercícios 9 - Construção de Funções](#) para podermos conquistar o MUNDO!!!

From:

<http://ecor.ib.usp.br/> - **ecoR**

Permanent link:

http://ecor.ib.usp.br/doku.php?id=02_tutoriais:tutorial8:start&rev=1656262252



Last update: **2022/06/26 13:50**