| Nesta lição, veremos como extrair elementos de um vetor com base em condições que podemos especificar.

...

|=== | 2%

| Vou dar alguns exemplos: Nós podemos estar interessados apenas nos 20 primeiros elementos de um vetor, ou

| apenas nos elementos que não são NA, ou apenas naqueles que são positivos ou correspondem a uma variável

| específica de interesse. No final desta lição, você saberá como lidar com cada um desses cenários.

...

|===== | 5%

| Eu criei para você um vetor chamado x que contém 20 números aleatórios e 20 NAs. Digite x agora para ver seu

| conteúdo.

> x

[1] -1.41187701 0.14041059 NA 1.44665833 NA 0.17373786 NA 0.48504045

[9] NA NA -1.85117177 0.63428744 NA 0.42570388 1.58361499 NA

[17] -0.28772704 NA NA NA NA NA NA NA

[25] -1.30275810 -0.10371976 -0.11134464 -0.12514674 -0.93480333 0.38004226 NA NA

[33] NA 1.32200575 NA -0.06142626 NA -0.94727621 NA -1.59179723

| Mantenha esse bom nível!

|======== | 8%

| A maneira como você diz ao R que deseja selecionar alguns elementos específicos (isto é, um "subconjunto")

| de um vetor é colocando um "vetor de índice" entre colchetes imediatamente após o nome do vetor.

...

|========== | 10%

| Para um exemplo simples, digite x[1:10] para ver os primeiros dez elementos de x.

> x[1:10]

[1] -1.4118770 0.1404106 NA 1.4466583 NA 0.1737379 NA 0.4850405 NA

[10] NA

| Na mosca! Bom trabalho!

|============= | 12%

| Lembre-se que 1:10 resulta em um vetor de 10 posições.

...

|=============== | 15%

| Os vetores de índices (ou vetores de filtros) podem ser de quatro tipos diferentes - vetores lógicos,

| vetores de inteiros positivos, vetores de inteiros negativos e vetores nomeados (com strings) - cada um dos

| quais abordaremos nesta lição.

...

|================== | 18%

| Vamos começar indexando com vetores lógicos. Um cenário comum ao trabalhar com dados do mundo real é quando

| queremos extrair todos os elementos de um vetor que não são NA (ou seja, dados ausentes). Lembre-se de que

| is.na(x) produz um vetor de valores lógicos com o mesmo comprimento de x, com TRUEs correspondendo aos

| valores de NA em x e FALSEs correspondendo a valores não-NA em x.

...

|==================== | 20%

| O que você acha que x[is.na(x)] lhe dará?

1: Um vetor de TRUEs e FALSEs

2: Um vetor de comprimento 0

3: Um vetor sem NAs

4: Um vetor de todos os NAs

Selection: 1

| Quase! Tente novamente.

| Lembre-se de que is.na(x) nos diz onde os NAs estão em um vetor. Então, se subfiltrar x com base nisso, o

| que você espera que aconteça?

1: Um vetor sem NAs

2: Um vetor de todos os NAs

3: Um vetor de TRUEs e FALSEs

4: Um vetor de comprimento 0

Selection: 2

| Bom trabalho!

|======================= | 22%

| Veja se isso é verdade, digitando x[is.na(x)].

>

> x[is.na(x)]

[1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA

| Todo o trabalho duro está dando resultado!

|========================= | 25%

| Lembre que `!` Nos dá a negação de uma expressão lógica, então !is.na(x) pode ser lido como 'não é NA'.

| Portanto, se quisermos criar um vetor chamado y que contenha todos os valores não-NA de x, podemos usar y <-

| x[!is.na(x)]. Veja se funciona.

> y <- x[!is.na(x)]

| Continue assim e você chegará lá!

|============================ | 28%

| Imprima y no console.

> y

[1] -1.41187701 0.14041059 1.44665833 0.17373786 0.48504045 -1.85117177 0.63428744 0.42570388

[9] 1.58361499 -0.28772704 -1.30275810 -0.10371976 -0.11134464 -0.12514674 -0.93480333 0.38004226

[17] 1.32200575 -0.06142626 -0.94727621 -1.59179723

| Continue assim e você chegará lá!

|============================== | 30%

| Agora que nós isolamos os valores não-NA de x e os colocamos em y, podemos usar y como quisermos.

...

|================================= | 32%

| Lembre-se que a expressão y > 0 nos dará um vetor de valores lógicos com o mesmo comprimento que y, com

| TRUEs correspondendo a valores de y que são maiores que zero e FALSEs correspondendo a valores de y que são

| menores ou iguais a zero. O que você acha que y[y > 0] vai lhe dar?

1: Um vetor de todos os NAs

2: Um vetor de comprimento 0

3: Um vetor de todos os elementos negativos de y

4: Um vetor de TRUEs e FALSEs

5: Um vetor de todos os elementos positivos de y

Selection: 5

| Toda a prática está rendendo frutos!

|=================================== | 35%

| Digite y[y > 0] para ver se obtemos todos os elementos positivos de y, que também são os elementos positivos

| do nosso vetor x original.

> y[y>0]

[1] 0.1404106 1.4466583 0.1737379 0.4850405 0.6342874 0.4257039 1.5836150 0.3800423 1.3220058

| Essa é a resposta que eu esperava.

|====================================== | 38%

| Você pode se perguntar por que não começamos apenas com x[x > 0] para isolar os elementos positivos de x.

| Tente isso agora para ver o motivo.

> x[x>0]

[1] 0.1404106 NA 1.4466583 NA 0.1737379 NA 0.4850405 NA NA 0.6342874

[11] NA 0.4257039 1.5836150 NA NA NA NA NA NA NA

[21] NA 0.3800423 NA NA NA 1.3220058 NA NA NA

| Essa é a resposta que eu esperava.

|======================================== | 40%

| Como NA não é um valor, mas sim um espaço reservado para uma quantidade desconhecida, a expressão NA > 0 é

| avaliada como NA. Por isso, temos um monte de NAs misturados com nossos números positivos quando fazemos

| isso.

...

|=========================================== | 43%

| Combinando nosso conhecimento de operadores lógicos com nosso novo conhecimento de subconjuntos, poderíamos

| fazer isso: x[!is.na(x) & x > 0]. Experimente.

> x[!is.na(x) & x > 0]

[1] 0.1404106 1.4466583 0.1737379 0.4850405 0.6342874 0.4257039 1.5836150 0.3800423 1.3220058

| Excelente trabalho!

|============================================= | 45%

| Neste caso, solicitamos apenas valores de x que são ambos não ausentes E maiores que zero.

...

|================================================ | 48%

| Eu já mostrei como subdividir apenas os primeiros dez valores de x usando x[1:10]. Nesse caso, estamos

| fornecendo um vetor de inteiros positivos dentro dos colchetes, que informa R para retornar apenas os

| elementos de x numerados de 1 a 10.

...

|================================================== | 50%

| Muitas linguagens de programação usam o que é chamado de 'indexação baseada em zero', o que significa que o

| primeiro elemento de um vetor é considerado elemento 0. R usa 'indexação baseada em um', que você certamente

| já percebeu! Significa o primeiro elemento de um vetor. é considerado o elemento 1.

...

|===================================================== | 52%

| Você consegue descobrir como filtrar os elementos 3, 5 e 7 de x? Dica - Use a função c() para especificar os

| números dos elementos como um vetor numérico.

> x[c(3,5,7)]

[1] NA NA NA

| Todo o trabalho duro está dando resultado!

|======================================================== | 55%

| É importante que, ao usar vetores inteiros para subconjunto de nosso vetor x, permaneçamos apenas com o

| conjunto de índices {1, 2, ..., 40}, já que x possui apenas 40 elementos. O que acontece se tentar buscar o

| elemento zero de x (ex: x[0])? Tenta aí.

> x[0]

numeric(0)

| Você está indo muito bem!

|========================================================== | 58%

| Como você poderia esperar, não saiu nada útil para nós. Infelizmente, o R não nos impede de fazer isso. E se

| pedirmos o elemento de número 3000 de x? Experimente.

> x[3000]

[1] NA

| Ótimo!

|============================================================= | 60%

| Mais uma vez, nada de útil, mas o R não nos impede de pedir por isso também. É importante você conhecer como

| o R funciona e você deve sempre se certificar de que o que você está pedindo está dentro dos limites do

| vetor que você está trabalhando.

...

|=============================================================== | 62%

| E se estivermos interessados em todos os elementos de x EXCETO o segundo e o décimo? Seria muito entediante

| construir um vetor contendo todos os números de 1 a 40 EXCETO 2 e 10.

...

|================================================================== | 65%

| Felizmente, o R aceita índices inteiros negativos. Enquanto x[c(2, 10)] nos dá APENAS o 2o e o 10o elementos

| de x, x[c(-2, -10)] nos dá todos os elementos de x EXCETO o 2o e 10o elementos. Digite x[c(-2, -10)] agora

| para ver isso.

> x[c(-2,-10)]

[1] -1.41187701 NA 1.44665833 NA 0.17373786 NA 0.48504045 NA

[9] -1.85117177 0.63428744 NA 0.42570388 1.58361499 NA -0.28772704 NA

[17] NA NA NA NA NA NA -1.30275810 -0.10371976

[25] -0.11134464 -0.12514674 -0.93480333 0.38004226 NA NA NA 1.32200575

[33] NA -0.06142626 NA -0.94727621 NA -1.59179723

| Você é muito bom, amig@!

|==================================================================== | 68%

| Um modo abreviado de especificar vários números negativos é colocar o sinal negativo na frente do vetor de

| números positivos. Digite x[-c(2, 10)] para obter exatamente o mesmo resultado.

> x[-c(2,10)]

[1] -1.41187701 NA 1.44665833 NA 0.17373786 NA 0.48504045 NA

[9] -1.85117177 0.63428744 NA 0.42570388 1.58361499 NA -0.28772704 NA

[17] NA NA NA NA NA NA -1.30275810 -0.10371976

[25] -0.11134464 -0.12514674 -0.93480333 0.38004226 NA NA NA 1.32200575

[33] NA -0.06142626 NA -0.94727621 NA -1.59179723

| Maravilha!

|======================================================================= | 70%

| Até agora, cobrimos três tipos de vetores de índice - lógico, inteiro positivo e inteiro negativo. O único

| tipo remanescente requer que introduzamos o conceito de elementos 'nomeados'. Já vimos isso na aula

| presencial, mas ainda não vimos isso aqui no curso interativo.

...

|========================================================================= | 73%

| Crie um vetor numérico com três elementos nomeados usando vect <- c(foo = 11, bar = 2, norf = NA).

> vect <- c(foo=11, bar=2, norf=Na)

Error: object 'Na' not found

> vect <- c(foo=11, bar=2, norf=NA)

| Maravilha!

|============================================================================ | 75%

| Quando imprimimos o vect no console, você verá que cada elemento tem um nome. Experimente.

> vect

foo bar norf

11 2 NA

| Você é muito bom, amig@!

|============================================================================== | 78%

| Nós também podemos obter os nomes de vect passando vect como um argumento para a função names().

| Experimente.

> vect.names()

Error in vect.names() : could not find function "vect.names"

> names(vect)

[1] "foo" "bar" "norf"

| Você é muito bom, amig@!

|================================================================================= | 80%

| Também, podemos criar um vetor vect2 sem nome com c(11, 2, NA). Faça isso agora.

> vect2 <- c(11,2,NA)

| Você é muito bom, amig@!

|=================================================================================== | 82%

| Então, podemos adicionar o atributo `names` ao vetor vect2 usando a função names(vect2) <- c("foo", "bar",

| "norf"). Vai lá, você consegue!"

> names(vect2) <- c('foo','bar','norf')

| Todo o trabalho duro está dando resultado!

|====================================================================================== | 85%

| Agora, vamos verificar se vect e vect2 são iguais, passando-os como argumentos para a função identical().

> identical(vect, vect2)

[1] TRUE

| Ótimo trabalho!

|======================================================================================== | 88%

| De fato, vect e vect2 são vetores nomeados idênticos.

...

|=========================================================================================== | 90%

| Agora, voltemos à questão do subconjunto de um vetor por elementos nomeados. Qual dos seguintes comandos

| você acha que nos daria o segundo elemento do vect?

1: vect["2"]

2: vect[bar]

3: vect["bar"]

Selection: 3

| Excelente!

|============================================================================================= | 92%

| Agora, execute a instrução correspondente à opção que você escolheu.

> vect['bar']

bar

2

| Bom trabalho!

|================================================================================================ | 95%

| Da mesma forma, podemos filtrar usando um vetor de nomes com vect[c("foo", "bar")]. Experimente.

> vect[c('foo', 'bar')]

foo bar

11 2

| Excelente!

|================================================================================================== | 98%

| Agora você conhece todos os quatro métodos de filtrar subconjuntos de dados de vetores. Diferentes

| abordagens são melhores em diferentes cenários . Em caso de dúvida, vai tentando!

...

|=====================================================================================================| 100%

| Gostaria de informar ao professor sobre a conclusão desta lição

1: Sim

2: Não

Selection: 1

| Qual o código da sua turma? (exemplo FIAP-01IA)

24IA

| Qual seu código de aluno?

344154

| Qual seu nome?

Diego Cohen

| O que achou deste exercício?

Ótimo!

[1] "Tentando submeter ao professor, tentativa 1 ... (max 5) ..."

[1] "saved"

#############################################################################################################

Seu resultado foi salvo!

#############################################################################################################

| Mantenha esse bom nível!