Stringr

Pedro de Brito Neto

01/05/2021

Introdução

```
#carregando o stringr
library(stringr)
```

um desafio muito grande na manipulação de dados é extrair informaçõs de caracteres. O objetivo é "lapidar" esse conjunto de caracteres para que seja posível usa-los nas análises.

- Detectar: "Tem ou não tem?"
- Localizar: "qual a posição deste elemento?"
- Extrair: selecionar a parte que nos interessa.
- Substituir: fazer mudanças e correções

Uma das principais soluções para isso é fazer é fazer o uso de funções e expressões regulares (Regular expressions). O proprio R possui algumas funções que nos ajudam a trabalhar com string's. Porém, o pacote `stringr` deixa essas (e outras) funções de uma maneira muito mais intuitiva de ser utilizadas. Vamos primeiro mostrar algumas funções báscias do pacote que podem nos auxiliar nessa manipulação de caracteres e depois falaremos sobre as expressões regulares. Por último, vamos descrever funções que costumam serem usadas com as expressões regulares Após ter um conhecimento em ambas as partes (funções e Regex), combinal-as irá tornar sua análise muito mais fácil.

str_length

A primeira função que vamos falar é a `str_length`. Uma função bem simples do `stringr` que tem como objetivo fornecer o número de caracteres de uma string, ou seja, o comprimento da string (não confunda com o `length()` de um vetor). Uma observação é que para valores `NA` ela ira preservar esse valores ao invés de retornar "2". Vamos a um exemplo

```
ex_1 <- c("tamanho", "das", "palavras", "com", "str_length", NA, "3")

str_length(ex_1)
```

```
## [1]  7  3  8  3 10 NA  1
```

str_sub

É comum você precisar pegar partes fixas de strings, como apenas a parte final das strings ou apenas os 2 primeiros caracteres. Essa função possui 5 argumentos, você pode ver de uma forma mais clara utilizando o help do r (rodar `?str_sub` no console). Os três primeiros argumentos são em geral os mais utilizados. O primeiro irá receber um vetor de caracteres, o objeto a qual irá fazer as alterações. O segunda `start` você indica onde irá começar e o argumento `end` indica até onde ir. Para este exemplo em particular vamos criar uma vetor mais organizado para ficar mais claro.

```
#vamos criar um vetor de caracteres padronizando "nome-idade"
ex_sub <- c("João-42", "Maria-29", "Pedro-20", "Tereza-50")
```

Primeiro vamos supor que a gente queira pegar apenas os nomes. Como nesse caso os caracteres possuem tamanhos diferentes para os nomes, mas a idade possui dois digitos para todos. Uma característica interessante do `str_sub()` é sua capacidade de trabalhar com índices negativos nas posições starte end. Quando usamos uma posição negativa, str_sub()conta regressivamente a partir do último caractere:

```
str_sub(ex_sub, end  = -4)
```

```
## [1] "João"   "Maria"  "Pedro"  "Tereza"
```

Caso a gente queira pegar apenas as idade podemos utilizar o argumento start para indicar onde iremos começar.

```
str_sub(ex_sub, start = -2)
```

```
## [1] "42" "29" "20" "50"
```

```
#neste caso iremos pegar do penultimo caractere até o último
```

Também é possível utilizar o `start` e o `end` em conjunto.

```
ex_sub2 <- c("__SP__", "__MG__", "__RJ__", "__ES__")
str_sub(ex_sub2, 3, 4)
```

```
## [1] "SP" "MG" "RJ" "ES"
```

```
#estamos pegando valores da posição 3 até a posição 4.
```

str_trim

Dentro de um conjunto de dados é comum que a gente encontre textos com espaços a mais ou em lugares que não deveriam ter espaços. Espaços antes e após o texto são especialmente chatos, pois pode ser difícil detectá-los Isso pode acontecer principalmente quando estamos tratando de dados provenientes de formulários que contém respostas abertas, ou seja, cada usuário pode escrever da forma que preferir. Isso pode gerar alguns problemas, e um deles é serem criadas categorias diferentes para valores que deveriam ser iguais. A função `str_trim` pode nos auxiliar na resolução desse problema removendo os espaços excedentes antes e depois da string. Os argumentos da função são 2: a string, e `side`, onde você indica o lado que deseja tirar os espaçõs, `side = c("both", "left", "right"))`.

```
extrim <- c(" vetor", " para ", "exemplificar", "o uso", "  da função  ")

str_trim(extrim, side = "left") #removendo espaçoes excedentes à esquerda
```

```
## [1] "vetor"        "para "        "exemplificar" "o uso"        "da função  "
```

```
str_trim(extrim, side = "right") #removendo espaçoes excedentes à direita
```

```
## [1] " vetor"       " para"        "exemplificar" "o uso"        "  da função"
```

```
str_trim(extrim, side = "both") #removendo espaçoes excedentes em ambos os lados
```

```
## [1] "vetor"        "para"         "exemplificar" "o uso"        "da função"
```

str_detect e str_which

```
#criando um vetor

altura <- c("1.85", "1 metro e 60 centimetros", "191 cm", "167",
            "1,58 metros", "188")
```

Caso a gente queira detectar alguma característica dentro desse vetor como números, letras ou palavras por exemplo, poderiamos fazer o uso do `str_detect`.

```
#exemplo

altura %>% str_detect("8"); #Aqui ele retorna um vetor boleano com valores "TRUE" caso o caractere cont
```

```
## [1]  TRUE FALSE FALSE FALSE  TRUE  TRUE
```

```
#Também é possível utilizar operadores logicos

altura %>% str_detect("cm|centimetro")
```

```
## [1] FALSE  TRUE  TRUE FALSE FALSE FALSE
```

Note que na segunda vez que utilizamos função, escrevemos "centimetro" e o vetor retornou um `TRUE` na posição dois, entretando ela esta no plural (centimetros). É exatamento isso que a função faz, ela não está buscando toda a palavra, ela busca padrões, qualquer caracter que tiver, nesse caso, "centimetro" ele irá retornar o valor `TRUE`. Neste mesmo conceito, existe a função `str_which` que no lugar de retornar um vetor boleano, ela retona a possição da observação com a característica buscada

```
altura %>% str_which("cm|centimetro")
```

```
## [1] 2 3
```

str_subset()

Digamos então que a gente queira não só detectar essas características, mas utiliza-las. A função `str_subset()` retorna essas strings compatíveis com a regex, sendo possível armazenar essas strings em um objeto.

```
altura %>% str_subset("cm|centimetro")
```

```
## [1] "1 metro e 60 centimetros" "191 cm"
```

str_remove e str_remove_all

A função `str_remove` é usar para remover padrões das strings, por exemplo, você pode desejar tirar alguma letra, alguma numero ou algum caractere. Essa função (mas não só ela) possui umas variante que é `str_remove_all` que basicamente funciona da mesma maneira que a primeira, porém ela vai remover TODOS os padrões dentro da mesma string. Vamos demonstrar em um exemplo.

```
altura %>% str_remove("8")
```

```
## [1] "1.5"                    "1 metro e 60 centimetros"
## [3] "191 cm"                 "167"
## [5] "1,5 metros"             "18"
```

```
altura %>% str_remove_all("8")
```

```
## [1] "1.5"                    "1 metro e 60 centimetros"
## [3] "191 cm"                 "167"
## [5] "1,5 metros"             "1"
```

Observe que ao utilizar o `str_remove`, ainda ficou um "8" no último objeto. Isso acontece basicamente porque essa função remove o padrão escolhido de todas as strings, entretando apenas o primeiro encontrado. O `str_remove_all` é uma alternativa caso queira contornar isso.

str_replace e str_repacle_all

Agora digamos que a gente não queira remover certos padrões e sim, substituir por alguma outra coisa. A função `str_remove` nos axulia nisso e assim como a `str_remove`, ela também possui a variante `str_replace_all`. Essas funções possuem 3 argumentos. Em resumo, o primeiro argumento `string` irá receber um vetor de caracteres. O segundo argumento `pattern` irá receber o padrão a ser procurado. Por fim, o terceiro argumento `replacement` recebe um vetor de caracteres de substituições e deve ter o comprimento um ou o mesmo comprimento que `string` ou `pattern` . Vamos a um exemplo simples.

```
str_replace(altura, "cm|metros|centimetros", "cm." )
```

```
## [1] "1.85"             "1 metro e 60 cm." "191 cm."          "167"
## [5] "1,58 cm."         "188"
```

```
str_replace_all(altura, c("cm" = "cm1", "centimetros" = "cm2", "metros" = ""))
```

```
## [1] "1.85"             "1 metro e 60 cm2" "191 cm1"         "167"
## [5] "1,58 "            "188"
```

```
#note que no último argumento eu subsitui "metros" por nada.
```

Expressões regulares

Agora que já conhecemos algumas das funções do pacote stringr, vamos falar um pouco sobre as expressões regulares (regex). Para trabalhar com textos de uma maneira fácil, é necessário saber um pouco de regex. Elas permitem identificar conjuntos de caracteres, palavras e outros padrões por meio de uma sintaxe concisa. O pacote divide algumas "características" das expressões regulares em subgrupos, como são muitos e normalmente seguem um padrão vamos falar apenas de algumas. Você pode ter acesso a toda parte de regex atráves do cheat sheet. Para facilitar o entendimento, após introduzir uma base das expressões regulares, vamos seguir com os exemplos praticos, exemplicando passo a passo o que está acontecendo.

Classes

Suplentes

- `x|y` ou
- `[xy]` qualquer um de

- `[^xy]` qualquer coisa menos
- `x-y` entre

Âncoras

- `^x` começo da string
- `x$` fim da string

Olhar em volta

- `x(?=y)` seguido por
- `x(?!y)` não seguido por
- `(?<=x)y` precedido por
- `(?<!x)y` não precedido por

Quantificadores

- `x?` zero ou um
- `x*` zero ou mais
- `x+` um ou mais
- `x{n}` exatamente "n"
- `x{n,}` "n" ou mais
- `x{n,m}` entre "n" e "m"

Alguns operadores interessantes

- `\\s` espaço; `\\S` não espaço
- `\\d` qualquer dígito; `\\D` não dígito
- `\\w` qualquer caractere de palavra
- `[:lower:]` letra minusculas
- `[:upper:]` letras maiusculas
- `[:punct:]` pontuação
- `[:graph:]` letras, números e pontuações
- `.` qualquer coisa (caso queira usar o "." em sua forma literal, use `\\.`)

Exemplificando classes e operadores

Mostramos acima algumas classes e operadores interesserantes e que geralmente são muito utilizados. Vamos agora mostrar em alguns exemplos como eles funcionam.

vamos criar então 2 vetores contendo as mesmas informações (3 modelos de carros), porém o segundo vetor terá espaços excedentes no meio das palavras. Imagino que poderiamos ter um conjunto de dados muito grande, "corrigir" um caractere por vez se tornaria improdutivo. Então, vamos pensar em uma solução para este exemplo usando regex.

```
carros1 <- c("Ford Mustang", "Chevrolet Camaro", "Cherry Tigo")


carros2 <- c("Ford      Mustang", "Chevrolet    Camaro", "Cherry      Tigo")

identical(carros1, carros2)
```

```
## [1] FALSE
```

Usamos a função `identical` para mostrar que o r reconhece que os vetores não são iguais. Para resolver esse problema, podemos usar a seguinte lógica: sabemos que o correto é haver apenas um espaço entre as palavras. Qualquer quantidade de espaços que for maior que um, iremos corrigir. Podemos fazer isso combinando regex com a função `str_replace_all`.

```
carros2 <- carros2 %>%
  str_replace_all("\\s{2,}", " ")

identical(carros1, carros2)
```

```
## [1] TRUE
```

Como já vimos, a função `str_replace_all` nos ajuda a fazer substiuições de caracteres. Nesse caso, a regex `\\s` signifca "espaço". O "{2,}", como já vimos, indica que queremos indificar algo se repita duas ou mais vezes. Então a função basicamente irá identificar toda vez que um espaço aparecer seguidamente duas ou mais vezes e vai substituir por apenas um espaço. Note que agora o identical retorna `true`, ou seja, os vetores são iguais. Lembrando que já falamos da `str_trim()` que remove espaços no começo e no fim dos caracteres, agora sabemos como tratar qualquer tipo de problema relacionado a espaços excedentes.

Vamos agora voltar para um vetor que já utilizamos acima. Devido a maneira com que os caracteres estão armazenadas nesse vetor, fica complicado transformar esses caracteres em informações. Ao utilizarmos expressões de regulares de uma forma bem simples, já conseguimos trabalhar com essas informações.

```
altura
```

```
## [1] "1.85"                    "1 metro e 60 centimetros"
## [3] "191 cm"                  "167"
## [5] "1,58 metros"             "188"
```

```
altura %>%
  str_remove_all("\\D")
```

```
## [1] "185" "160" "191" "167" "158" "188"
```

observe que ao utilizar o `\\D`, obtivemos apenas números. Sendo assim já conseguimos uniformizar a maneira com que as informações estão sendo exibidar. A partir daqui já podemos até obter algumas estatísticas descritivas, como a média por exemplo.

```
a <- c("não", "Não", "nao", "n", "NAO")


a %>% str_replace_all("n|N[aãA]", "Não") %>%
  str_sub(end = 3)
```

```
## [1] "Não" "Não" "Não" "Não" "Não"
```