Guide 5.1 strings in R

September 10, 2019

1 Guía 5.1: Strings en R

Computación 2, IES. Profesor: Eduardo Jorquera, eduardo.jorquera@postgrado.uv.cl

1.1 Librerías!

Usaremos los siguientes paquetes para manipular cadenas de caracteres:

```
In [1]: library(tidyverse)
        library(stringr)
        options(jupyter.rich_display=T)
```

```
 Attaching packages  tidyverse 1.2.1
 ggplot2 3.2.1      purrr   0.3.2
 tibble  2.1.3      dplyr   0.8.3
 tidyr   0.8.3      stringr 1.4.0
 readr   1.3.1      forcats 0.4.0
 Conflicts  tidyverse_conflicts()
 dplyr::filter() masks stats::filter()
 dplyr::lag()    masks stats::lag()
```

2 Subconjunto de strings

Puedes extraer parrtes de un string usando `str_sub()`. Así, la función toma un `start` y un `end` como argumentos los cuales dan la posición (inclusiva) de la cadena de caracteres:

```
In [2]: x <- c("Manzana", "Naranja", "Pera")
        str_sub(x, 1, 3)

        str_sub(x, -3, -1)
```

1. 'Man' 2. 'Nar' 3. 'Per'
1. 'ana' 2. 'nja' 3. 'era'

Note que `str_sub()` no fallará si el string es demasiado corto, simplemente retornará tanto como sea posible:

```
In [3]: str_sub("a", 1, 5)
```

'a'

También puees asignar a `str_sub()` para modificar strings:

```
In [4]: str_sub(x, 1, 1) <- str_to_lower(str_sub(x, 1, 1))
        x
```

1. 'manzana' 2. 'naranja' 3. 'pera'

3 Locales

Antes usamos `str_to_lower()` para cambiar las letras minúsculas. También puedes usar `str_to_upper()` ó `str_to_title()`. De cualquier manera, cambiar el caso es más complicado de lo que parece si trabajas con más de un idioma. A la configuración de cada idioma, es a lo que se le llama "locale". Los distintos lenguages usan diferentes reglas de gramática y escritura para usar matúsculas y minúsculas. Puedes seleccionar cualquier conjunto de reglas para usar especificando un locale:

```
In [5]: str_to_upper(c("i", ""))
        str_to_upper(c("i", ""), locale = "tr") #turco
```

1. 'I' 2. 'I'
1. '' 2. 'I'

El locale es especificado como un código de lenguage ISO 639, el cual es una abreviación de dos o tres letras. Wikipedia tiene un buen listado de códigos de lenguage (https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes). Si dejas el campo en blanco, usará el locale almacenado en tu computador, proveído por tu sistema operativo.

Otra operación importante que es afectado por el locale es el orden. La base de R tiene las funciones `order()` y `sort()`, que usan el locale del sistema. Si quieres un comportamiento más robusto para diferentes computadores, quizás estés interesado en usar `str_sort()` y `str_order()`, que toman el argumento `locale` adicionalmente:

```
In [6]: x <- c("apple", "eggplant", "banana")

        str_sort(x, locale = "en")  # inglés

        str_sort(x, locale = "haw") # hawaii
```

1. 'apple' 2. 'banana' 3. 'eggplant'
1. 'apple' 2. 'eggplant' 3. 'banana'

4 Ejercicios

- En varios códigos que no usan `stringr`, frecuentemente verás las funciones `paste` y `paste0`. Cuál es la diferencia entre las dos funciones? Qué función de stringr es equivalente? Cómo difieren en el uso de `NA`?

- Con tus palabras, describe la diferencia entre los argumentos `sep` y `collapse` en `str_c()`.
- Usa `str_length()` y `str_sub()` para extraer el caracter del medio de un string. Qué harías si el string tiene número par de caracteres?
- Qué hace `str_wrap()`? Cuándo querrías usarlo?
- Qué hace `str_trim()`? Cuál es el opuesto de esta función?
- Escribe una función que convierte (por ejemplo) un vector `c("a","b","c")` en un string `a, b, y c`, sin importar el largo del input. Piensa cuidadosamente sobre qué debería hacer si el input es un vector de tamaño 0, 1, ó 2.

```
In [ ]:
```

5 Emparejar patrones con expresiones regulares

Las expresiones regulares te permiten describir patrones en strings. Toma un poco de tiempo digerirlo en la cabeza, pero una vez que las entiendes, son sumamente útiles.

Para aprender expresiones regulares, usaremos `str_view()` y `str_view_all()`. Estas funciones toman un vector de caracteres y expresiones regulares, y muestran cómo se emparejan. Comenzaremos con expresiones regulares simples y gradualmente se tornan más y más complicadas. Una vez que seas un maestro de las expresiones regulares, aprenderás cómo aplicar varias ideas con varias funciones de `stringr`.

6 Emparejamientos simples

El emparejamiento de patrones más simples es con strings exactos:

```
In [25]: x <- c("Manzana", "Banana", "Pera")
         str_view(x, "an")
```

```
HTML widgets cannot be represented in plain text (need html)
```

No se puede ver directamente usando jupyter (depende de una configuración), pero si quieres ver el output de esto puedes usar Rstudio.

El siguiente paso para aumentar la complejidad, es usar `"."`, que hace emparejamiento con cualquier caracter (exceoti un salto de línea).

```
In [26]: str_view(x, ".a.")
```

```
HTML widgets cannot be represented in plain text (need html)
```

Pero si `"."` empareja con cualquier caracter, cómo lo emparejas con el caracter `"."` (punto)? Como vimos anteriormente, debemos usar un escape para decirle a la expresión regular que quieres hacer un emparejamiento exacto. No uses su comportamiento especial. Como los strings, las expresiones regulares (regexp) usan el slash invertido (o backslash, `"\"`, para escapar del comportamiento especial. Entonces para emparejar un punto, necesitas usar la expresión regular `"\."`.

Desafortunadamente, esto crea un problema. Usamos strings para representar expresiones regulares, y "\" también es usado para usar símbolos en strings. Entonces para crear expresiones regulares "\." necesitamos el string "\\.":

```
In [27]: # Para crear una expresión regular, necesitamos \\
         punto <- "\\."

         # Pero la expresión por sí misma sólo contiene una:
         writeLines(punto)

         # Y esto le dice a R que busque por un . explícito:
         str_view(c("abc", "a.c", "bef"), "a\\.c")
```

```
\.
```

```
HTML widgets cannot be represented in plain text (need html)
```

Si \ es usado como un caracter de escape en una expresión regular, cómo emparejas un \ de manera literal? Necesitas hacerle un escape, creando la expresión regular \\. Para crear esa expresión regular, necesitas usar un string, que también necesita escapar con \. Esto quiere decir, que para emparejar un \ de manera literal, necesitas escribir "\\\\" (cuatro backslashes para emparejar uno!):

```
In [28]: x <- "a\\b"
         writeLines(x)
         #> a\b

         str_view(x, "\\\\")
```

```
a\b
```

```
HTML widgets cannot be represented in plain text (need html)
```

7 Ejercicios:

- Explique porqué cada uno de los siguientes strings no se emparejan con un \: `"\"`, `"\\"`, `"\\\"`.
- Cómo emparejarías la secuencia `"'\`?
- Qué patrones se emparejarán con la expresión regular `\..\..\..`? Cómo lo representarías como un string?

```
In [ ]:
```

8 Anclas

Por defecto, las expresiones regulares se emparejarán con cualquier parte de un string. Es comúnmente útil *anclar* la expresión regular de tal manera que se empareje con el principio o el final de un string. Puedes usar: * `^` para emparejar el inicio del string. * `$` para emparejar el fin del string.

```
In [31]: x <- c("Manzana", "Plátano", "pera")
         str_view(x, "^p")
         str_view(x, "a$")
```

```
HTML widgets cannot be represented in plain text (need html)
```

```
HTML widgets cannot be represented in plain text (need html)
```

Para forzar a una expresión regular para que sólo empareje un string completo, ánclalo con ambos `^` y `$`:

```
In [32]: x <- c("apple pie", "apple", "apple cake")
         str_view(x, "apple")
         str_view(x, "^apple$")
```

```
HTML widgets cannot be represented in plain text (need html)
```

```
HTML widgets cannot be represented in plain text (need html)
```

También puedes emparejar el límite entre palabras con `\b`. Cuando quieres hacer una búsqueda de una palabra en específica, que conoces de principio a fin, es bastante útil. Por ejemplo, puedes buscar por `\bsum\b` para evitar que como resultado esté `summarise`, `summary`, `rowsum` etc.

9 Ejercicios

- Cómo encontrarías en un texto a la siguiente cadena de caracteres de manera literal: `"$^$"`?
- Dado el corpus de palabras comunes en `stringr::words`, creauna expresión regular que encuentre todas las palabras que:
  - empiecen con "y"
  - terminen con "x"
  - que sean de tres letras de largo (sin usar `str_length()`)
  - tienen siete letras o más. Ya que esta lista es larga, quizás quieras usar el argumento `match` de `str_view()` para mostrar sólo las palabras que se emparejan o las que no.

```
In [ ]:
```

10 Clases de caracteres y alternativas

Hay un número especial de patrones que se emparejan más que un caracter. Ya has visto el caso de ".", que se empareja con cualquier caracter que no sea un salto de línea. Así, hay otras 4 herramientas útiles: * `\d`: empareja sólo un dígito. * `\s`: empareja cualquier espacio en blanco (por ejemplo espacio, tab, salto de línea). * `[abc]`: empareja sólo a, b o c. * `[^abc]`: empareja todo excepto a,b, o c.

Recuerda, para crear una expresión regular conteniendo `\d` o `\s`, necesitas escaparla con `\` para el string, entonces tendrás que escribir `"\\d"` o `"\\s"`.

Una clase caracter conteniendo un sólo caracter es una alternativa bonita para usar backslash (escape) cuando quieres incluir un sólo metacaracter en una expresión regular. Para muchas personas, lo siguiente puede ser leído de manera más fácil:

```
In [34]: str_view(c("abc", "a.c", "a*c", "a c"), "a[.]c")
         str_view(c("abc", "a.c", "a*c", "a c"), ".[*]c")
         str_view(c("abc", "a.c", "a*c", "a c"), "a[ ]")
```

```
HTML widgets cannot be represented in plain text (need html)
```

```
HTML widgets cannot be represented in plain text (need html)
```

```
HTML widgets cannot be represented in plain text (need html)
```

Esto funciona para la mayoría (no todos) los metacaracteres: `$ . | ? * + ( ) [ {`. Desafortunadamente, pocos caracteres tienen un significado especial dentro de una clase de caracteres y deben ser manejados con backslash y escapes: `] \ ^` y `-`.

Puedes usar la alternación para tomar entre uno o más patrones de alternativas. Por ejemplo, `abc|d..f` emparejará cualquier "abc", o "deaf". Nota que la precedencia de `|` es lenta, entonces `abc|xyz` emparejará con `abc` o `xyz`, no `abcyz` o `abxyz`. Como con expresiones matemáticas, si el uso de estas precedencias te parece confusa, usa paréntesis para hacer claro lo que quieres emparejar:

```
In [35]: str_view(c("holla", "hoya"), "ho(ll|y)a")
```

```
HTML widgets cannot be represented in plain text (need html)
```

11 Ejercicios

- Crea la expresión regular para encontrar todas las palabras (en `words`) que:
  - empiecen con vocales.
  - sólo contengan consonantes (pista: piensa en no-vocales).
  - Terminen con `ed`, pero no con `eed`.
  - Terminen con `ing` o `ise`.

- Empíricamente, verifique si la letra "i" siempre está antes de la "e" excepto si la "i" está después de una "c".
- Es la "q" siempre seguida por una "u"?
- Cree una expresión regular que empareje los número telefónicos como se escriben comúnmente en Chile (+56 ...).

12 Tarea

1. Guarda el siguiente texto en un archivo y léelo importándolo a R:

```
In [ ]: Polina was not at all pleased at my questions; I could see that she was doing her best

        "It amuses me to see you grow angry," she continued. "However, inasmuch as I allow you

        "I consider that I have a perfect right to put these questions to you," was my calm ret

        Polina giggled.

        "Last time you told mewhen on the Shlangenbergthat at a word from me you would be ready

        Then she made a movement to rise. Her tone had sounded very angry. Indeed, of late her

        "May I ask you who is this Mlle. Blanche?" I inquired (since I did not wish Polina to c

        "You KNOW who she isjust Mlle. Blanche. Nothing further has transpired. Probably she wi

        "And is the General at last in love?"

        "That has nothing to do with it. Listen to me. Take these 700 florins, and go and play
        So saying, she called Nadia back to her side, and entered the Casino, where she joined

        Still, she had charged me with a commissionto win what I could at roulette. Yet all the
```

2. Cuántos párrafos tiene el texto?
3. Cuántos caracteres tiene el texto?
4. Colapse los parrafos en uno y muestrelo en pantalla (no en una lista)
5. Convierta el texto a mayúsculas y gruárdelo en un nuevo archivo "`gambler-upper.txt`".
6. Al texto original, cambie las letras 'a' y 't' por 'A' y 'T'.
7. El texto contiene la palabra "lucky"?
8. Cuántas palabras hay en el texto original? Asumiendo que las palabras son sub-strings (sub-cadenas de caracteres) separadas por un espacio o el caracter de nueva línea?
9. Cuántas veces la palabra "money" se encuentra en el texto?