Strings

Jorge Loría

Strings básicos

Ya hemos trabajado con strings dentro de dataframes, sin embargo no hemos tenido la oportunidad de trabajarlos por su cuenta. Para esto usamos el paquete:

```
library(stringr)
```

Strings básicos

Ya hemos trabajado con strings dentro de dataframes, sin embargo no hemos tenido la oportunidad de trabajarlos por su cuenta. Para esto usamos el paquete:

```
library(stringr)
```

Una parte es aprender sobre expresiones regulares en su generalidad, y otra parte es aprender sobre las funciones específicas que trabaja `stringr`.

Expresiones básicas

Las barras inclinadas: \, tienen diversos propósitos dentro de los strings. Por ejemplo, si se quiere incluir una comilla en un string antes se debe poner \. `'\''`. Además, se puede hacer `'"'`, `"'"`, `"\""`, etc...

Expresiones básicas

Las barras inclinadas: \, tienen diversos propósitos dentro de los strings. Por ejemplo, si se quiere incluir una comilla en un string antes se debe poner \. `'\''`. Además, se puede hacer `'"'`, `"'"`, `"\""`, etc...

Otra manera en que se suelen usar es para ejecutar caracteres tipo **UTF-8**, por ejemplo:

```
s1 <- '\u00a2'
```

Expresiones básicas

Las barras inclinadas: \, tienen diversos propósitos dentro de los strings. Por ejemplo, si se quiere incluir una comilla en un string antes se debe poner \. `'\''`. Además, se puede hacer `'"'`, `"'"`, `"\""`, etc...

Otra manera en que se suelen usar es para ejecutar caracteres tipo **UTF-8**, por ejemplo:

```
s1 <- '\u00a2'
```

Si se quiere ver el símbolo generado, se puede usar la función `writeLines`:

```
writeLines(s1)
```

```
## ¢
```

Dos expresiones muy frecuentes

Dos expresiones que aparecen en *toooodo* lado, porque las usamos en todo lado son: `'\t'` y `'\n'`. Que representan, un *tab* y una línea nueva, respectivamente.

Ejercicio: Use `writeLines` para ver cómo se ven estos strings: `'a\tack'`, y `'bue\no'`.

Funciones básicas

Inicialmente solo vamos a trabajar con `str_extract`, pero también se puede considerar `str_view` y ven las coincidencias en el **Viewer** de RStudio en una forma más cómoda.

Funciones básicas

Inicialmente solo vamos a trabajar con `str_extract`, pero también se puede considerar `str_view` y ven las coincidencias en el **Viewer** de RStudio en una forma más cómoda.

```
f1 <- c("piña", "naranja", "limón")
str_match(f1, "an")
```

```
##      [,1]
## [1,] NA
## [2,] "an"
## [3,] NA
```

Expresiones regulares

> *Cuando usted tiene un problema y lo ataca con una expresión regular, entonces tiene dos problemas.*

Por ejemplo, la expresión regular para calzar un correo electrónico.

Expresiones regulares

> *Cuando usted tiene un problema y lo ataca con una expresión regular, entonces tiene dos problemas.*

Por ejemplo, la expresión regular para calzar un correo electrónico.

```
f2 <- c("manzana", "banano", "pera")
str_match(f2, "n")
```

```
##      [,1]
## [1,] "n"
## [2,] "n"
## [3,] NA
```

Anclas

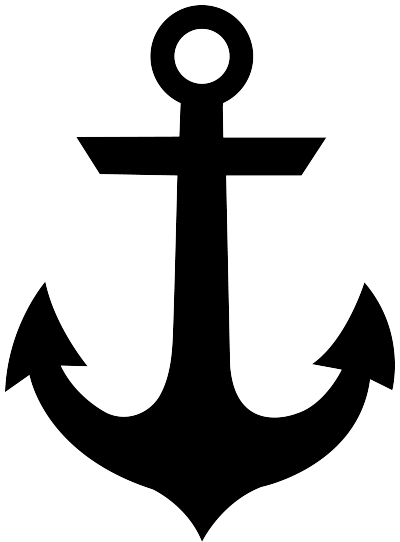

Figure 1: Otro tipo de ancla

Anclas

Hay dos anclas:

- `'^'`: lo que lo sigue es lo que va al inicio del string.
- `'$'`: lo que viene antes es lo que va al final del string.

Anclas

Hay dos anclas:

- `'^'`: lo que lo sigue es lo que va al inicio del string.
- `'$'`: lo que viene antes es lo que va al final del string.

Para recordarlo: *'Begin with power, you end with money'*

Anclas

Hay dos anclas:

- `'^'`: lo que lo sigue es lo que va al inicio del string.
- `'$'`: lo que viene antes es lo que va al final del string.

Para recordarlo: *'Begin with power, you end with money'*

Por ejemplo, si quiero obtener las palabras que comiencen con `'a'`, pongo: `'^a'` y si quiero las que terminan con `'a'`, pongo `'a$'`.

Ejercicio

La función que indica si un string contiene o no (booleano) una expresión, es `str_detect`.

Sabiendo esto, y usando `ggplot2::luv_colours$col`, identifique cuántos coloes comienzan con `'blue'`, y cuantas terminan con `'red'`.

Otras expresiones

- `'.'`: cualquier caracter.
- `'\\d'`: dígitos
- `'\\s'`: espacios en blanco
- `'[asdf]'`: cualquiera de a, s, d ó f.
- `'[^qwer]'`: cualquiera que **NO** sea q, w, e ó r.
- `asdf|uiop`: si es asdf ó uiop

Por ejemplo

Si queremos ver los colores que comienzan con 'w' o 't' y luego va una 'h', podemos hacer:

```
library(tidyverse)
luv_colours %>%
  filter(str_detect(col,'^[wt]h'))
```

```
##           L             u         v        col
## 1  9341.570 -3.370649e-12     0.000      white
## 2  7646.660  1.307466e+03 -1672.026    thistle
## 3  8724.731  1.518395e+03 -1941.769   thistle1
## 4  8255.476  1.436705e+03 -1837.301   thistle2
## 5  7327.193  1.268391e+03 -1622.057   thistle3
## 6  5372.190  9.130116e+02 -1167.587   thistle4
## 7  8438.146  1.698468e+03  3356.759      wheat
## 8  8711.263  1.759674e+03  3482.715     wheat1
## 9  8251.864  1.637730e+03  3290.972     wheat2
```

[]

Los paréntesis cuadrados tienen una peculiaridad: si se incluye un caracter especial adentro, van a dar una coincidencia aunque no se 'escape':

```
str_extract(c('au.ge','Hola.','ca.lza'),'a[.]')
```

```
## [1] NA   "a." "a."
```

[]

Los paréntesis cuadrados tienen una peculiaridad: si se incluye un caracter especial adentro, van a dar una coincidencia aunque no se 'escape':

```
str_extract(c('au.ge','Hola.','ca.lza'),'a[.]')
```

```
## [1] NA   "a." "a."
```

Repita lo anterior, pero sin usar los paréntesis cuadrados.

[]

Dentro de los paréntesis cuadrados se pueden indicar más cosas, como que sea parte del alfabeto, usando: `[A-z]`. Por ejemplo, si queremos buscar todos los colores que terminan con una letra del alfabeto, usamos:

```
luv_colours %>%
  filter(str_detect(col, "[A-z]$"))
```

Si queremos que solo coincida en letras minúsculas, cambiamos la 'A' por una 'a'.

[]

Dentro de los paréntesis cuadrados se pueden indicar más cosas, como que sea parte del alfabeto, usando: `[A-z]`. Por ejemplo, si queremos buscar todos los colores que terminan con una letra del alfabeto, usamos:

```
luv_colours %>%
  filter(str_detect(col, "[A-z]$"))
```

Si queremos que solo coincida en letras minúsculas, cambiamos la 'A' por una 'a'.Y si solo queremos las mayúsculas, cambiamos la 'z' por una 'Z'. Además, podemos preguntar por dígitos, usando `[:digit:]`.

[]

Dentro de los paréntesis cuadrados se pueden indicar más cosas, como que sea parte del alfabeto, usando: `[A-z]`. Por ejemplo, si queremos buscar todos los colores que terminan con una letra del alfabeto, usamos:

```
luv_colours %>%
  filter(str_detect(col, "[A-z]$"))
```

Si queremos que solo coincida en letras minúsculas, cambiamos la 'A' por una 'a'.Y si solo queremos las mayúsculas, cambiamos la 'z' por una 'Z'. Además, podemos preguntar por dígitos, usando `[:digit:]`. Si buscamos todos los colores que terminen con una letra y un número, hacemos:

```
luv_colours %>%
  filter(str_detect(col, "[A-z][:digit:]$"))
```

```
##             L             u             v             col
## 1   8935.2255  1.065698e+03  1.674595e+03   antiquewhite1
```

Es un *o*, simple y sencillo. El problema es que tiene prioridad sobre las demás expresiones, entonces, uno puede usar *(* paréntesis *)* para poder señalar el orden de prioridad. Por ejemplo:

```
cas <- c('cas','caza','casa','zorro')
str_detect(cas,'ca(s|z)')
```

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

Es un *o*, simple y sencillo. El problema es que tiene prioridad sobre las demás expresiones, entonces, uno puede usar *(* paréntesis *)* para poder señalar el orden de prioridad. Por ejemplo:

```
cas <- c('cas','caza','casa','zorro')
str_detect(cas,'ca(s|z)')
```

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

Si no le ponemos paréntesis:

```
str_detect(cas,'cas|z')
```

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

Ejercicios

- ¿Cuántos colores terminan con un 4?
- ¿Hay algún color que comience con un dígito?
- ¿Cuántos colores hay que sean exactamente 3 caracteres cualesquiera, y el cuarto y último sea un número?

Ejercicios

- ¿Cuántos colores terminan con un 4?
- ¿Hay algún color que comience con un dígito?
- ¿Cuántos colores hay que sean exactamente 3 caracteres cualesquiera, y el cuarto y último sea un número?

Ejercicios

- ¿Cuántos colores terminan con un 4?
- ¿Hay algún color que comience con un dígito?
- ¿Cuántos colores hay que sean exactamente 3 caracteres cualesquiera, y el cuarto y último sea un número?

98

FALSE

Ejercicios

- ¿Cuántos colores terminan con un 4?
- ¿Hay algún color que comience con un dígito?
- ¿Cuántos colores hay que sean exactamente 3 caracteres cualesquiera, y el cuarto y último sea un número?

98

FALSE

8

Repeticiones

Figure 2: Otro tipo de repeticiones

Repeticiones

Muchas veces a uno le interesa ver si un caracter apareció cierta cantidad de veces. Para esto usamos:

- `'?'`: 0 ó 1 vez
- `'+'`: 1 vez o más
- `'*'`: 0 veces o más
- `'{n}'`: exactamente *n* veces
- `'{n,}'`: al menos *n* veces
- `'{n,m}'`: entre *n* y *m* veces

Repeticiones, ejemplo:

Por ejemplo, si se quieren revisar si tienen dos o más zetas:

```
f3 <- c('pizza','grizzly','sizzler','drizzzler','caza')
str_extract(f3,pattern = 'z{2,}')
```

```
## [1] "zz"  "zz"  "zz"  "zzz" NA
```

O si se quiere revisar entre 1 y cuatro:

```
str_extract(f3,pattern = 'z{1,4}?')
```

```
## [1] "z" "z" "z" "z" "z"
```

Ejercicio

- Encuentre todos los colores que comiencen con al menos 3 consonantes.
- Encuentre todos los colores que tengan al menos dos números seguidos.

Referenciar

Muchas veces a uno le interesa revisar un patrón más de una vez, para esto se pueden 'agrupar' usando los paréntesis, y al usar `'\\1'` se hace referencia a ese valor.

```
str_view(fruit, "(..)\\1", match = TRUE)
```

Si se quieren los colores que comiencen y terminen con la misma letra, se puede hacer:

```
luv_colours %>%
  filter(str_detect(col, "^(.).*\\1$")) %>%
  select(col)
```

```
##             col
## 1 darkgoldenrod
## 2    darkorchid
## 3       darkred
## 4    lightcoral
## 5     mintcream
## 6    papayawhip
```

Más grupos

Si queremos una expresión que haga referencia a más de un grupo, podemos usar otros números.

```
luv_colours %>%
  filter(str_detect(col, "^(.).*(.).*\\2.*\\1$")) %>%
  select(col)
```

```
##             col
## 1 darkgoldenrod
## 2    darkorchid
## 3       darkred
## 4    papayawhip
```

Más grupos

Si queremos una expresión que haga referencia a más de un grupo, podemos usar otros números.

```
luv_colours %>%
  filter(str_detect(col, "^(.).*(.).*\\2.*\\1$")) %>%
  select(col)
```

```
##             col
## 1 darkgoldenrod
## 2    darkorchid
## 3       darkred
## 4    papayawhip
```

¿Qué buscamos si en la expresión anterior cambiamos el 1 por el 2 y viceversa?

Ejercicio

- Encuentre todos los colores que tengan una vocal repetida y terminen con una no-vocal.
- Encuentre todos los colores que terminen con dos números iguales.

Buscar

Buscar

Figure 3: Pero no con google

Buscar alrededor

Algunas veces a uno le interesa que coincida un valor que está próximo, pero que no sea parte del resultado. Para esto usamos los operadores 'look around':

- `'(?=algo)'`: revisa si 'algo' está justo delante
- `'(?!puede)'`: revisa si 'puede' NO está justo delante
- `'(?<=estar)'`: revisa si 'estar' está justo atrás
- `'(?<!oNO)'`: revisa si 'oNO' NO está justo atrás

Buscar alrededor

```
montos <- c('$100', '186 dolares','295 dolares', '$1580')
str_detect(montos,'\\d+\\s(?=dolares)')
```

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

Buscar alrededor

```
montos <- c('$100', '186 dolares','295 dolares', '$1580')
str_detect(montos,'\\d+\\s(?=dolares)')
```

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

Todavía no lo corran. Veánlo y piensen porqué da este resultado:

```
str_detect(montos,'(?<!\\$)\\d{3,}')
```

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

Buscar alrededor

```
montos <- c('$100', '186 dolares','295 dolares', '$1580')
str_detect(montos,'\\d+\\s(?=dolares)')
```

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

Todavía no lo corran. Veánlo y piensen porqué da este resultado:

```
str_detect(montos,'(?<!\\$)\\d{3,}')
```

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

¿Cómo podríamos modificarlo para que solo calce con los que NO tienen el símbolo de dólares?

Herramientas

Herramientas

Figure 4: Herramientas!

Herramientas

- `str_view`: para ver

Herramientas

- `str_view`: para ver
- `str_detect`: identifica si contiene la expresión

Herramientas

- `str_view`: para ver
- `str_detect`: identifica si contiene la expresión
- `str_match`: toma la parte de la expresión que coincide, lo pone en forma matricial

Herramientas

- `str_view`: para ver
- `str_detect`: identifica si contiene la expresión
- `str_match`: toma la parte de la expresión que coincide, lo pone en forma matricial
- `str_extract`: toma la parte de la expresión que coincide, lo pone en un vector

Herramientas

- `str_view`: para ver
- `str_detect`: identifica si contiene la expresión
- `str_match`: toma la parte de la expresión que coincide, lo pone en forma matricial
- `str_extract`: toma la parte de la expresión que coincide, lo pone en un vector
- `str_subset`: toma los strings que cumplen la condición, es como combinar el filter y el detect

Herramientas

- `str_view`: para ver
- `str_detect`: identifica si contiene la expresión
- `str_match`: toma la parte de la expresión que coincide, lo pone en forma matricial
- `str_extract`: toma la parte de la expresión que coincide, lo pone en un vector
- `str_subset`: toma los strings que cumplen la condición, es como combinar el filter y el detect
- `str_count`: cuenta cuantas veces en el string se dio la coincidencia

subset

Por ejemplo, si yo quiero tomar únicamente los colores que comienzan con una no-vocal, y terminan con esa misma letra, hago:

```
luv_colours$col %>%
  str_subset(pattern =  "^([^aeiou]).*\\1$")
```

```
## [1] "darkgoldenrod" "darkorchid"    "darkred"       "lightcoral"
## [5] "mintcream"     "papayawhip"
```

Quijote

Vamos a usar el primer capítulo del Quijote, que está en formato txt, y para leerlo usamos:

Quijote

Vamos a usar el primer capítulo del Quijote, que está en formato txt, y para leerlo usamos:

```
quijote <- readLines('Quijote_Capitulo_Primero.txt',
                     encoding = 'UTF-8')
```

Quijote

Vamos a usar el primer capítulo del Quijote, que está en formato txt, y para leerlo usamos:

```
quijote <- readLines('Quijote_Capitulo_Primero.txt',
                     encoding = 'UTF-8')
```

Si queremos ver cuantas veces por linea aparece 'con', usamos:

```
quijote %>% str_count(pattern = 'con')
```

```
##   [1] 0 1 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
##  [36] 0 1 0 0 0 0 0 1 0 1 1 0 0 2 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1
##  [71] 0 0 1 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 1 1 0
## [106] 0 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 1 1 0 0 1 0 0
## [141] 0 2 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

all

Es posible que se den múltiples coincidencias dentro de una misma frase que uno indique, si uno ocupara todas, puede hacer:

```
quijote %>%
  str_extract_all('de',simplify = TRUE)
```

```
##         [,1] [,2] [,3] [,4]
##    [1,] ""   ""   ""   ""
##    [2,] "de" "de" ""   ""
##    [3,] "de" ""   ""   ""
##    [4,] ""   ""   ""   ""
##    [5,] ""   ""   ""   ""
##    [6,] "de" "de" ""   ""
##    [7,] "de" "de" ""   ""
##    [8,] "de" ""   ""   ""
##    [9,] ""   ""   ""   ""
##   [10,] "de" ""   ""   ""
```

all

¿Qué pasa si no pone `simplify = TRUE`?

all

¿Qué pasa si no pone `simplify = TRUE`?

También está `str_match_all`, que hace algo similar, pero no tiene simplify, y da un formato distinto:

```
quijote %>% # Qué puede reemplazar a {0,1}?
  str_match_all('don{0,1}')
```

```
## [[1]]
##      [,1]
##
## [[2]]
##      [,1]
##
## [[3]]
##      [,1]
## [1,] "don"
##
```

sub

Una función particularmente útil, es `str_sub`, que toma la porción del string que se le indica, con base en el número de letras:

```
f3 %>%
  str_sub(start = 3,end = 5)
```

```
## [1] "zza" "izz" "zzl" "izz" "za"
```

¿Qué pasa si el `end` queda después de la última letra?

sub

Una función particularmente útil, es `str_sub`, que toma la porción del string que se le indica, con base en el número de letras:

```
f3 %>%
  str_sub(start = 3,end = 5)
```

```
## [1] "zza" "izz" "zzl" "izz" "za"
```

¿Qué pasa si el `end` queda después de la última letra?

No agrega más :)

replace

Reemplaza. Modifica en la las partes donde haya coincidencia con el patrón, y reemplaza con lo que se indique:

```
f3 %>%
  str_replace(pattern = 'z{1,2}',
              replacement = 'AAA')
```

```
## [1] "piAAAa"     "griAAAly"   "siAAAler"   "driAAAzler" "caAAAa"
```

all

Esta función también viene con una versión all, y hace el reemplazo en todos los lugares donde haya coincidencia:

```
f3 %>%
  str_replace_all(pattern = 'z{1,2}',
                  replacement = 'OEOE')
```

```
## [1] "piOEOEa"         "griOEOEly"       "siOEOEler"       "driOEOEOEOEler"
## [5] "caOEOEa"
```

str_c

Pega vectores... de forma vectorial:

```
str_c(1:13,letters,sep = '|-k-|')
```

```
##  [1] "1|-k-|a"  "2|-k-|b"  "3|-k-|c"  "4|-k-|d"  "5|-k-|e"  "6|-k-|f"
##  [7] "7|-k-|g"  "8|-k-|h"  "9|-k-|i"  "10|-k-|j" "11|-k-|k" "12|-k-|l"
## [13] "13|-k-|m" "1|-k-|n"  "2|-k-|o"  "3|-k-|p"  "4|-k-|q"  "5|-k-|r"
## [19] "6|-k-|s"  "7|-k-|t"  "8|-k-|u"  "9|-k-|v"  "10|-k-|w" "11|-k-|x"
## [25] "12|-k-|y" "13|-k-|z"
```

str_c

Pega vectores... de forma vectorial:

```
str_c(1:13,letters,sep = '|-k-|')
```

```
##  [1] "1|-k-|a"  "2|-k-|b"  "3|-k-|c"  "4|-k-|d"  "5|-k-|e"  "6|-k-|f"
##  [7] "7|-k-|g"  "8|-k-|h"  "9|-k-|i"  "10|-k-|j" "11|-k-|k" "12|-k-|l"
## [13] "13|-k-|m" "1|-k-|n"  "2|-k-|o"  "3|-k-|p"  "4|-k-|q"  "5|-k-|r"
## [19] "6|-k-|s"  "7|-k-|t"  "8|-k-|u"  "9|-k-|v"  "10|-k-|w" "11|-k-|x"
## [25] "12|-k-|y" "13|-k-|z"
```

Vamos a 'pegar' todo el capítulo en un solo string:

```
quijote1 <- str_c(quijote,collapse = ' ')
```

Más herramientas

- `str_length`: indica la cantidad de letras que tiene una palabra
  `str_length('mas')` da 3

Más herramientas

- `str_length`: indica la cantidad de letras que tiene una palabra `str_length('mas')` da 3

- `str_to_lower`: transforma los strings a letras minúsculas `str_to_lower('No eS AbuRrIdO')` da: no es aburrido

Más herramientas

- `str_length`: indica la cantidad de letras que tiene una palabra `str_length('mas')` da 3
- `str_to_lower`: transforma los strings a letras minúsculas `str_to_lower('No eS AbuRrIdO')` da: no es aburrido
- `str_to_upper`: pasa a mayúsculas `str_to_upper('Algo interesante')` da: ALGO INTERESANTE

Más herramientas

- `str_length`: indica la cantidad de letras que tiene una palabra `str_length('mas')` da 3
- `str_to_lower`: transforma los strings a letras minúsculas `str_to_lower('No eS AbuRrIdO')` da: no es aburrido
- `str_to_upper`: pasa a mayúsculas `str_to_upper('Algo interesante')` da: ALGO INTERESANTE
- `str_to_title`: pone la primera letra de cada palabra en mayúscula y las demás a minúscula, `str_to_title('eSte Es un tiTuLo')` da: Este Es Un Titulo

Más herramientas

- `str_length`: indica la cantidad de letras que tiene una palabra `str_length('mas')` da 3
- `str_to_lower`: transforma los strings a letras minúsculas `str_to_lower('No eS AbuRrIdO')` da: no es aburrido
- `str_to_upper`: pasa a mayúsculas `str_to_upper('Algo interesante')` da: ALGO INTERESANTE
- `str_to_title`: pone la primera letra de cada palabra en mayúscula y las demás a minúscula, `str_to_title('eSte Es un tiTuLo')` da: Este Es Un Titulo
- `str_replace_na`: quita los NAs y los transforma en un string que dice 'NA'

str_split

str_split

Se usa para separar strings según la expresión que se indique. En este caso, vamos a separar el resultado anterior, por medio de los espacios:

```
quijote1 %>%
  str_split('\\s')
```

Con lo que se obtienen todas las palabras del primer capítulo del Quijote :)

Ejercicios

- ¿Cuantos números aparecen en el capítulo del Quijote?

Ejercicios

- ¿Cuantos números aparecen en el capítulo del Quijote?
- ¿Cuantas veces aparece 'ella' en el capítulo del quijote? ¿Y 'el'? ¿Como puede asegurarse que no sea un 'ella'

Ejercicios

- ¿Cuantos números aparecen en el capítulo del Quijote?
- ¿Cuantas veces aparece ‘ella’ en el capítulo del quijote? ¿Y ‘el’? ¿Como puede asegurarse que no sea un ‘ella’
- ¿Como se lee la segunda línea si modificamos las vocales por por ‘u’?

Ejercicios

- ¿Cuantos números aparecen en el capítulo del Quijote?
- ¿Cuantas veces aparece 'ella' en el capítulo del quijote? ¿Y 'el'? ¿Como puede asegurarse que no sea un 'ella'
- ¿Como se lee la segunda línea si modificamos las vocales por por 'u'?
- Cree un dataframe con las palabras del capítulo del quijote. Elimine las de longitud cero y las demás páselas a minúscula. Cuente cuantas veces aparece cada palabra, e identifique cuantas de las palabras observadas comienzan con 'qu'. ¿Siempre que aparece una 'q' aparece una 'u' después? ¿Qué sucede más una 'qui' o una 'que'?