.[RegEx]*

stringr

www.rstudio.com

Functions to modify strings

Base R	**StringR (tidyverse)**
nchar	str_length
tolower	str_to_lower
toupper	str_to_upper
chartr	str_replace_all
substr	str_sub
setdiff	
setequal	
sub	str_replace
gsub	str_replace_all

Pattern matching

stringr provides pattern matching functions to **detect**, **locate**, **extract**, **match**, **replace**, and **split** strings.

Each pattern matching function has the same first two arguments, a character vector of `string`s to process and a single `pattern` to match.

Function	Description
grep	returns the index or value of the matched string
grepl	returns the Boolean value (True or False) of the matched string
regexpr	return the index of the first match
gregexpr	returns the index of all matches
regexec	is a hybrid of regexpr and gregexpr
regmatches	returns the matched string at a specified index. It is used in conjunction with regexpr and gregexpr.

Regular expressions in R

Regular expressions in R can be divided into 5 categories:

1. Metacharacters
2. Sequences
3. Quantifiers
4. Character Classes
5. POSIX character classes

Metacharacters

Metacharacters comprises a set of special operators which regex doesn't capture. Yes, regex work by its own rules. These operators are most common in every line of text you would come across. These characters include: . \ | ( ) [ ] { } $ * + ?

If any of these characters are available in a string, regex won't detect them unless they are prefixed with double backslash (\) in R.

Base R

```
dt <- c("percent%","percent")

grep(pattern = "percent\\%",x = dt,value = T)

[1] "percent%"
```

StringR

\\\\

In fact, if you find a double backslash in a string, you'll need to prefix it with another double backslash to get detected. Following is an example:

```
gsub(pattern = "\\\\",replacement = "-",x = "Barcelona\\Spain")
```

```
[1] "Barcelona-Spain"
```

Quantifies ***“Small but mighty”***

Quantifiers are mainly used to determine the length of the resulting match. Always remember, that quantifiers exercise their power on items to the immediate left of it.

Combinations of quantifiers help us match a pattern.

The nature of these quantifiers is better known in two ways:

- **Greedy Quantifiers** : The symbol .* is known as a greedy quantifier. It says that for a particular pattern to be matched, it will try to match the pattern as many times as its repetition are available.
- **Non-Greedy Quantifiers** : The symbol .? is known as a non-greedy quantifier. Being non-greedy, for a particular pattern to be matched, it will stop at the first match.

Quantifier	Description
.	It matches everything except a newline.
?	The item to its left is optional and is matched at most once.
*	The item to its left will be matched zero or more times.
+	The item to its left is matched one or more times.
{n}	The item to its left is matched exactly n times. The item must have a consecutive repetition at place. e.g. Anna
{n, }	The item to its left is matched n or more times.
{n,m}	The item to its left is matched at least n times but not more than m times.

Sequences

As the name suggests, sequences contain special characters used to describe a pattern in a given string.

Sequences	Description
\d	matches a digit character
\D	matches a non-digit character
\s	matches a space character
\S	matches a non-space character
\w	matches a word character
\W	matches a non-word character
\b	matches a word boundary
\B	matches a non-word boundary

Character classes

Character classes refer to a set of characters enclosed in a square bracket [ ]. These classes match only the characters enclosed in the bracket. These classes can also be used in conjunction with quantifiers. The use of the caret (^) symbol in character classes is interesting. It negates the expression and searches for everything except the specified pattern.

Characters	Description
[aeiou]	matches lower case vowels
[AEIOU]	matches upper case vowels
[0123456789]	matches any digit
[0-9]	same as the previous class
[a-z]	match any lower case letter
[A-Z]	match any upper case letter
[a-zA-Z0-9]	match any of the above classes
[^aeiou]	matches everything except letters
[^0-9]	matches everything except digits

POSIX Character classes

In R, these classes can be identified as enclosed within a double square bracket ([[ ]]). They work like character classes. A caret ahead of an expression negates the expression value.

POSIX Characters	Description
[[:lower:]]	matches lower case letter
[[:upper:]]	matches upper case letter
[[:alpha:]]	matches letters
[[:digit:]]	matches digits
[[:space:]]	matches space characters eg. tab, newline, vertical tab, space, etc
[[:blank:]]	matches blank characters (same as previous) such as space, tab
[[:alnum:]]	matches alphanumeric characters, e.g. AB12, ID101, etc
[[:cntrl:]]	matches control characters. Control characters are non-printable characters such as \t (tab), \n (new line), \e (escape), \f (form feed), etc
[[:punct:]]	matches punctuation characters
[[:xdigit:]]	matches hexadecimal digits (0 - 9 A - E)
[[:print:]]	matches printable characters ([[:alpha:]] [[:punct:]] and space)
[[:graph:]]	matches graphical characters. Graphical characters comprise [[:alpha:]] and [[:punct:]]

Suggestions of real problems where you need to use regex in R to explore in a future session.

Acknowledgement - material based on below sources

https://www.hackerearth.com/practice/machine-learning/advanced-techniques/regular-expressions-string-manipulation-r/tutorial/

https://cran.r-project.org/web/packages/stringr/vignettes/stringr.html