Manipulating strings with stringr

Presented by Emi Tanaka

Department of Econometrics and Business Statistics

MONASH University

1st December 2020 @ Statistical Society of Australia | Zoom
Manipulating strings

- The \texttt{stringr} package is powered by the \texttt{stringi} package which in turn uses the \texttt{ICU} C library to provide fast performance for string manipulation.

- Main functions in \texttt{stringr} prefix with \texttt{str\_} (\texttt{stringi} prefix with \texttt{stri\_}) and the \texttt{first argument is string} (or a vector of strings).

- What do you think \texttt{str\_trim} and \texttt{str\_squish} do?

```r
str_trim(c("    Apple ", "  Goji    Berry     "))
## [1] "Apple"         "Goji Berry"
str_squish(c("    Apple ", "  Goji    Berry     
## [1] "Apple"      "Goji Berry"
```


Gagolewski M. and others (2020). R package \texttt{stringi}: Character string processing facilities.
<table>
<thead>
<tr>
<th>Base R</th>
<th>stringr</th>
</tr>
</thead>
<tbody>
<tr>
<td>gregexpr(pattern, x)</td>
<td>str_locate_all(x, pattern)</td>
</tr>
<tr>
<td>grep(pattern, x, value = TRUE)</td>
<td>str_subset(x, pattern)</td>
</tr>
<tr>
<td>grep(pattern, x)</td>
<td>str_which(x, pattern)</td>
</tr>
<tr>
<td>grepl(pattern, x)</td>
<td>str_detect(x, pattern)</td>
</tr>
<tr>
<td>gsub(pattern, replacement, x)</td>
<td>str_replace_all(x, pattern, replacement)</td>
</tr>
<tr>
<td>nchar(x)</td>
<td>str_length(x)</td>
</tr>
<tr>
<td>order(x)</td>
<td>str_order(x)</td>
</tr>
<tr>
<td>regexec(pattern, x) + regmatches()</td>
<td>str_match(x, pattern)</td>
</tr>
<tr>
<td>regexpr(pattern, x) + regmatches()</td>
<td>str_extract(x, pattern)</td>
</tr>
<tr>
<td>regexpr(pattern, x)</td>
<td>str_locate(x, pattern)</td>
</tr>
</tbody>
</table>
Why use stringr?

- There are a number of considerations to ensure there is consistency in syntax and user expectation (both for input and output)
- For example, let's consider combining multiple strings into one.

**Base R**

```r
paste0("Area", "1", c("A", "B"))
## [1] "Area1A" "Area1B"
paste0("Area", "1", c("A", NA, "C"))
## [1] "Area1A" "Area1NA" "Area1C"
```

**stringr**

```r
str_c("Area", "1", c("A", "B"))
## [1] "Area1A" "Area1B"
str_c("Area", "1", c("A", NA, "C"))
## [1] "Area1A" NA "Area1C"
```

- If the Base R result is preferable then NA can be replaced with character with `str_replace_na("A", NA, "C")` first
LGA <- ozmaps::abs_lga %>% pull(NAME)

LGA[1:7]

## [1] "Broken Hill (C)" "Waroona (S)" "Toowoomba (R)" "West Arthur (S)"
## [5] "Moreton Bay (R)" "Etheridge (S)" "Cleve (DC)"

C = Cities  A = Areas  RC = Rural Cities
B = Boroughs  S = Shires  DC = District Councils
M = Municipalities  T = Towns  AC = Aboriginal Councils
RegC = Regional Councils

_extract the LGA status from the LGA names

How?
What is `\(\(.+\)\)`???

- This is a pattern expressed as regular expression or regex for short.
- Note in R, you have to add an extra `\` when `\` is included in the pattern (yes this means that you can have a lot of backslashes... just keep adding `\` until it works! Enjoy this xkcd comic.)
- From R v4.0.0 onwards, you can use raw string to eliminate all the extra `\`, e.g. `r"(\(\+.\))"` is the same as `"\\(\(.+\)\)"`
Regular expression, or *regex*, is a string of characters that define a search pattern for text.

Regular expression is... hard, but comes up often enough that it's worth learning.

```r
ozanimals <- c("koala", "kangaroo", "kookaburra", "numbat")
```

### Basic match

```r
str_detect(ozanimals, "oo")
## [1] FALSE  TRUE  TRUE FALSE

str_extract(ozanimals, "oo")
## [1] NA "oo" "oo" NA

str_match(ozanimals, "oo")
##      [,1]
## [1,]  NA
## [2,] "oo"
## [3,] "oo"
## [4,]  NA
```
Regular expressions  

== Meta-characters

- "." a wildcard to match any character except a new line

```
str_starts(c("color", "colour", "colour", "red-column"), "col...")
## [1] FALSE  TRUE  TRUE FALSE
```

- "(. | .)" a marked subexpression with alternate possibilities marked with |

```
str_replace(c("lovelove", "move", "stove", "drove"), "(l|dr|st)o", "ha")
## [1] "havelove" "move"     "have"     "have"
```

- "[ ... ]" matches a single character contained in the bracket

```
str_replace_all(c("cake", "cookie", "lamington"), "[aeiou]", ".")
```
Meta-character quantifiers

- "?" zero or one occurrence of preceding element

```r
str_extract(c("color", "colour", "colour", "red"), "colou?r")
## [1] "color"    NA "colour" NA
```

- "*" zero or more occurrences of preceding element

```r
str_extract(c("color", "colour", "colour", "red"), "colou*r")
## [1] "color"  "colouur" "colour"  NA
```

- "+" one or more occurrences of preceding element

```r
str_extract(c("color", "colour", "colour", "red"), "colou+r")
```
- "\{n\}" preceding element is matched exactly n times

```r
str_replace(c("banana", "bananana", "bana", "banananana"), "ba(na){2}", "-"
```

```r
## [1] "-" "-na" "bana" "-nana"
```

- "\{min,\}" preceding element is matched min times or more

```r
str_replace(c("banana", "bananana", "bana", "banananana"), "ba(na){2,}" , "-"
```

```r
## [1] "-" "-" "bana" "-
```

- "\{min,max\}" preceding element is matched at least min times but no more than max times

```r
str_replace(c("banana", "bananana", "bana", "banananana"), "ba(na){1,2}", "-"
```

```r
## [1] "-" "-na" "-" "-nana"
```
= Character classes

- [:alpha:] or [A-Za-z] to match alphabetic characters
- [:alnum:] or [A-Za-z0-9] to match alphanumeric characters
- [:digit:] or [0-9] or \\d to match a digit
- [^0-9] to match non-digits
- [a-c] to match a, b or c
- [A-Z] to match uppercase letters
- [a-z] to match lowercase letters
- [:space:] or [ \t\r\n\v\f] to match whitespace characters
- and more...
str_view(c("banana", "bananana", "bana", "banabananana"), "ba(na){1,2}")

• When a function in stringr ends with _all, all matches of the pattern are considered
• The one *without* _all only considers the first match

str_view_all(c("banana", "bananana", "bana", "banabananana"), "ba(na){1,2}")
Where the same Local Government Area name appears in different States or Territories, the State or Territory abbreviation appears in parenthesis after the name. Local Government Area names are therefore unique.

-Australian Bureau of Statistics
str_extract(LGA, "\(([^)]+)\)") %>%
  # remove the brackets
  str_replace_all("\[(\)]", "") %>%
  table()

##
##    A   AC    B    C   DC    M    R   RC RegC    S    T
##  100    2    1  125   41   27   39    7    1  183   12

- "[ ]" for single character match
- We want to match ( and ) but these are meta-characters
- So we need to escape it to have it as a literal: \( and \)
- But we must escape the escape character... so it's actually \\
  \( \)
str_extract(LGA, r"([\^[]+\\)"") %>%
  # remove the brackets
  str_replace_all(r"([\((\)])"", "") %>%
table()

# .
## A   AC   B   C   DC   M   R   RC RegC   S   T
## 100 2  1  125  41  27  39  7   1  183  12

- If using R v4.0.0 onwards, you can use the raw string version instead
Regex still difficult? Try RStudio addin `regexplain`
If you still find it difficult, you may find an expressive piping approach to be easier for you:

```r
library(RVerbalExpressions)

## Warning: package 'RVerbalExpressions' was built under R version 4.0.2

rx_start_of_line() %>%
  rx_find('http') %>%
  rx_maybe('s') %>%
  rx_find('://') %>%
  rx_maybe('www.') %>%
  rx_anything_but(' ') %>%
  rx_end_of_line()

## [1] "^(http(s)?(\://)(www\.)?([^ \]*)$"
```

Tyler Littlefield (2019). RVerbalExpressions: Create Regular Expressions Easily. R package version 0.1.0
animal <- c("koala", "kangaroo", "numbat")
quality <- c("cuddly", "cool", "cute")
paste0("I love ", animal, ", it's so ", quality, ")

## [1] "I love koala, it's so cuddly!"  "I love kangaroo, it's so cool!"
## [3] "I love numbat, it's so cute!"

It works, but we have to break out of the string constantly to refer to variables in the environment, but str_glue saves you the trouble!

str_glue("I love {animal}, it's so {quality}!")

## I love koala, it's so cuddly!
## I love kangaroo, it's so cool!
## I love numbat, it's so cute!

str_glue is just a wrapper for glue from the glue package
```r
df <- data.frame(animal = animal,
                 quality = quality)

glue::glue_data(df, "I love {animal}, it's so {quality}!"

## I love koala, it's so cuddly!
## I love kangaroo, it's so cool!
## I love numbat, it's so cute!

stringr::str_glue_data(df, "I love {animal}, it's so {quality}!"

## I love koala, it's so cuddly!
## I love kangaroo, it's so cool!
## I love numbat, it's so cute!
```
These slides are licensed under

![Creative Commons License](cc.png)