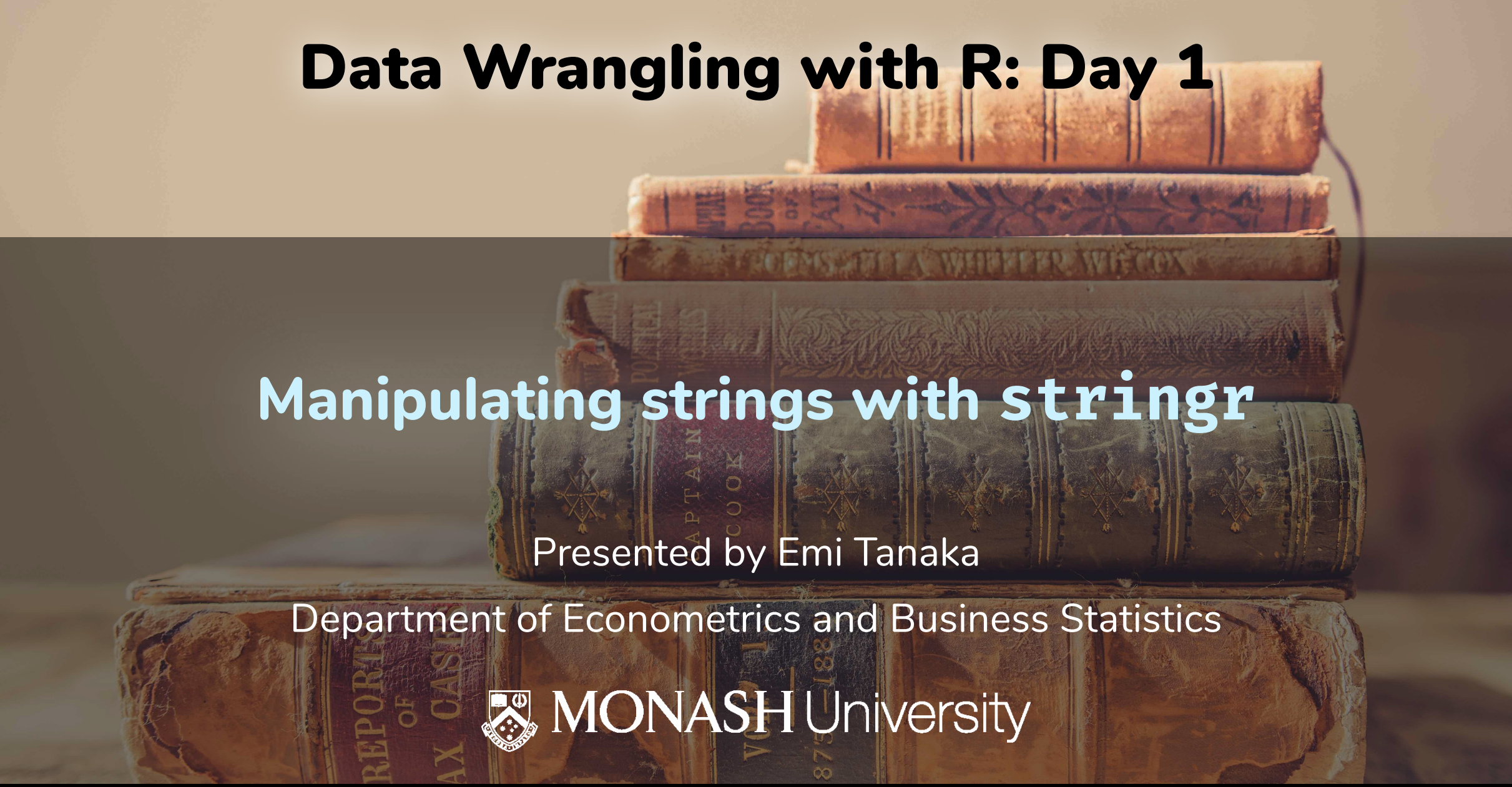
Data Wrangling with R: Day 1



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 `stringr` package is powered by the `stringi` package which in turn uses the ICU C library to provide fast peformance for string manipulation
- Main functions in `stringr` **prefix with `str_`** (`stringi` prefix with `stri_`) and the **first argument is string** (or a vector of strings)
- What do you think `str_trim` and `str_squish` do?

```
str_trim(c("    Apple ", "  Goji    Berry     "))
```

```
## [1] "Apple"         "Goji    Berry"
```

```
str_squish(c("    Apple ", "  Goji    Berry     "))
```

```
## [1] "Apple"      "Goji Berry"
```

Hadley Wickham (2019). stringr: Simple, Consistent Wrappers for Common String Operations. R package version 1.4.0.

Gagolewski M. and others (2020). R package stringi: Character string processing facilities.

Base R and stringr

Base R	stringr
gregexpr(pattern, x)	str_locate_all(x, pattern)
grep(pattern, x, value = TRUE)	str_subset(x, pattern)
grep(pattern, x)	str_which(x, pattern)
grepl(pattern, x)	str_detect(x, pattern)
gsub(pattern, replacement, x)	str_replace_all(x, pattern, replacement)
nchar(x)	str_length(x)
order(x)	str_order(x)
regexec(pattern, x) + regmatches()	str_match(x, pattern)
regexpr(pattern, x) + regmatches()	str_extract(x, pattern)
regexpr(pattern, x)	str_locate(x, pattern)

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

```
paste0("Area", "1", c("A", "B"))

## [1] "Area1A" "Area1B"

paste0("Area", "1", c("A", NA, "C"))

## [1] "Area1A"  "Area1NA" "Area1C"
```

`stringr`

```
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

Case study 🔍 Aussie Local Government Area

```
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

B = Boroughs

M = Municipalities

RegC = Regional Councils

A = Areas

S = Shires

T = Towns

RC = Rural Cities

DC = District Councils

AC = Aboriginal Councils

🎯 **Extract the LGA status from the LGA names**

How?

Extracting the string

```
str_extract(LGA, "\\(.+\\)")
```

```
##   [1] "(C)"       "(S)"       "(R)"       "(S)"       "(R)"
##   [6] "(S)"       "(DC)"      "(R)"       "(DC)"      "(C)"
##  [11] "(DC)"      "(S)"       "(S)"       "(S)"       "(DC)"
##  [16] "(A)"       "(C)"       "(A)"       "(T)"       "(RC)"
##  [21] "(A)"       "(S)"       "(S)"       "(S)"       "(C)"
##  [26] "(DC)"
##  [31] "(S)"
##  [36] "(R)"
##  [41] "(S)"
##  [46] "(AC)"
##  [51] "(A)"
##  [56] "(S)"
##  [61] "(C)"
##  [66] "(C)"      "(S)"       "(DC)"      "(DC)"      "(S)"
```

- 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 elimiate all the extra \, e.g. `r"(\(.+\))" is the same as "\\(.+\\)"`

Regular expressions Part 1

- **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

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

= Basic match

```
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 Part 2

Meta-characters

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

```
str_starts(c("color", "colouur", "colour", "red-column"), "col...")
```

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

- `"(.|.)"` a marked subexpression with alternate possibilites 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]", "_")
```

Regular expressions Part 3

Meta-character quantifiers

- "?" zero or one occurence of preceding element

```
str_extract(c("color", "colouur", "colour", "red"), "colou?r")
```

```
## [1] "color"  NA       "colour" NA
```

- "*" zero or more occurence of preceding element

```
str_extract(c("color", "colouur", "colour", "red"), "colou*r")
```

```
## [1] "color"   "colouur" "colour"  NA
```

- "+" one or more occurence of preceding element

```
str_extract(c("color", "colouur", "colour", "red"), "colou+r")
```

Regular expressions Part 4

- `"{n}"` preceding element is matched exactly `n` times

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

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

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

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

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

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

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

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

Regular expressions Part 5

= 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...

View matches with regular expressions

```
str_view(c("banana", "bananana", "bana", "banabanana"), "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

```
banana
bananana
bana
banabanana
```

```
str_view_all(c("banana", "bananana", "bana", "banabanana"), "ba(na){1,2}")
```

```
banana
bananana
bana
```

Back to Extracting the string

```
str_extract(LGA, "\\(.+\\)") %>%
  table()

## .
##        (A)       (AC)        (B)        (C)  (C) (NSW)   (C) (SA) (C) (Vic.)
##        100          2          1        120          2          1          2
##       (DC)  (DC) (SA)        (M) (M) (Tas.)        (R)  (R) (Qld)        (RC)
##         40          1         23          4         38          1          7
##     (RegC)        (S)  (S) (Qld)        (T)
##          1        182          1         12
```

> “ 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

Retry Extracting the string

```
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 \\(\\)

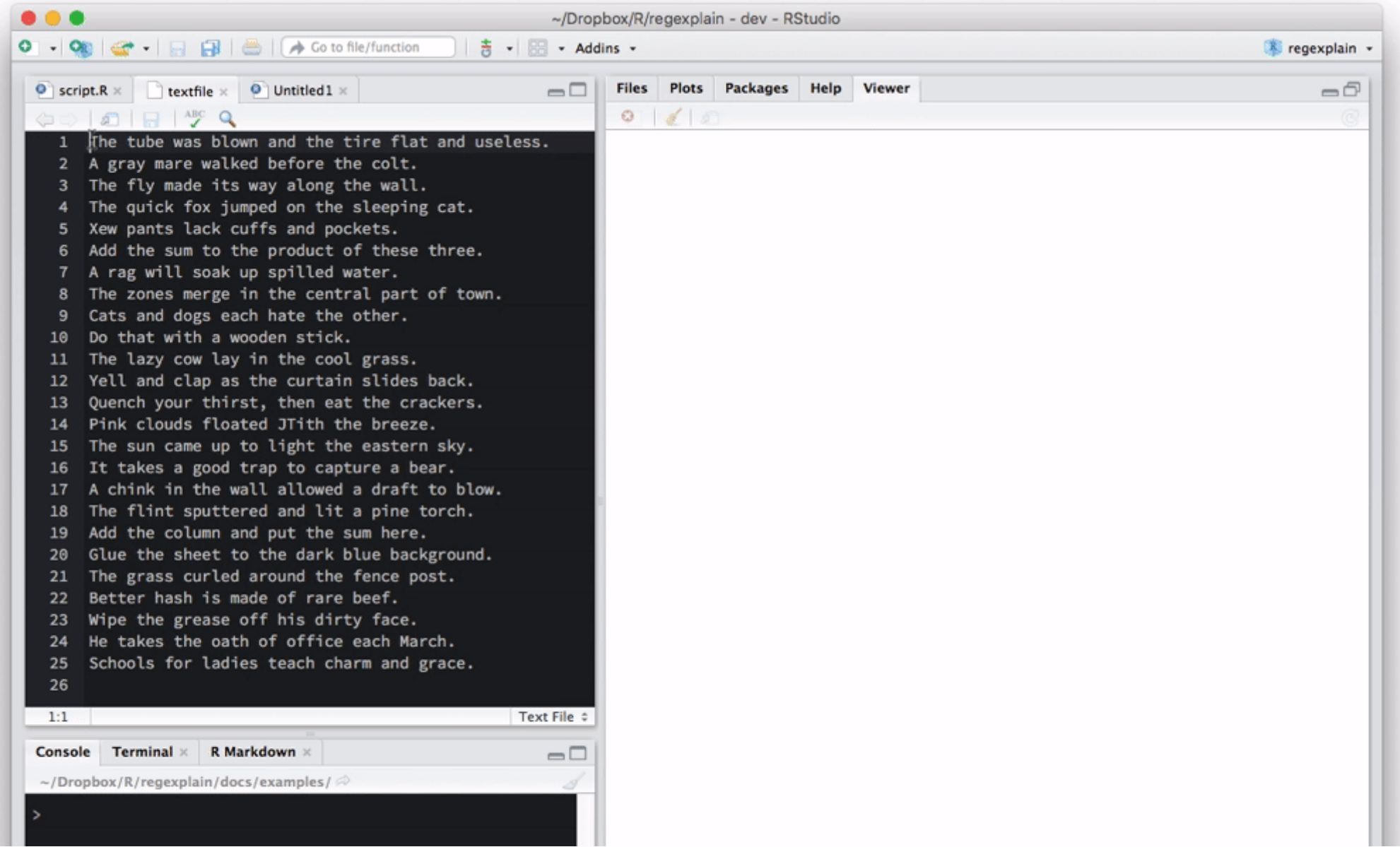
R v4.0.0 Extracting the string

```
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`



RVerbalExpressions

- If you still find it difficult, you may find an expressive piping approach to be easier for you:

```
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\\.)?([^ ]*)$"
```

stringr::str_glue or glue::glue

```
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

stringr::str_glue_data or glue::glue_data

```
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!
```

Session Information

```
devtools::session_info()
```

```
## ─ Session info ─────────────────────────────────────────────────────────────
##  setting  value
##  version  R version 4.0.1 (2020-06-06)
##  os       macOS Catalina 10.15.7
##  system   x86_64, darwin17.0
##  ui       X11
##  language (EN)
##  collate  en_AU.UTF-8
##  ctype    en_AU.UTF-8
##  tz       Australia/Melbourne
##  date     2020-11-26
##
## ─ Packages ─────────────────────────────────────────────────────────────────
##  package             * version    date       lib
##  anicon                0.1.0      2020-06-21 [1]
##  assertthat            0.2.1      2019-03-21 [2]
```