Programming for Data Analytics

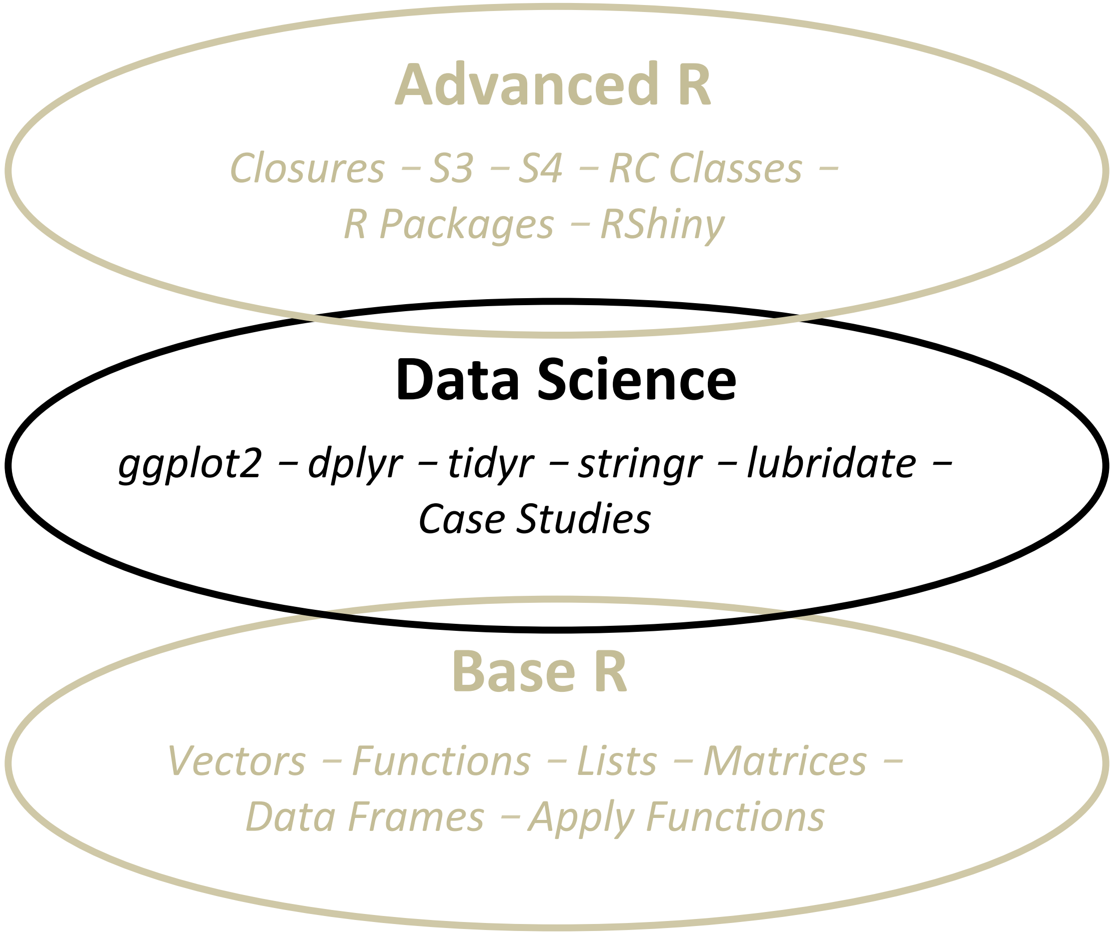
Lecture 8: stringr

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

https://twitter.com/_jimduggan

Lecture Overview

- stringr basics
- Regular Expressions overview
  - str_view()
  - Anchors
  - Repetition
- Tools
  - str_detect()
  - str_count()
  - str_extract()
  - str_replace()
  - str_split()



(1) Stringr Basics

- Focus on the stringr package, which is not part of the tidyverse.
- All stringr functions start with **str_**
- Strings can be created with single or double quotes, recommend to use “”
- Multiple strings can be stored in a character vector

```
> s1 <- "This is a string"
> s1
[1] "This is a string"
>
> s2 <- 'This is a string too'
> s2
[1] "This is a string too"
>
> s3 <- c("One","Two three")
> s3
[1] "One"       "Two three"
```

String Length: str_length()

- Returns the number of characters in a string
- Is vectorised

```
> s1
[1] "This is a string"
>
> str_length(s1)
[1] 16
>
> s3 <- c("One","Two three")
>
> str_length(s3)
[1] 3 9
```

Combining Strings: str_c()

- To combine two or more strings, use str_c()
- Use the sep argument to control how they are separated
- str_c() is vectorised, and automatically recycles shorter vectors to the same length as the longest

```
> str_c("x","y")
[1] "xy"
>
> str_c("x","y","z")
[1] "xyz"
>
> str_c("x","y","z",sep="-")
[1] "x-y-z"
>
> str_c(c("x","y","c"),"ab")
[1] "xab" "yab" "cab"
>
> str_c(c("x","y","c"),"ab",
+        collapse = "")
[1] "xabyabcab"
```

Subsetting Strings: str_sub()

- As well as the string, this function takes the start and end arguments that give the (inclusive) position of the string.

```
> x <- c("Apple","Banana","Pear")
> str_sub(x,1,3)
[1] "App" "Ban" "Pea"
>
> str_sub(x, -3, -1)
[1] "ple" "ana" "ear"
```

(2) Matching Patterns with Regular Expressions

- Regexprs are very terse language that allows you to describe patterns in strings
- Very powerful features to process strings
- To learn regular expressions:
  - use str_view() and str_view_all()
  - These take a character vector, and a regular expression and show the match

Basic matches

- The simplest patterns match exact strings

```
x <- c("Apple","Banana","Pear")
str_view(x,"an")
```

```
Apple
Banana
Pear
```

- The next step up in complexity is ., which matches any character (except a newline)

```
x <- c("Apple","Banana","Pear")
str_view(x,".a.")
```

```
Apple
Banana
Pear
```

To match ., we need \\.

```r
y <- c("abc","a.c","bef")
str_view(y,"a\\.c")
```

```
abc
a.c
bef
```

Anchors

- By default, a regular expression will match any part of a string.
- It can be useful to *anchor* the expression so that it matches from the start or end of the string
  - ^ match start
  - $ match end

```
x <- c("apple pie","apple","apple cake")
str_view(x,"^apple")
str_view(x,"^apple$")
```

```
apple pie
apple
apple cake
```

```
apple pie
apple
apple cake
```

Challenge 8.1

- Take a random sample (50, seed=99) from the corpus of common words in stringr::words and create regular expressions that find all words that:
  - Start with “w”
  - End with “o”
  - Are exactly three characters long
- The match argument can be used to str_view() to show the matching/non-matching words.

Other useful matching tools

- \d – matches any digit
- \s – matches any whitespace
- [abc] – matches a, b or c
- [^abc] – matches anything except a, b or c.

```
str_view(y,"\\d")
```

```
abc
abc1
```

```
str_view(c("grey","gray"),"gr(e|a)y")
```

```
grey
gray
```

Challenge 8.2

- With the sample sample data set, create regular expressions to find words that:
  - Start with a vowel
  - Start with two successive consonants
  - End with ing

Repetition

- Next, we can control how many times a pattern matches
  - ? → 0 or 1
  - + → 1 or more
  - * → 0 or more

```r
x <- "AACABCDDD"

str_view(x,"CD?")
str_view(x,"CD+")
str_view(x,"CD*")
```

AACABCDDD

AACABCDDD

AACABCDDD

Specifying the number of matches

- The number of matches can be specified precisely (default match is "greedy" – will match the longest string possible)
  - {n}: exactly n
  - {n,}: n or more
  - {,m}: at most m
  - {n,m}: between n and m

```r
x <- "AACABCDDD"

str_view(x,"A{2}")
str_view(x,"D{1,}")
str_view(x,"D{2,3}")
str_view(x,"D{2,3}?")
```

AACABCDDD

AACABCDDD

AACABCDDD

AACABCDDD

Challenge 8.3

- Create regular expressions that find all words that:
  - Start with two consonants
  - Have three vowels in a row
- Make use of the sample data set.

(3) Regex Tools in R

- Given the basics just covered, these can now be applied to real problems. The **stringr** package has functions to:
  - Determine which strings match a pattern
  - Find the positions of the matches
  - Extract the content of matches
  - Replace matches with new values
  - Split a string based on a match

Detect Matches: str_detect()

- Determines if a character vector matches a pattern, and returns a logical vector of the same length
- Use of sum() useful… count number of matches.

```
> x <- c("apple", "banana","pear")
>
> str_detect(x,"e")
[1]  TRUE FALSE  TRUE
>
> sum(str_detect(words,"[aeiou]$"))
[1] 271
>
> mean(str_detect(words,"[aeiou]$"))
[1] 0.2765306
```

Using with dplyr and str_count()

```
df <- tibble(word=words,i=seq_along(words))

df %>%
  mutate(
    vowels = str_count(word,"[aeiou]"),
    consonants = str_count(word,"[^aeiou]")
  )
```

```
# A tibble: 980 x 4
        word     i vowels consonants
       <chr> <int>  <int>      <int>
 1         a     1      1          0
 2      able     2      2          2
```

Challenge 8.4

- Use str_detect() to find all words that start with a vowel and end in a consonant.
- What word in the corpus has the highest number of vowels?

Exact Matches: str_extract()

- To extract the actual text of a match, use str_extract()
- Using the Harvard Sentences data set, designed to test VOIP systems

```
> head(sentences)
[1] "The birch canoe slid on the smooth planks."
[2] "Glue the sheet to the dark blue background."
[3] "It's easy to tell the depth of a well."
[4] "These days a chicken leg is a rare dish."
[5] "Rice is often served in round bowls."
[6] "The juice of lemons makes fine punch."
```

Example

- The task is to locate all sentences that contain a colour.
- str_subset() useful as it filters the data set based on a regular expression

```
> colours <- c(
+   "red","orange","yellow","green","blue","purple"
+ )
>
> col_match <- str_c(colours,collapse = "|")
>
> col_match
[1] "red|orange|yellow|green|blue|purple"
```

Note: str_extract() returns 1st match

```
> has_colour <- str_subset(sentences, col_match)
> matches <- str_extract(has_colour,col_match)
> head(matches)
[1] "blue" "blue" "red"  "red"  "red"  "blue"
>
> more <- sentences[str_count(sentences,col_match) > 1]
> more
[1] "It is hard to erase blue or red ink."
[2] "The green light in the brown box flickered."
[3] "The sky in the west is tinged with orange red."
>
> str_extract(more,col_match)
[1] "blue"   "green"  "orange"
```

```
> str_extract_all(more,col_match)
[[1]]
[1] "blue" "red"

[[2]]
[1] "green" "red"

[[3]]
[1] "orange" "red"

>
> str_extract_all(more,col_match,simplify = T)
     [,1]     [,2]
[1,] "blue"   "red"
[2,] "green"  "red"
[3,] "orange" "red"
```

Grouped Matches

- Parentheses can be used to clarify precedence
- They can also be used to extract parts of a complex match
- For example, if we need to extract nouns from sentences
- A heuristic:
  - *Any word that come after "a" or "the"*
  - *A word is defined as a sequence of at least one character that isn't a space*

Example: Two groups

```
noun <- "(a|the) ([^ ]+)"

has_noun <- sentences %>%
  str_subset(noun) %>%
  head(10)
```

```
> has_noun
 [1] "The birch canoe slid on the smooth planks."
 [2] "Glue the sheet to the dark blue background."
 [3] "It's easy to tell the depth of a well."
 [4] "These days a chicken leg is a rare dish."
 [5] "The box was thrown beside the parked truck."
```

Using str_extract()

```
> has_noun
 [1] "The birch canoe slid on the smooth planks."
 [2] "Glue the sheet to the dark blue background."
 [3] "It's easy to tell the depth of a well."
 [4] "These days a chicken leg is a rare dish."
 [5] "The box was thrown beside the parked truck."

> has_noun %>%
+   str_extract(noun)
 [1] "the smooth" "the sheet"  "the depth"  "a chicken"
 [5] "the parked" "the sun"    "the huge"   "a ball"
 [9] "the woman"  "a helps"
```

Using str_match()

- Column 1 is the complete match
- Successive columns are for each group.
- Suggested nouns are in group 3
- *All matches not included*

```
> has_noun %>%
+   str_match(noun)
      [,1]          [,2]  [,3]
 [1,] "the smooth" "the" "smooth"
 [2,] "the sheet"  "the" "sheet"
 [3,] "the depth"  "the" "depth"
 [4,] "a chicken"  "a"   "chicken"
 [5,] "the parked" "the" "parked"
 [6,] "the sun"    "the" "sun"
 [7,] "the huge"   "the" "huge"
 [8,] "the ball"   "the" "ball"
 [9,] "the woman"  "the" "woman"
[10,] "a helps"    "a"   "helps"
```

Using with dplyr & extract()

```
tibble(sentence=sentences) %>%
  tidyr::extract(
    sentence,c("article","noun"),"(a|the) ([^ ]+)",
    remove=F
  )
```

```
# A tibble: 720 x 3
                                    sentence article    noun
 *                                     <chr>   <chr>   <chr>
 1  The birch canoe slid on the smooth planks.     the  smooth
 2 Glue the sheet to the dark blue background.     the   sheet
 3      It's easy to tell the depth of a well.     the   depth
 4    These days a chicken leg is a rare dish.       a chicken
 5        Rice is often served in round bowls.    <NA>    <NA>
```

Replacing Matches

- str_replace() and str_replace_all() allow the replacement of matches with new strings.

```
> x <- c("apple","pear","banana")
>
> x
[1] "apple"  "pear"   "banana"
>
> str_replace(x,"[aeiou]","_")
[1] "_pple"  "p_ar"   "b_nana"
>
> str_replace_all(x,"[aeiou]","_")
[1] "_ppl_"  "p__r"   "b_n_n_"
```

Perform multiple replacements

```
> x <- c("1 house","2 cars","3 people")
>
> str_replace_all(x,c("1"="one","2"="two","3"="three"))
[1] "one house"    "two cars"     "three people"
```

Flip the order of second and first words (note groupings)

```
> sentences %>%
+   str_replace("([^ ]+) ([^ ]+)","\\2 \\1") %>%
+   head(5)
[1] "birch The canoe slid on the smooth planks."
[2] "the Glue sheet to the dark blue background."
[3] "easy It's to tell the depth of a well."
[4] "days These a chicken leg is a rare dish."
[5] "is Rice often served in round bowls."
```

Challenge 8.5

- Switch the first and last words in the sentences

```
> s
[1] "The birch canoe slid on the smooth planks."
[2] "Glue the sheet to the dark blue background."
[3] "It's easy to tell the depth of a well."
[4] "These days a chicken leg is a rare dish."
[5] "Rice is often served in round bowls."
```

```
> ans
[1] "planks birch canoe slid on the smooth The."
[2] "background the sheet to the dark blue Glue."
[3] "well easy to tell the depth of a It's."
[4] "dish days a chicken leg is a rare These."
[5] "bowls is often served in round Rice."
```

Splitting Strings – str_split()

- Use str_split() to split a string into pieces
- With simplify = TRUE, a matrix is returned

```
> x <- "This is a sentence"
>
> x
[1] "This is a sentence"
>
> str_split(x, " ")
[[1]]
[1] "This"     "is"       "a"        "sentence"
```

Other types of pattern

- When you use a pattern that's a string, it is automatically wrapped into a call to regex()
- regex() can use other arguments to control the details of the match

```
> x <- c("APPLE","aPPle","Apple")
>
> x
[1] "APPLE" "aPPle" "Apple"
>
> x[str_detect(x,regex("apple",ignore_case=T))]
[1] "APPLE" "aPPle" "Apple"
```

comments = TRUE

- Allows you to use comments and white space to make complex regular expressions more understandable

```
> phone <- regex("
+   \\(?      # optional opening parens
+   (\\d{3}) # area code
+   [)- ]?   # optional closing parens, dash, or space
+   (\\d{3}) # another three numbers
+   [ -]?    # optional space or dash
+   (\\d{4}) # four more numbers
+   ", comments=TRUE)
>
> str_match("514-791-8111",phone)
     [,1]           [,2]  [,3]  [,4]
[1,] "514-791-8111" "514" "791" "8111"
```

Other Uses of Regular Expressions

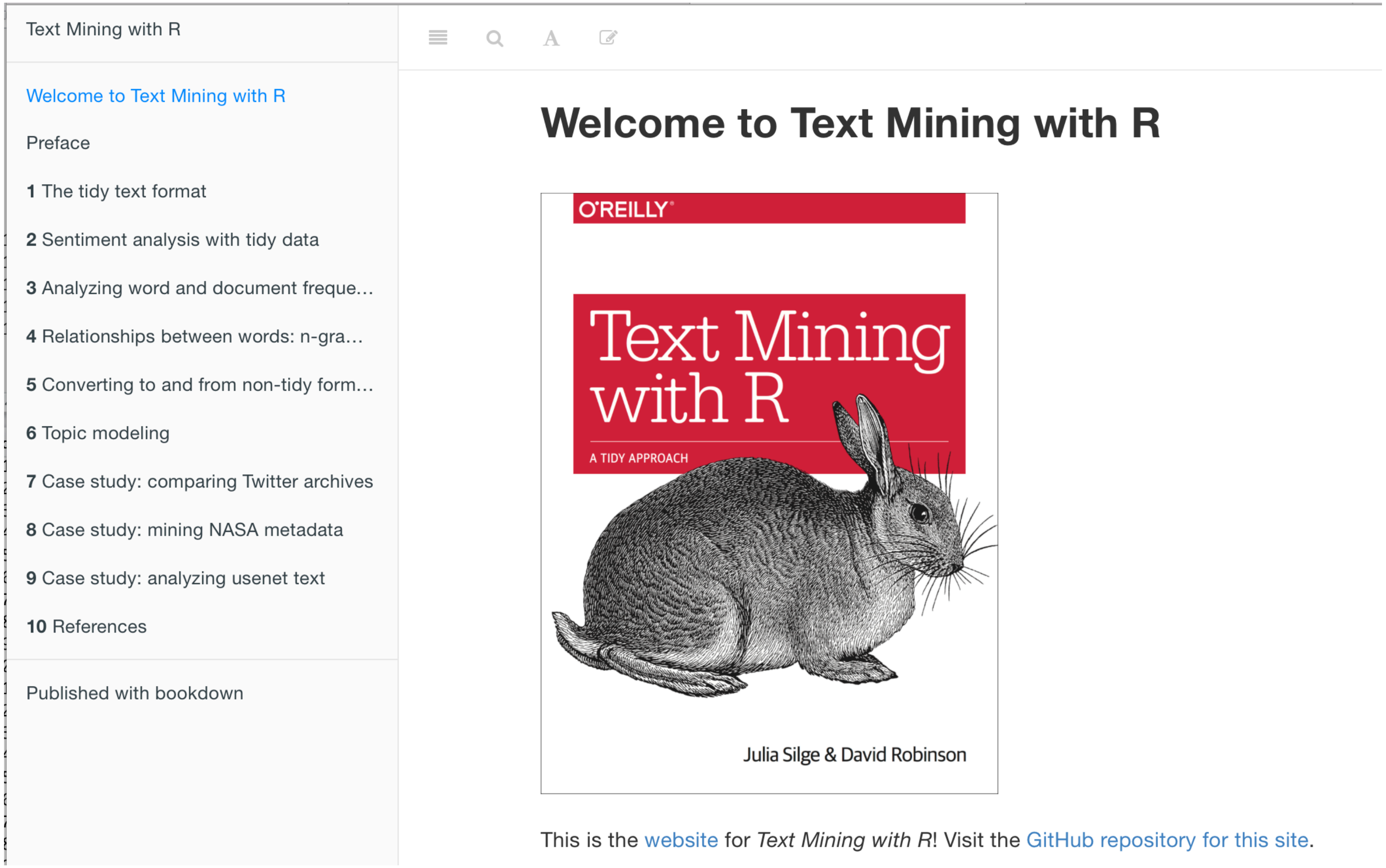
- apropos() searches all objects available from the global environment.

```
> apropos("str_")
 [1] "str_c"           "str_conv"        "str_count"
 [4] "str_detect"      "str_dup"         "str_extract"
 [7] "str_extract_all" "str_interp"      "str_join"
[10] "str_length"      "str_locate"      "str_locate_all"
[13] "str_match"       "str_match_all"   "str_order"
[16] "str_pad"         "str_replace"     "str_replace_all"
[19] "str_replace_na"  "str_sort"        "str_split"
[22] "str_split_fixed" "str_sub"         "str_sub<-"
[25] "str_subset"      "str_to_lower"    "str_to_title"
[28] "str_to_upper"    "str_trim"        "str_trunc"
[31] "str_view"        "str_view_all"    "str_wrap"
```

stringi Package

- stringr is built on top of stringi package
- stringr exposes a minimal set of functions
- stringi is designed to be comprehensive, and contains 234 functions (as compared to 42 for stringr)
- Exercises, find the stringi functions that
  - Count the number of words
  - Find duplicated strings
  - Generate random text

http://tidytextmining.com/index.html



Summary

- **stringr** package very useful in R
- Regular expressions key to string manipulations
- Rich set of functions to support string processing