

# R for bioinformatics, Strings and regular expression

HUST Bioinformatics course series

Wei-Hua Chen (CC BY-NC 4.0)

27 September, 2022

# section 1: TOC

# 前情提要

Talks so far:

- ① introduction to R
- ② R language basics, part 1
- ③ R language basics, part 2
- ④ R language basics, part 3, factors
- ⑤ data wrangler, part 1
- ⑥ data wrangler, part 2

# packages we have touched so far

## 1 tidyverse

- dplyr
- tidyr
- ggplot2

## 2 readr

## 3 tibble

## 4 forcats ...

# 本次提要

stringr

## ① basics

- length
- uppercase, lowercase
- unite, separate
- string comparisons, sub string

## ② regular expression

## section 2: contents

# get ready for the class

```
library(tidyverse);
```

```
## -- Attaching packages ----- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr  1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.2      v forcats 0.5.2
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(stringr);
```

## 其它著名的 packages

- stringi

## section 2: simple manipulations ...



# string

```
string1 <- "This is a string";
string2 <- 'If I want to include a "quote" inside a string, I use single quotes';
```

```
( string3 <- "a multiline
string" );
```

```
## [1] "a multiline \nstring"
```

```
## 注意与上面的区别
writeLines( string3 );
```

```
## a multiline
## string
```

# quotes & other special characters

```
( double_quote <- "\"" );
```

```
## [1] "\""
```

```
( single_quote <- "'" );
```

```
## [1] "'"
```

```
( x <- "\u00b5" )
```

```
## [1] "µ"
```

```
## 注意不同!!!!
```

```
( y <- "\\\" )
```

```
## [1] "\\\""
```

```
writeLines( y );
```

```
## \
```

# string length

```
## 系统自带  
nchar( c("a", "R for data science", NA) );
```

```
## [1]  1 18 NA
```

```
## stringr  
str_length(c("a", "R for data science", NA));
```

```
## [1]  1 18 NA
```

# string combine

```
## 系统自带
paste( "a", "b", "c", sep = "" );
```

```
## [1] "abc"
```

```
## stringr
str_c( "a", "b", "c" );
```

```
## [1] "abc"
```

```
paste( c( "a", "b", "c" ), 2, sep = "" );
```

```
## [1] "a2" "b2" "c2"
```

```
str_c( c( "a", "b", "c" ), 2 );
```

```
## [1] "a2" "b2" "c2"
```

# string comparison

```
## direct comparison ; 可用于排序 ...
"A" > "abc";
```

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

```
##
library(pracma);
```

```
##
## Attaching package: 'pracma'
```

```
## The following object is masked from 'package:purrr':
##
##      cross
```

```
strcmp( "chen", "chenweihua" );
```

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

```
strcmpi( "chen", "CHEN" );
```

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

# other simple functions

```
toupper( letters[1:10] );
```

```
## [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

```
tolower( LETTERS[1:5] );
```

```
## [1] "a" "b" "c" "d" "e"
```

```
library(stringi);  
stri_reverse( "ABC" );
```

```
## [1] "CBA"
```

## tricks

- stringi package 里的 function 都以 stri\_ 开头
- strinr 则以 str\_ 开头

## section 3: regular expression basics

# what is regular expression (正则表达式) ?

在介绍更多 string manipulation 函数之前，先介绍正则表达式：  
a sequence of characters that define a search pattern.

```
## 比如: [ab] 表示寻找 a 或 b
c( "abc", "chen", "liu", "blah" ) %>% str_subset( "[ab]" );
```

```
## [1] "abc" "blah"
```

```
## 匹配并取出字符中间的数字
c( "a1334bc", "ch13e_45n", "liu", "b100ah" ) %>% str_extract( "\\d+" );
```

```
## [1] "1334" "13" NA "00"
```



# useful tools

<https://regexr.com/> <https://regex101.com/>

The screenshot shows the regex101.com website interface. The top navigation bar includes links for @regex101, donate, contact, bug reports & feedback, and a wiki. The main interface is divided into several sections:

- SAVE & SHARE:** Includes a 'Save Regex' button with a share icon and a '+s' suffix.
- FLAVOR:** A list of programming languages and their respective regex engines: PCRE (PHP) (checked), ECMAScript (JavaScript), Python, and Golang.
- TOOLS:** Includes 'Code Generator' and 'Regex Debugger'.
- REGULAR EXPRESSION:** The input field contains the pattern `/wei\b` with flags `/gm`. A status bar indicates '1 match, 5 steps (~0ms)'.
- TEST STRING:** The input field contains the string 'chen wei hua'. The word 'wei' is highlighted in blue, indicating a match.
- EXPLANATION:** Shows the match details: 'wei\b / gm' and 'wei matches the characters wei literally (case sensitive)'.
- MATCH INFORMATION:** Shows 'Match 1' with 'Full match' at positions 5-8, corresponding to the word 'wei'.
- QUICK REFERENCE:** Includes a 'Search reference' input field and a list of tokens: 'All Tokens', 'A single ... [abc]', and 'A char... [^abc]'.

Figure 1: regex101

# 正则表达式的组成部分

## ❶ 字符规则 (Character classes): (不) 匹配什么样的字符

Character Classes	
<code>[:digit:]</code> or <code>\d</code>	Digits; [0-9]
<code>\D</code>	Non-digits; [^0-9]
<code>[:lower:]</code>	Lower-case letters; [a-z]
<code>[:upper:]</code>	Upper-case letters; [A-Z]
<code>[:alpha:]</code>	Alphabetic characters; [A-z]
<code>[:alnum:]</code>	Alphanumeric characters [A-z0-9]
<code>\w</code>	Word characters; [A-z0-9_]
<code>\W</code>	Non-word characters
<code>[:xdigit:]</code> or <code>\x</code>	Hexadec. digits; [0-9A-Fa-f]
<code>[:blank:]</code>	Space and tab
<code>[:space:]</code> or <code>\s</code>	Space, tab, vertical tab, newline, form feed, carriage return
<code>\S</code>	Not space; [^[:space:]]
<code>[:punct:]</code>	Punctuation characters; !"#\$%&'()*+,-./:;<=>?@[ ]^_`{ }~
<code>[:graph:]</code>	Graphical char.; [[:alnum:]][[:punct:]]
<code>[:print:]</code>	Printable characters; [[:alnum:]][[:punct:]]\s
<code>[:cntrl:]</code> or <code>\c</code>	Control characters; \n, \r etc.

Figure 2: 字符规则

# 示例

```
"abc_123_?$$^" %>% str_extract( "\\s+" ); ## 此字符串包括 空格 吗？
```

```
## [1] NA
```

```
"abc_123_?$$^" %>% str_extract( "\\d+" ); ## 数字 ??
```

```
## [1] "123"
```

```
"abc_123_?$$^" %>% str_extract( "\\w+" ); ## [A-z0-9_]
```

```
## [1] "abc_123_"
```

str\_extract : 取出第一个匹配的部分

## 2. 匹配位置

Anchors	
<code>^</code>	Start of the string
<code>\$</code>	End of the string
<code>\\b</code>	Empty string at either edge of a word
<code>\\B</code>	NOT the edge of a word
<code>\\&lt;</code>	Beginning of a word
<code>\\&gt;</code>	End of a word

Figure 3: 匹配位置

# 示例

```
## 以 wei 结束的字符串
c("chen wei hua", "chen wei", "chen") %>% str_subset( "wei$" );
```

```
## [1] "chen wei"
```

```
## 以 wei 结束的字
c("chen wei hua", "chen wei", "chen") %>% str_subset( "wei\\b" );
```

```
## [1] "chen wei hua" "chen wei"
```

### 3. 匹配数量

Quantifiers	
*	Matches at least 0 times
+	Matches at least 1 time
?	Matches at most 1 time; optional string
{n}	Matches exactly n times
{n,}	Matches at least n times
{,n}	Matches at most n times
{n,m}	Matches between n and m times

Figure 4: 匹配数量

# 示例

```
##
"1234abc" %>% str_extract( "\\d+" );
```

```
## [1] "1234"
```

```
"1234abc" %>% str_extract( "\\d{3}" );
```

```
## [1] "123"
```

```
"1234abc" %>% str_extract( "\\d{5,6}" );
```

```
## [1] NA
```

```
"1234abc" %>% str_extract( "\\d{2,6}" );
```

```
## [1] "1234"
```

## 4. classes and groups

Character Classes and Groups	
.	Any character except \n
	Or, e.g. (a b)
[...]	List permitted characters, e.g. [abc]
[a-z]	Specify character ranges
[^...]	List excluded characters
(...)	Grouping, enables back referencing using \\N where N is an integer

**Figure 5:** classes and groups



## 5. 特别字符

Special Metacharacters	
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Tab
<code>\v</code>	Vertical tab
<code>\f</code>	Form feed

Figure 6: 特别字符

## section 4: tasks of regular expression

# tasks of regular expression

## ① detect patterns : 检查目标 string 里有无 pattern

```
grep( "\\d+", c( "123", "abc", "wei555hua" ) ); ##
```



```
## [1] 1 3
```

```
grep1( "\\d+", c( "123", "abc", "wei555hua" ) ); ##
```

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

```
c( "123", "abc", "wei555hua" ) %>% str_detect( "\\d+" );
```

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

# 1. detection patterns, cont.

count patterns: 统计匹配的数量

```
x <- c("why", "video", "cross", "extra", "deal", "authority");  
str_detect(x, "[aeiou]");
```

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

```
str_count(x, "[aeiou]");
```

```
## [1] 0 3 1 2 2 4
```

## 2. locate patterns (定位)

```
regexpr( "\\d+", c( "123", "abc", "wei555hua" ) ); ##
```

```
## [1] 1 -1 4
## attr(,"match.length")
## [1] 3 -1 3
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

```
c( "123", "abc", "wei555hua" ) %>% str_locate( "\\d+" );
```

```
##      start end
## [1,]     1  3
## [2,]    NA NA
## [3,]     4  6
```

### 3. extract patterns (抽取匹配的字串)

```
c( "123", "abc", "wei555hua" ) %>% str_extract ( "\\d+" );
```

```
## [1] "123" NA      "555"
```

```
c( "123", "abc", "wei555hua" ) %>% str_match ( "\\d+" );
```

```
##      [,1]
## [1,] "123"
## [2,] NA
## [3,] "555"
```

?? str\_extract`` 和 str\_match `` 的区别在哪 ??

# str\_extract vs. str\_match

```
x;
```

```
## [1] "why"      "video"    "cross"    "extra"    "deal"     "authority"
```

```
str_extract(x, "[aeiou]");
```

```
## [1] NA "i" "o" "e" "e" "a"
```

```
str_match(x, "(.)[aeiou](.)"); ## extract the characters on either side of the vowel ????
```

```
##      [,1] [,2] [,3]
## [1,] NA   NA   NA
## [2,] "vid" "v"  "d"
## [3,] "ros"  "r"  "s"
## [4,] NA    NA   NA
## [5,] "dea"  "d"  "a"
## [6,] "aut"  "a"  "t"
```



?? 为什么有 3 个输出??

# str\_extract\_all 和 str\_match\_all

```
x;

## [1] "why"      "video"    "cross"    "extra"    "deal"     "authority"

str_extract_all( x, "[aeiou]+" );

## [[1]]
## character(0)
##
## [[2]]
## [1] "i"  "eo"
##
## [[3]]
## [1] "o"
##
## [[4]]
## [1] "e" "a"
##
## [[5]]
## [1] "ea"
##
## [[6]]
## [1] "au" "o"  "i"

str_match_all( x, "[aeiou]+" );
```



## 4. replace patterns (匹配并替换)

```
str_replace( c( "123", "abc", "wei555hua" ) , "\\d+", "###");
```

```
## [1] "###"      "abc"      "wei###hua"
```

```
str_replace_all( "123_abc_456_789" , "\\d+", "###");
```

```
## [1] "###_abc_###_###"
```

## 5. split by patterns

```
str_split(x, "");
```

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

```
## [1] "w" "h" "y"
```

```
##
```

```
## [[2]]
```

```
## [1] "v" "i" "d" "e" "o"
```

```
##
```

```
## [[3]]
```

```
## [1] "c" "r" "o" "s" "s"
```

```
##
```

```
## [[4]]
```

```
## [1] "e" "x" "t" "r" "a"
```

```
##
```

```
## [[5]]
```

```
## [1] "d" "e" "a" "l"
```

# stringr 函数

更多内容见这里: <https://stringr.tidyverse.org>

# 示例

```
( dat <-
tibble(chrom = readLines(textConnection("chr11:69464719-69502928
chr7:55075808-55093954
chr8:128739772-128762863
chr3:169389459-169490555
chr17:37848534-37877201
chr19:30306758-30316875
chr1:150496857-150678056
chr12:69183279-69260755
chr11:77610143-77641464
chr8:38191804-38260814
chr12:58135797-58156509"))) ) );
```

```
## # A tibble: 11 x 1
##   chrom
##   <chr>
## 1 chr11:69464719-69502928
## 2 chr7:55075808-55093954
## 3 chr8:128739772-128762863
## 4 chr3:169389459-169490555
## 5 chr17:37848534-37877201
## 6 chr19:30306758-30316875
## 7 chr1:150496857-150678056
## 8 chr12:69183279-69260755
## 9 chr11:77610143-77641464
## 10 chr8:38191804-38260814
## 11 chr12:58135797-58156509
```

# 示例, cont.

任务：分为三列，chr, start, end

```
dat$chrom %>% str_split( '[:-]', simplify = T );
```

```
##      [,1]      [,2]      [,3]
## [1,] "chr11" "69464719" "69502928"
## [2,] "chr7"  "55075808" "55093954"
## [3,] "chr8"  "128739772" "128762863"
## [4,] "chr3"  "169389459" "169490555"
## [5,] "chr17" "37848534"  "37877201"
## [6,] "chr19" "30306758"  "30316875"
## [7,] "chr1"  "150496857" "150678056"
## [8,] "chr12" "69183279"  "69260755"
## [9,] "chr11" "77610143"  "77641464"
## [10,] "chr8"  "38191804"  "38260814"
## [11,] "chr12" "58135797"  "58156509"
```

# 示例, cont.

## 另一种解决方案

```
library(tidyr)
extract(dat, chrom, into=c('chr', 'chrStart', 'chrEnd'),
        '([^:]+):([^-]+)-(.*)', convert=TRUE);
```

```
## # A tibble: 11 x 3
##   chr    chrStart  chrEnd
##   <chr>    <int>    <int>
## 1 chr11  69464719  69502928
## 2 chr7   55075808  55093954
## 3 chr8  128739772 128762863
## 4 chr3  169389459 169490555
## 5 chr17 37848534  37877201
## 6 chr19 30306758  30316875
## 7 chr1  150496857 150678056
## 8 chr12 69183279  69260755
## 9 chr11 77610143  77641464
## 10 chr8  38191804  38260814
## 11 chr12 58135797  58156509
```

## section 5: Exercise and home work

# 小结

## 今次提要

- ① string basics
  - length
  - uppercase, lowercase
  - unite, separate
  - string comparisons, sub string
- ② regular expression
  - detect patterns
  - locate patterns
  - extract patterns
  - replace patterns
  - split patterns ...

## 下次预告

data iteration & parallel computing



# 练习 & 作业

- Exercises and homework 目录下 talk07-homework.Rmd 文件;
- 完成时间: 见钉群的要求