Data Analysis Tools and Practice(Using R)

2020.04.30

String&Time ...

北京大学 软件与微电子学院
School of Software and Microelectronics, Peking University

Huiping Sun(孙惠平)
sunhp@ss.pku.edu.cn

包介绍

提交: PPT、代码（R或Rmd）、介绍或说明文档

各组缺介绍和说明文档的补一下

课程项目

5月7日前确定下来发到群里

每组找一个有代表性的数据分析题目

kaggle、天池、数字中国、比特币、企查查 ...

独立完成数据分析工作、和现有工作比较、工作量&新意

字符处理

字符函数

nchar()	计算x中的字符数
substr(s,start,stop)	提取或替换一个字符向量中的子串
strsplit(x,split)	在split处分割字符向量x中的元素
toupper(x), tolower()	大小写转换
paste(..., sep="")	连接字符串
grep(pattern,x,ignore.case=FALSE,fixed=FLASE)	搜索
sub(pattern,replacement,x,ignore.case=FALSE,fixed=FLASE)	搜索替换

字符函数

```
> paste("My","Job")
[1] "My Job"
>
> labs <- paste("X",1:6, sep="")
> labs
[1] "X1" "X2" "X3" "X4" "X5" "X6"
>
> paste("Today is", date())
[1] "Today is Wed Mar  2 12:41:21 2016"
>
> paste(c("a","b"),collapse=".")
[1] "a.b"
```

文本文件输入输出

```
> cat("这是一个测试", "2 3 5 7", "", "11 13 17", file = "testCN.data",sep = "\n")
> text <- readLines("testCN.data", encoding = "UTF-8")
> text
[1] "这是一个测试" "2 3 5 7"      ""                 "11 13 17"
```

cat **readLines**

```
> scan("testCN.data", what = character(0))
Read 8 items
[1] "这是一个测试" "2"            "3"                "5"                "7"
[6] "11"           "13"           "17"
> scan("testCN.data", what = character(0), sep = "\n")
Read 3 items
[1] "这是一个测试" "2 3 5 7"      "11 13 17"
```

每个单词作为字符向量的一个元素

scan

每一行文本作为向量的一个元素

scan("testCN.data", what = character(0), sep = ".")

writeLines

cat(text, file = "file.txt", sep = "\n")
writeLines(text, con = "file.txt", sep = "\n", useBytes = F)

字符统计与转换

```
> x <- c("we are the world", "we are the children")
> x
[1] "we are the world"    "we are the children"
> nchar(x)
[1] 16 19
> length(x)
[1] 2
```

```
> nchar("")
[1] 0
> length("")
[1] 1
```

nchar(NA)：2

中文：2个Bytes

```
> dna <- "AgCTaaGGGcctTagct"
> dna
[1] "AgCTaaGGGcctTagct"
> tolower(dna)
[1] "agctaagggccttagct"
> toupper(dna)
[1] "AGCTAAGGGCCTTAGCT"
> chartr("Tt", "Uu", dna)
[1] "AgCUaaGGGccuUagcu"
```

tolower　toupper　chartr

字符串拆分

strsplit(x, split, fixed= F, perl= F, useBytes= F)

- x：为字符串格式向量，函数依次对向量的每个元素进行拆分

- split：拆分位置的字串向量，即在哪个字串处开始拆分；该参数默认是**正则表达式**匹配；若设置fixed= T则表示是用普通文本匹配或者正则表达式的精确匹配。用普通文本来匹配的运算速度要快些

- per：逻辑值，是否兼容perl的正则表达（regexps）

- useBytes：是否逐字节进行匹配，默认为FALSE，表示是按字符匹配而不是按字节进行匹配

strsplit

字符串拆分

```
> text <- "We are the world.\nWe are the children!"
> text
[1] "We are the world.\nWe are the children!"
> cat(text)
We are the world.
We are the children!
> strsplit(text, " ")
[[1]]
[1] "We"          "are"         "the"         "world.\nWe" "are"         "the"
[7] "children!"

> strsplit(text, "\\s")
[[1]]
[1] "We"        "are"       "the"       "world."    "We"        "are"       "the"
[8] "children!"
```

正则表达式

匹配一个空白字符，包括空格、制表符和换行符

```
> class(strsplit(text, "\\s"))
[1] "list"
> unlist(strsplit(text, "\\s"))
[1] "We"        "are"       "the"       "world."    "We"        "are"       "the"
[8] "children!"
```

字符串截取

字符串查询

grep(pattern, x, **ignore.case= F**, perl= F, value= F, fixed= F, useBytes= F, **invert= F**)

grepl(pattern, x, ignore.case= F, perl= F, fixed= F, useBytes= F)

```
> text <- c("We are the world", "we are the children")
> grep("We", text)
[1] 1
> grep("We", text, invert = T)
[1] 2
> grep("we", text, ignore.case = T)
[1] 1 2
> grepl("are", text)
[1] TRUE TRUE
```

匹配与否

忽略大小写

逻辑

pattern
正则表达式

字符串查询

regexpr(pattern, text, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

gregexpr(pattern, text, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

regexec(pattern, text, ignore.case = FALSE, fixed = FALSE, useBytes = FALSE)

与grep()和grepl()不同的是它们返回的结果中包含了匹配的具体位置和字符串长度信息

```
> regexpr("e", text)
[1] 2 2
attr(,"match.length")
[1] 1 1
attr(,"useBytes")
[1] TRUE
> class(regexpr("e", text))
[1] "integer"
```

第一次

regex
Regular Expression

```
> gregexpr("e", text)
[[1]]
[1]  2  6 10
attr(,"match.length")
[1] 1 1 1
attr(,"useBytes")
[1] TRUE

[[2]]
[1]  2  6 10 18
attr(,"match.length")
[1] 1 1 1 1
attr(,"useBytes")
[1] TRUE

> class(gregexpr("e", text))
[1] "list"
```

所有

```
> regexec("e", text)
[[1]]
[1] 2
attr(,"match.length")
[1] 1

[[2]]
[1] 2
attr(,"match.length")
[1] 1

> class(regexec("e", text))
[1] "list"
```

返回信息更丰富

正则表达式规则

[ ]：括号内的任意字符将被匹配；

\：具有两个作用：
1.对元字符进行转义
2.一些以\开头的特殊序列表达了一些字符串组

^：匹配字符串的开始.将^置于character class的首位表达的意思是取反义。如[^5]表示匹配除了”5”以外的任何字符

$：匹配字符串的结束。但将它置于character class内则消除了它的特殊含义。如[akm$]将匹配’a’,’k’,’m’或者’$’

+：前面的字符(组)将被匹配一次或多次

*：前面的字符(组)将被匹配零次或多次

.：匹配除换行符以外的任意字符。

?：前面的字符(组)是可有可无的，并且最多被匹配一次

( )：表示一个字符组，括号内的字符串将作为一个整体被匹配

|：或者

正则表达式规则

代码	含义说明
`?`	重复零次或一次
`*`	重复零次或多次
`+`	重复一次或多次
`{n}`	重复n次
`{n,}`	重复n次或更多次
`{n,m}`	重复n次到m次

代码	含义说明
`[:digit:]`	数字：0-9
`[:lower:]`	小写字母：a-z
`[:upper:]`	大写字母：A-Z
`[:alpha:]`	字母：a-z及A-Z
`[:alnum:]`	所有字母及数字
`[:punct:]`	标点符号，如`. , ;`等
`[:graph:]`	Graphical characters,即[:alnum:]和[:punct:]
`[:blank:]`	空字符，即：Space和Tab
`[:space:]`	Space，Tab，newline，及其他space characters
`[:print:]`	可打印的字符，即：[:alnum:]，[:punct:]和[:space:]

代码	含义说明
`\w`	字符串，等价于`[:alnum:]`
`\W`	非字符串，等价于`[^[:alnum:]]`
`\s`	空格字符，等价于`[:blank:]`
`\S`	非空格字符，等价于`[^[:blank:]]`
`\d`	数字，等价于`[:digit:]`
`\D`	非数字，等价于`[^[:digit:]]`
`\b`	Word edge（单词开头或结束的位置）
`\B`	No Word edge（非单词开头或结束的位置）
`\<`	Word beginning（单词开头的位置）
`\>`	Word end（单词结束的位置）

Stringr

函数列表

str_c:　　字符串拼接。
str_join:　　字符串拼接，同str_c。
str_trim:　　去掉字符串的空格和TAB(\t)
str_pad:　　补充字符串的长度
str_dup:　　复制字符串
str_wrap:　　控制字符串输出格式
str_sub:　　截取字符串
str_sub<-　　截取字符串，并赋值，同str_sub

str_count:　　字符串计数
str_length:　　字符串长度
str_sort:　　字符串值排序
str_order:　　字符串索引排序，规则同str_sort

str_conv:　　字符编码转换
str_to_upper: 字符串转成大写
str_to_lower: 字符串转成小写,规则同str_to_upper
str_to_title: 字符串转成首字母大写,规则同str_to_upper

str_split:　　字符串分割
str_split_fixed: 字符串分割，同str_split
str_subset: 返回匹配的字符串
word:　　从文本中提取单词
str_detect: 检查匹配字符串的字符
str_match:　从字符串中提取匹配组。
str_match_all: 从字符串中提取匹配组，同str_match
str_replace: 字符串替换
str_replace_all: 字符串替换，同str_replace
str_replace_na:把NA替换为NA字符串
str_locate: 找到匹配的字符串的位置。
str_locate_all: 找到匹配的字符串的位置,同str_locate
str_extract: 从字符串中提取匹配字符
str_extract_all: 从字符串中提取匹配字符，同str_extract

boundary:　　定义使用边界
coll:定义字符串标准排序规则。
fixed:　　定义用于匹配的字符，包括正则表达式中的转义符
regex:　　定义正则表达式

字符串拼接

str_c(..., sep = "", collapse = NULL)
str_join(..., sep = "", collapse = NULL)

把多个字符串拼接为一个大的字符串

…: 多参数的输入

sep: 把多个字符串拼接为一个大的字符串，用于字符串的分割符。

collapse: 把多个向量参数拼接为一个大的字符串，用于字符串的分割符。

```
> str_c('a','b')
[1] "ab"
> str_c('a','b',sep='-')
[1] "a-b"
> str_c(c('a','a1'),c('b','b1'),sep='-')
[1] "a-b"   "a1-b1"
```

```
> str_c(head(letters), collapse = "")
[1] "abcdef"
> str_c(head(letters), collapse = ", ")
[1] "a, b, c, d, e, f"
> str_c('a','b',collapse = "-")
[1] "ab"
> str_c(c('a','a1'),c('b','b1'),collapse='-')
[1] "ab-a1b1"
```

```
> str_c('a','b')
[1] "ab"
> paste('a','b')
[1] "a b"
> str_c(head(letters), collapse = "")
[1] "abcdef"
> paste(head(letters), collapse = "")
[1] "abcdef"
> str_c(c("a", NA, "b"), "-d")
[1] "a-d" NA    "b-d"
> paste(c("a", NA, "b"), "-d")
[1] "a -d"  "NA -d" "b -d"
```

去除空格和TAB

str_trim(string, side = c("both", "left", "right"))

string: 字符串，字符串向量。
side: 过滤方式，both两边都过滤，left左边过滤，right右边过滤

```
> str_trim("  left space\t\n",side='left')
[1] "left space\t\n"
> str_trim("  left space\t\n",side='right')
[1] "  left space"
> str_trim("  left space\t\n",side='both')
[1] "left space"
> str_trim("\nno space\n\t")
[1] "no space"
```

补充字符串长度

str_pad(string, width, side = c("left", "right", "both"), pad = " ")

string: 字符串，字符串向量。
width: 字符串填充后的长度
side: 填充方向，both两边都填充，left左边填充，right右边填充
pad: 用于填充的字符

```
> str_pad("conan", 20, "left")
[1] "               conan"
> str_pad("conan", 20, "right")
[1] "conan               "
> str_pad("conan", 20, "both")
[1] "       conan        "
> str_pad("conan", 20, "both",'x')
[1] "xxxxxxxconanxxxxxxxx"
```

字符串复制

str_dup(string, times)

string: 字符串，字符串向量。
times: 复制数量

```
> val <- c("abca4", 123, "cba2")
> str_dup(val, 2)
[1] "abca4abca4" "123123"     "cba2cba2"
> str_dup(val, 1:3)
[1] "abca4"         "123123"        "cba2cba2cba2"
```

字符串输出格式

str_wrap(string, width = 80, indent = 0, exdent = 0)

string: 字符串，字符串向量。
width: 设置一行所占的宽度。
indent: 段落首行的缩进值
exdent: 段落非首行的缩进值

```
> txt<-'R语言作为统计学一门语言，一直在小众领域闪耀着光芒。直到大数据的爆发，R语言变成了一门炙手可热的数据分析的利器。随着越来越多的工程背景的人的加入，R语言的社区在迅速扩大成长。现在已不仅仅是统计领域，教育，银行，电商，互联网….都在使用R语言。'
> cat(str_wrap(txt, width = 40), "\n")
R语言作为统计学一门语言，一直在小众领域
闪耀着光芒。直到大数据的爆发，R语言变成
了一门炙手可热的数据分析的利器。随着越来
越多的工程背景的人的加入，R语言的社区在
迅速扩大成长。现在已不仅仅是统计领域，教
育，银行，电商，互联网….都在使用R语言。
> cat(str_wrap(txt, width = 60, indent = 2), "\n")
  R语言作为统计学一门语言，一直在小众领域闪耀着光芒。直到大数
据的爆发，R语言变成了一门炙手可热的数据分析的利器。随着越来
越多的工程背景的人的加入，R语言的社区在迅速扩大成长。现在已
不仅仅是统计领域，教育，银行，电商，互联网….都在使用R语言。
> cat(str_wrap(txt, width = 10, exdent = 4), "\n")
R语言作为
    统计学一
    门语言，
    一直在小
    众领域闪
    耀着光芒。
    直到大数据
    的爆发，R
    语言变成了
    一门炙手可
    热的数据分
    析的利器。
    随着越来
    越多的工程
    背景的人的
    加入，R语
    言的社区在
    迅速扩大成
    长。现在已
    不仅仅是统
    计领域，教
    育，银行，
    电商，互联
    网….都在使
    用R语言。
```

字符串取子集

str_sub(string, start = 1L, end = -1L)

string: 字符串，字符串向量。
start : 开始位置
end : 结束位置

```
> txt <- "I am Conan."
> str_sub(txt, 1, 4)
[1] "I am"
> str_sub(txt, end=6)
[1] "I am C"
> str_sub(txt, 6)
[1] "Conan."
> str_sub(txt, c(1, 4), c(6, 8))
[1] "I am C" "m Con"
```

```
> x <- "AAABBBCCC"
> str_sub(x, 1, 1) <- 1; x
[1] "1AABBBCCC"
> str_sub(x, 2, -2) <- "2345"; x
[1] "12345C"
```

```
> str_sub(txt, -3)
[1] "an."
> str_sub(txt, end = -3)
[1] "I am Cona"
```

字符串计数

str_count(string, pattern = "")

string: 字符串，字符串向量。
pattern: 匹配的字符。

```
> str_count('aaa444sssddd', "a")
[1] 3
> fruit <- c("apple", "banana", "pear", "pineapple")
> str_count(fruit, "a")
[1] 1 3 1 1
> str_count(fruit, "p")
[1] 2 0 1 3
> str_count(c("a.", ".", ".a.",NA), ".")
[1]  2  1  3 NA
> str_count(c("a.", ".", ".a.",NA), fixed("."))
[1]  1  1  2 NA
> str_count(c("a.", ".", ".a.",NA), "\\.")
[1]  1  1  2 NA
```

str_length(string)

```
> str_length(c("I", "am", "张丹", NA))
[1]  1  2  2 NA
```

字符串排序

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "", ...)
str_order(x, decreasing = FALSE, na_last = TRUE, locale = "", ...)

x: 字符串，字符串向量。

decreasing: 排序方向。

na_last:NA值的存放位置，一共3个值，TRUE放到最后，FALSE放到最前，NA过滤处理

locale:按哪种语言习惯排序

```
> str_sort(c('a',1,2,'11'), locale = "en")
[1] "1"  "11" "2"  "a"
> str_sort(letters,decreasing=TRUE)
 [1] "z" "y" "x" "w" "v" "u" "t" "s" "r" "q" "p" "o" "n" "m" "l" "k" "j" "i" "h" "g" "f" "e"
[23] "d" "c" "b" "a"
> str_sort(c('你','好','粉','丝','日','志'),locale = "zh")
[1] "粉" "好" "你" "日" "丝" "志"
> str_sort(c(NA,'1',NA),na_last=TRUE)
[1] "1" NA  NA
> str_sort(c(NA,'1',NA),na_last=FALSE)
[1] NA  NA  "1"
> str_sort(c(NA,'1',NA),na_last=NA)
[1] "1"
```

字符串划分

str_split(string, pattern, n = Inf)
str_split_fixed(string, pattern, n)

string: 字符串，字符串向量。
pattern: 匹配的字符。
n: 分割个数

```
> val <- "abc,123,234,iuuu"
> s1<-str_split(val, ",");s1
[[1]]
[1] "abc"  "123"  "234"  "iuuu"

> s2<-str_split(val, ",",2);s2
[[1]]
[1] "abc"           "123,234,iuuu"

> class(s1)
[1] "list"
> s3<-str_split_fixed(val, ",",2);s3
     [,1]  [,2]
[1,] "abc" "123,234,iuuu"
> class(s3)
[1] "matrix"
```

字符串取子集

str_subset(string, pattern)

string: 字符串，字符串向量。
pattern: 匹配的字符。

```
> val <- c("abc", 123, "cba")
> str_subset(val, "a")
[1] "abc" "cba"
> str_subset(val, "^a")
[1] "abc"
> str_subset(val, "a$")
[1] "cba"
```

文中提取单词

word(string, start = 1L, end = start, sep = fixed(" "))

string: 字符串，字符串向量。
start: 开始位置。
end: 结束位置。
sep: 匹配字符。

```
> val <- c("I am Conan.", "http://fens.me, ok")
> word(val, 1)
[1] "I"                  "http://fens.me,"
> word(val, -1)
[1] "Conan." "ok"
> word(val, 2, -1)
[1] "am Conan." "ok"
```

其余函数

str_to_upper(string, locale = "")
str_to_lower(string, locale = "")
str_to_title(string, locale = "")

str_extract(string, pattern)
str_extract_all(string, pattern, simplify = FALSE)

str_conv(string, encoding)

str_detect(string, pattern)

str_replace(string, pattern, replacement)
str_replace_na(string, replacement = "NA")

str_locate(string, pattern)
str_locate_all(string, pattern)

str_match(string, pattern)
str_match_all(string, pattern)

函数对应

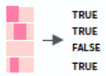
函数	功能说明	R Base中对应函数
使用正则表达式的函数		
`str_extract()`	提取首个匹配模式的字符	`regmatches()`
`str_extract_all()`	提取所有匹配模式的字符	`regmatches()`
`str_locate()`	返回首个匹配模式的字符的位置	`regexpr()`
`str_locate_all()`	返回所有匹配模式的字符的位置	`gregexpr()`
`str_replace()`	替换首个匹配模式	`sub()`
`str_replace_all()`	替换所有匹配模式	`gsub()`
`str_split()`	按照模式分割字符串	`strsplit()`
`str_split_fixed()`	按照模式将字符串分割成指定个数	-
`str_detect()`	检测字符是否存在某些指定模式	`grepl()`
`str_count()`	返回指定模式出现的次数	-
`str_sub()`	提取指定位置的字符	`regmatches()`
`str_dup()`	丢弃指定位置的字符	-
`str_length()`	返回字符的长度	`nchar()`
`str_pad()`	填补字符	-
`str_trim()`	丢弃填充，如去掉字符前后的空格	-
`str_c()`	连接字符	`paste(),paste0()`

String&Time... CheatSheet

String manipulation with stringr : : CHEAT SHEET

The **stringr** package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

Detect Matches



str_detect(string, **pattern**) Detect the presence of a pattern match in a string. *str_detect(fruit, "a")*

str_which(string, **pattern**) Find the indexes of strings that contain a pattern match. *str_which(fruit, "a")*

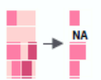
str_count(string, **pattern**) Count the number of matches in a string. *str_count(fruit, "a")*

str_locate(string, **pattern**) Locate the positions of pattern matches in a string. Also **str_locate_all**. *str_locate(fruit, "a")*

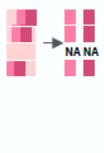
Subset Strings

str_sub(string, start = 1L, end = -1L) Extract substrings from a character vector. *str_sub(fruit, 1, 3); str_sub(fruit, -2)*

str_subset(string, **pattern**) Return only the strings that contain a pattern match. *str_subset(fruit, "b")*



str_extract(string, **pattern**) Return the first pattern match found in each string, as a vector. Also **str_extract_all** to return every pattern match. *str_extract(fruit, "[aeiou]")*



str_match(string, **pattern**) Return the first pattern match found in each string, as a matrix with a column for each ( ) group in pattern. Also **str_match_all**. *str_match(sentences, "(a|the) ([^]+)")*

Manage Lengths

str_length(string) The width of strings (i.e. number of code points, which generally equals the number of characters). *str_length(fruit)*

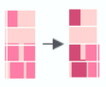
str_pad(string, width, side = c("left", "right", "both"), pad = " ") Pad strings to constant width. *str_pad(fruit, 17)*

str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Truncate the width of strings, replacing content with ellipsis. *str_trunc(fruit, 3)*

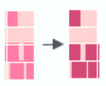
str_trim(string, side = c("both", "left", "right")) Trim whitespace from the start and/or end of a string. *str_trim(fruit)*

Mutate Strings

str_sub() <- value. Replace substrings by identifying the substrings with str_sub() and assigning into the results. *str_sub(fruit, 1, 3) <- "str"*

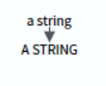


str_replace(string, **pattern**, replacement) Replace the first matched pattern in each string. *str_replace(fruit, "a", "-")*

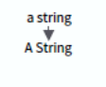


str_replace_all(string, **pattern**, replacement) Replace all matched patterns in each string. *str_replace_all(fruit, "a", "-")*

str_to_lower(string, locale = "en")[1] Convert strings to lower case. *str_to_lower(sentences)*



str_to_upper(string, locale = "en")[1] Convert strings to upper case. *str_to_upper(sentences)*



str_to_title(string, locale = "en")[1] Convert strings to title case. *str_to_title(sentences)*

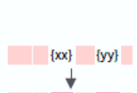
Join and Split

str_c(..., sep = "", collapse = NULL) Join multiple strings into a single string. *str_c(letters, LETTERS)*

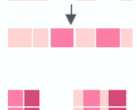
str_c(..., sep = "", **collapse = ""**) Collapse a vector of strings into a single string. *str_c(letters, collapse = "")*

str_dup(string, times) Repeat strings times times. *str_dup(fruit, times = 2)*

str_split_fixed(string, **pattern**, n) Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also **str_split** to return a list of substrings. *str_split_fixed(fruit, " ", n=2)*

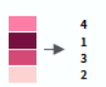


str_glue(..., .sep = "", .envir = parent.frame()) Create a string from strings and {expressions} to evaluate. *str_glue("Pi is {pi}")*



str_glue_data(.x, ..., .sep = "", .envir = parent.frame(), .na = "NA") Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate. *str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")*

Order Strings

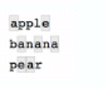


str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Return the vector of indexes that sorts a character vector. *x[str_order(x)]*

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)[1] Sort a character vector. *str_sort(x)*

Helpers

str_conv(string, encoding) Override the encoding of a string. *str_conv(fruit,"ISO-8859-1")*



str_view(string, **pattern**, match = NA) View HTML rendering of first regex match in each string. *str_view(fruit, "[aeiou]")*

str_view_all(string, **pattern**, match = NA) View HTML rendering of all regex matches. *str_view_all(fruit, "[aeiou]")*

str_wrap(string, width = 80, indent = 0, exdent = 0) Wrap strings into nicely formatted paragraphs. *str_wrap(sentences, 20)*

RStudio

[1] See bit.ly/ISO639-1 for a complete list of locales.

https://www.rdocumentation.org/packages/stringr/versions/1.4.0

Lubridate

日期函数和格式

日期函数	*as.Date(x, "input_format")*
%d	数字表示的日期（0-31）
%a, %A	星期名（缩写，非缩写）
%m	月份（0-12）
%b, %B	月份（缩写，非缩写）
%y, %Y	年份（两位，四位）
Sys.Date(), date(), difftime(), format()	

日期函数例子

```
> mydates <- as.Date(c("2007-06-22"))
> mydates
[1] "2007-06-22"
> mydates <- as.Date(c("2007-06-22"))
>
> strDates <- c("01/05/1965")
> dates <- as.Date(strDates, "%m/%d/%Y")
>
> Sys.Date()
[1] "2016-03-02"
> date()
[1] "Wed Mar  2 12:48:52 2016"
```

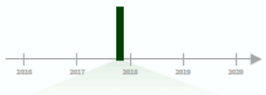
日期函数例子

```
> today <- Sys.Date()
> format(today, format = "%B %d %Y")
[1] "March 02 2016"
> format(today, format = "%A")
[1] "Wednesday"
>
> startdate <- as.Date("2004-02-13")
> enddate <- as.Date("2009-06-22")
> days <- enddate - startdate
>
> today <- Sys.Date()
> format(today, format = "%B %d %Y")
[1] "March 02 2016"
> dob <- as.Date("1956-10-10")
> format(dob, format = "%A")
[1] "Wednesday"
> difftime(today,dob, units="weeks")
Time difference of 3099 weeks
```

String&Time... CheatSheet

Dates and times with lubridate : : CHEAT SHEET

Date-times



`2017-11-28 12:00:00`

`2017-11-28 12:00:00`
A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

*dt <- **as_datetime**(1511870400)*
"2017-11-28 12:00:00 UTC"

`2017-11-28`
A **date** is a day stored as the number of days since 1970-01-01

*d <- **as_date**(17498)*
"2017-11-28"

`12:00:00`
An hms is a **time** stored as the number of seconds since 00:00:00

*t <- hms::**as.hms**(85)*
00:01:25

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

`2017-11-28T14:02:00` **ymd_hms**(), **ymd_hm**(), **ymd_h**(). *ymd_hms("2017-11-28T14:02:00")*

`2017-22-12 10:00:00` **ydm_hms**(), **ydm_hm**(), **ydm_h**(). *ydm_hms("2017-22-12 10:00:00")*

`11/28/2017 1:02:03` **mdy_hms**(), **mdy_hm**(), **mdy_h**(). *mdy_hms("11/28/2017 1:02:03")*

`1 Jan 2017 23:59:59` **dmy_hms**(), **dmy_hm**(), **dmy_h**(). *dmy_hms("1 Jan 2017 23:59:59")*

`20170131` **ymd**(), **ydm**(). *ymd(20170131)*

`July 4th, 2000` **mdy**(), **myd**(). *mdy("July 4th, 2000")*

`4th of July '99` **dmy**(), **dym**(). *dmy("4th of July '99")*

`2001: Q3` **yq**() Q for quarter. *yq("2001: Q3")*

`2:01` hms::**hms**() Also lubridate::**hms**(), **hm**() and **ms**(), which return periods.* *hms::hms(sec = 0, min= 1, hours = 2)*

`2017.5` **date_decimal**(decimal, tz = "UTC") *date_decimal(2017.5)*

now(tzone = "") Current time in tz (defaults to system tz). *now()*

today(tzone = "") Current date in a tz (defaults to system tz). *today()*

fast_strptime() Faster strptime. *fast_strptime('9/1/01', '%y/%m/%d')*

parse_date_time() Easier strptime. *parse_date_time("9/1/01", "ymd")*

GET AND SET COMPONENTS

Use an accessor function to get a component.

d ## "2017-11-28"
day(d) ## 28

Assign into an accessor function to change a component in place.

day(d) <- 1
d ## "2017-11-01"

`2018-01-31 11:59:59`

`2018-01-31 11:59:59` **date**(x) Date component. *date(dt)*

year(x) Year. *year(dt)*
isoyear(x) The ISO 8601 year.
epiyear(x) Epidemiological year.

month(x, label, abbr) Month. *month(dt)*

day(x) Day of month. *day(dt)*
wday(x,label,abbr) Day of week.
qday(x) Day of quarter.

hour(x) Hour. *hour(dt)*

minute(x) Minutes. *minute(dt)*

second(x) Seconds. *second(dt)*

week(x) Week of the year. *week(dt)*
isoweek() ISO 8601 week.
epiweek() Epidemiological week.

quarter(x, with_year = FALSE) Quarter. *quarter(dt)*

semester(x, with_year = FALSE) Semester. *semester(dt)*

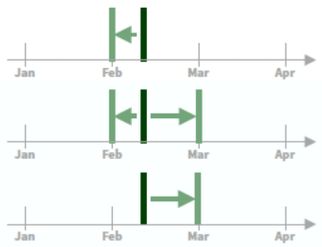
am(x) Is it in the am? *am(dt)*
pm(x) Is it in the pm? *pm(dt)*

dst(x) Is it daylight savings? *dst(d)*

leap_year(x) Is it a leap year? *leap_year(d)*

update(object, ..., simple = FALSE) *update(dt, mday = 2, hour = 1)*

Round Date-times



floor_date(x, unit = "second") Round down to nearest unit. *floor_date(dt, unit = "month")*

round_date(x, unit = "second") Round to nearest unit. *round_date(dt, unit = "month")*

ceiling_date(x, unit = "second", change_on_boundary = NULL) Round up to nearest unit. *ceiling_date(dt, unit = "month")*

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Roll back to last day of previous month. *rollback(dt)*

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date**() and **stamp_time**().

1. Derive a template, create a function
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Apply the template to dates
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

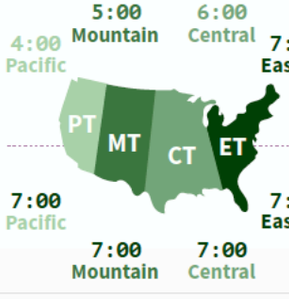
Tip: use a date with day > 12

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns *one* time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. *OlsonNames()*



with_tz(time, tzone = "") Get the **same date-time** in a new time zone (a new clock time). *with_tz(dt, "US/Pacific")*

force_tz(time, tzone = "") Get the **same clock time** in a new time zone (a new date-time). *force_tz(dt, "US/Pacific")*

https://www.rdocumentation.org/packages/lubridate/versions/1.7.8

Rmarkdown

R Markdown : : CHEAT SHEET

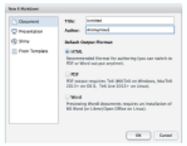
What is R Markdown?

.Rmd files · An R Markdown (.Rmd) file is a record of your research. It contains the code that a scientist needs to reproduce your work along with the narration that a reader needs to understand your work.

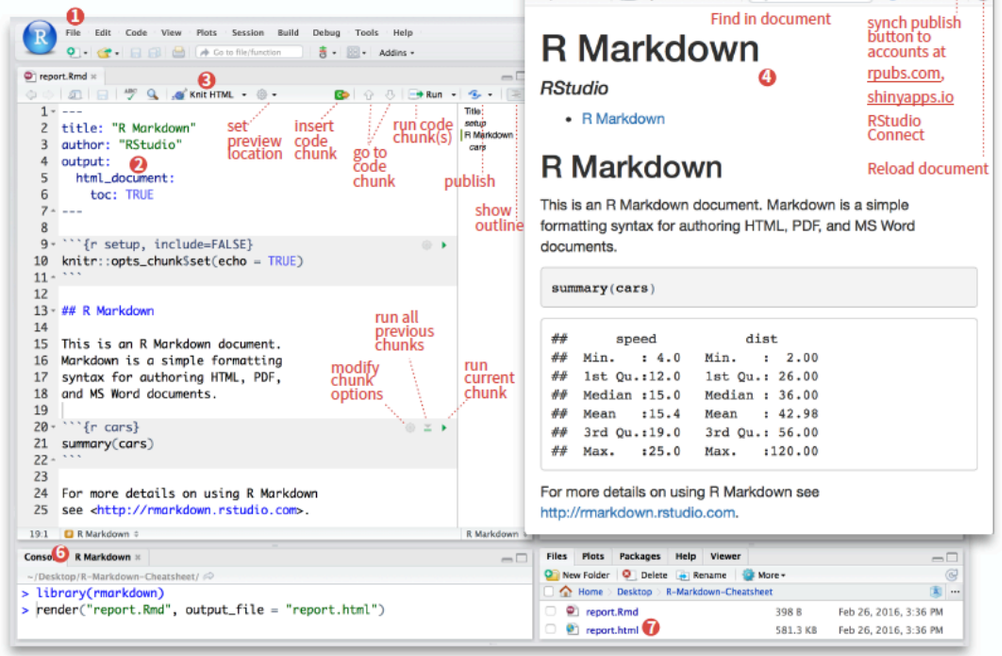
Reproducible Research · At the click of a button, or the type of a command, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.

Dynamic Documents · You can choose to export the finished report in a variety of formats, including html, pdf, MS Word, or RTF documents; html or pdf based slides, Notebooks, and more.

Workflow



1. **Open a new .Rmd file** at File ▶ New File ▶ R Markdown. Use the wizard that opens to pre-populate the file with a template
2. **Write document** by editing template
3. **Knit document to create report**; use knit button or **render()** to knit
4. **Preview Output** in IDE window
5. **Publish** (optional) to web server
6. **Examine build log** in R Markdown console
7. **Use output file** that is saved along side .Rmd



render

Use rmarkdown::**render()** to render/knit at cmd line. Important args:

input - file to render **output_format**	**output_options** - List of render options (as in YAML)	**output_file** **output_dir**	**params** - list of params to use	**envir** - environment to evaluate code chunks in	**encoding** - of input file

.rmd Structure

YAML Header
Optional section of render (e.g. pandoc) options written as key:value pairs (YAML).
At start of file
Between lines of ---

Text
Narration formatted with markdown, mixed with:

Code Chunks
Chunks of embedded code. Each chunk:
Begins with ```{r}
ends with ```
R Markdown will run the code and append the results to the doc.
It will use the location of the .Rmd file as the **working directory**

Parameters

Parameterize your documents to reuse with new inputs (e.g., data, values, etc.)

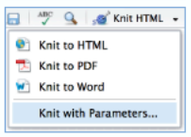
1. **Add parameters** · Create and set parameters in the header as sub-values of params

2. **Call parameters** · Call parameter values in code as params$<name>

3. **Set parameters** · Set values wth Knit with parameters or the params argument of render():
render("doc.Rmd", params = list(n = 1, d = as.Date("2015-01-01"))

```
---
params:
  n: 100
  d: !r Sys.Date()
---
```

Today's date is `r params$d`



Embed code with knitr syntax

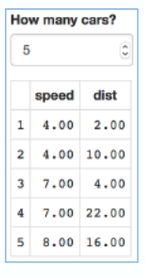
INLINE CODE
Insert with `` `r <code>` ``. Results appear as text without code.
Built with `` `r getRversion()` `` ➡ Built with 3.2.3

CODE CHUNKS
One or more lines surrounded with ```{r} and ```. Place chunk options within curly braces, after **r**. Insert with

```
```{r echo=TRUE}
getRversion()
```
```

➡

```
getRversion()
## [1] '3.2.3'
```

GLOBAL OPTIONS
Set with knitr::**opts_chunk$set()**, e.g.

```
```{r include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```

IMPORTANT CHUNK OPTIONS

cache - cache results for future knits (default = FALSE)

cache.path - directory to save cached results in (default = "cache/")

child - file(s) to knit and then include (default = NULL)

collapse - collapse all output into single block (default = FALSE)

comment - prefix for each line of results (default = '##')

dependson - chunk dependencies for caching (default = NULL)

echo - Display code in output document (default = TRUE)

engine - code language used in chunk (default = 'R')

error - Display error messages in doc (TRUE) or stop render when errors occur (FALSE) (default = FALSE)

eval - Run code in chunk (default = TRUE)

fig.align - 'left', 'right', or 'center' (default = 'default')

fig.cap - figure caption as character string (default = NULL)

fig.height, fig.width - Dimensions of plots in inches

highlight - highlight source code (default = TRUE)

include - Include chunk in doc after running (default = TRUE)

message - display code messages in document (default = TRUE)

results (default = 'markup')
'asis' - passthrough results
'hide' - do not display results
'hold' - put all results below all code

tidy - tidy code for display (default = FALSE)

warning - display code warnings in document (default = TRUE)

Options not listed above: R.options, aniopts, autodep, background, cache.comments, cache.lazy, cache.rebuild, cache.vars, dev, dev.args, dpi, engine.opts, engine.path, fig.asp, fig.env, fig.ext, fig.keep, fig.lp, fig.path, fig.pos, fig.process, fig.retina, fig.scap, fig.show, fig.showtext, fig.subcap, interval, out.extra, out.height, out.width, prompt, purl, ref.label, render, size, split, tidy.opts

Interactive Documents

Turn your report into an interactive Shiny document in 4 steps
1. Add runtime: shiny to the YAML header.
2. Call Shiny input functions to embed input objects.
3. Call Shiny render functions to embed reactive output.
4. Render w rmarkdown::run or click Run Document in RStudio IDE

```
---
output: html_document
runtime: shiny
---

```{r, echo = FALSE}
numericInput("n",
 "How many cars?", 5)

renderTable({
 head(cars, input$n)
})
```
```

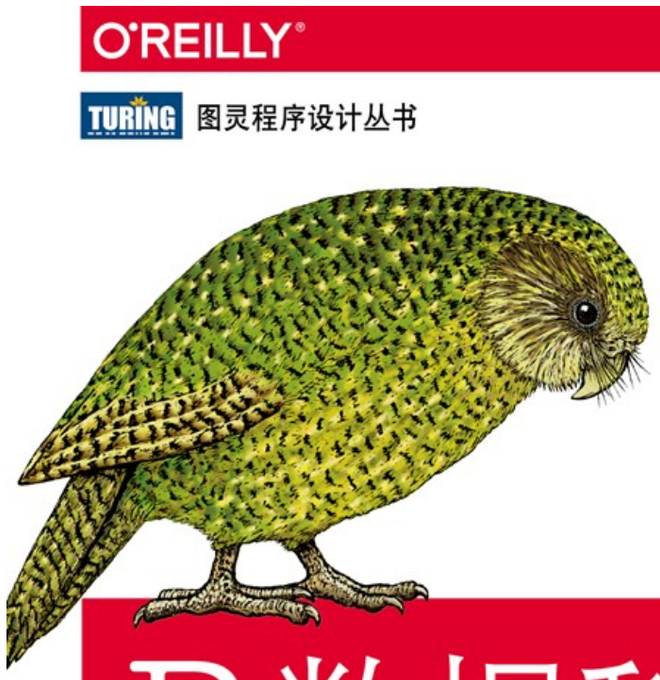


Embed a complete app into your document with shiny::**shinyAppDir()**

Publish on RStudio Connect, to share R Markdown documents securely, schedule automatic updates, and interact with parameters in real time.
www.rstudio.com/products/connect/

练习

练习 - 0040



O'REILLY®

TURING 图灵程序设计丛书

全彩印刷

R数据科学

R for Data Science

摒弃其他R语言工具书从头到尾讲统计的陋习
从实用的R包出发，带你重新认识R和数据科学

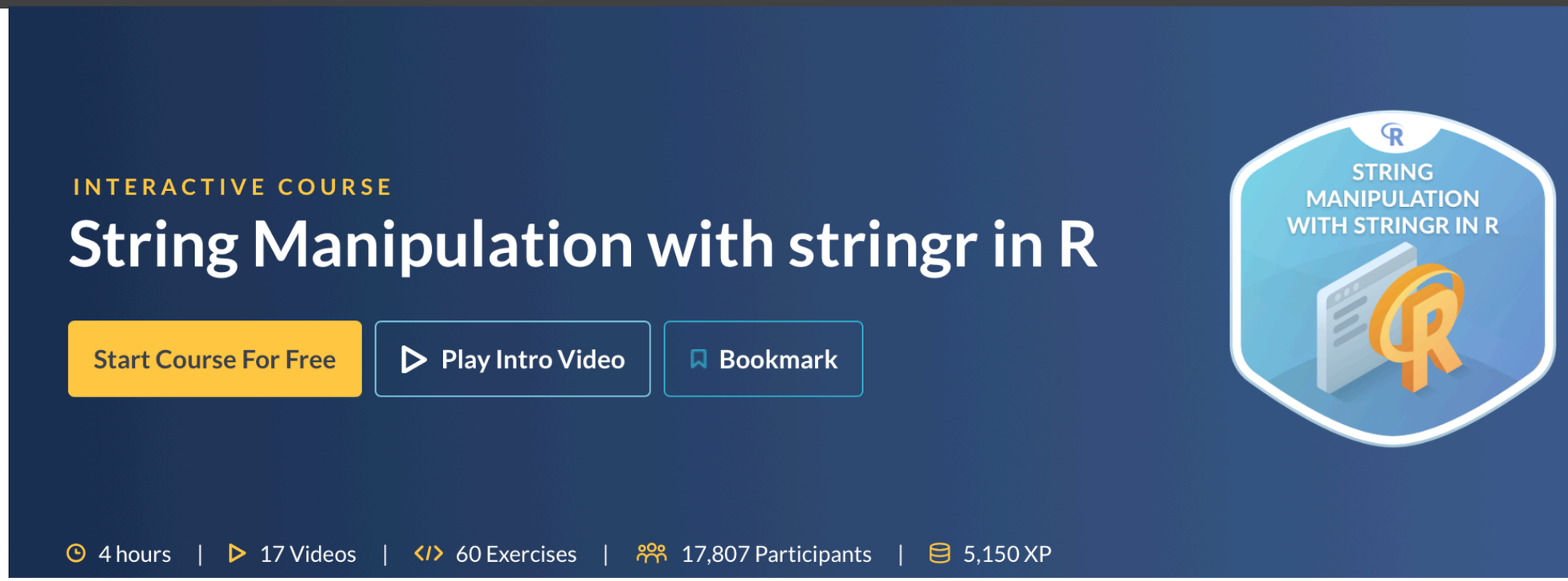
[新西兰] 哈德利·威克姆 [美] 加勒特·格罗勒芒德 著
陈光欣 译

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

第10章 第12章 第20章 第22章

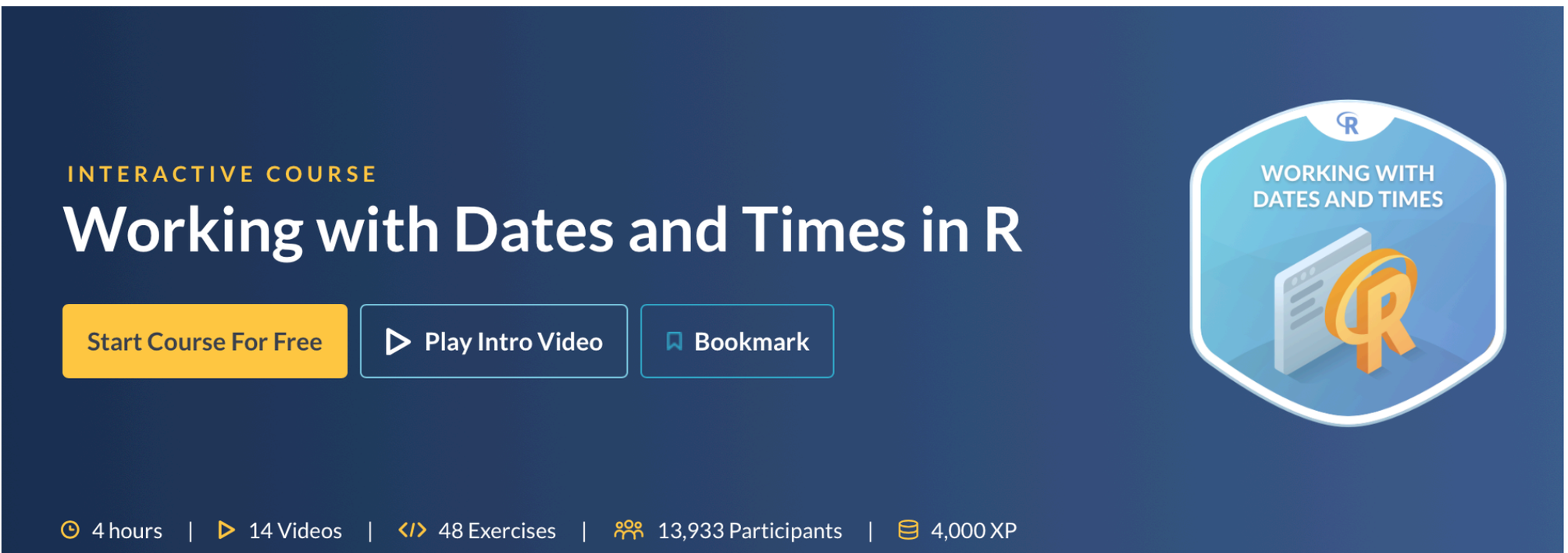
String&Time...

练习 - 0041



提交方式和以前一样！

https://www.datacamp.com/courses



练习 – 0042

- USArrsts数据集

```
> head(USArrests)
           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
Colorado      7.9     204       78 38.7
```

- 获取名字最长和最短的州名
- 含有元音的州名，州名中元音个数统计
- 州名中所有字母的个数统计

练习 - 0043

- 12306泄漏数据库，见@12306.txt

```
274667266@qq.com----6837605----郑一峰----332522198705040011----z6837605----15068860664----274667266@qq.com
zaistar@163.com----tianxia512----池善卿----350426197909063O1X----chitang520----18105013289----zaistar@163.com
weizhongjie55@163.com----wzj27713----卫忠杰----210602198711260513----wzj871126----18707734000----weizhongjie55@163.com
xujsh2004@yahoo.com.cn----19830307----许家圣----340103198303072554----xujsh2012----18225513108----xujsh2004@yahoo.com.cn
793925564@qq.com----793925564----李靖男----410183199307210015----lijingnan741----18024105681----793925564@qq.com
chenkan588@163.com----chengkang----陈侃----362326198306270039----chenkan588----18258288023----chenkan588@163.com
kangjie109@163.com----159648sl----康焕卉----430503198706130038----kangjie109----13716008430----kangjie109@163.com
a2135336@163.com----a2135336----池鹏----331081198601210014----cp165147----18888731462----a2135336@163.com
daqi1003@163.com----liudaqi----刘大奇----230103198509121352----daqi1003----15810596619----daqi1003@163.com
```

- 统计口令数量：仅有数字、仅有字母、包含特殊字符、字母＋数字
- 统计最常用的100个口令
- 分离出账号和口令有关系的信息，分离身份证和口令有关的信息
- 统计泄漏的年龄分布

注意：数据量较大，可以选择一个子集上完成

谢谢！

孙惠平

sunhp@ss.pku.edu.cn