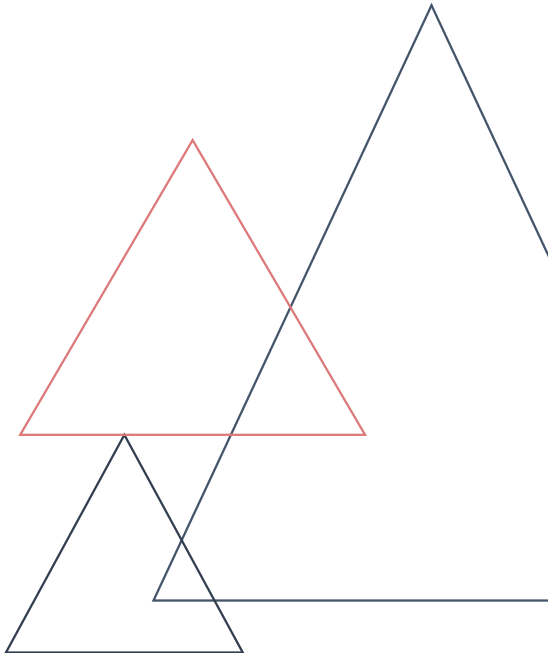
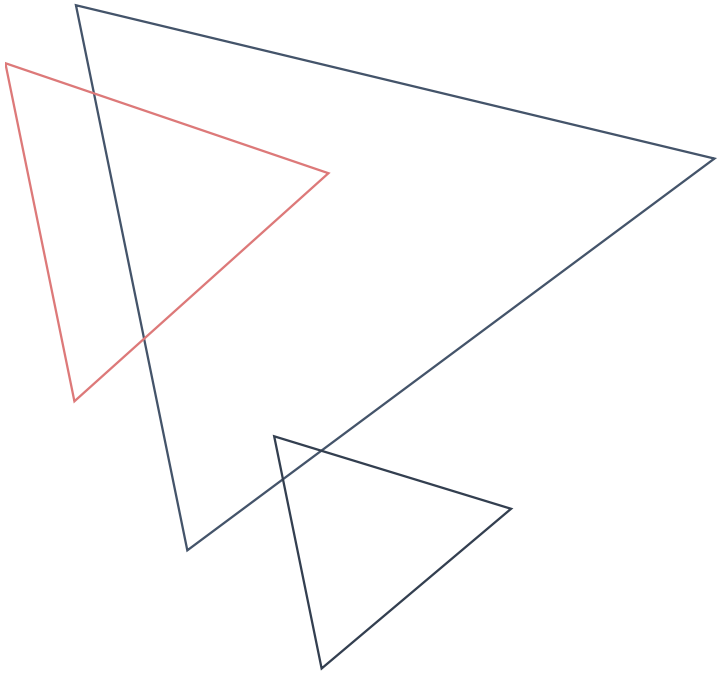
Stringr&Lubridate

数据分析工具实践

第一组
组长：张学思
组员：陈静、陈若愚、孙一辰

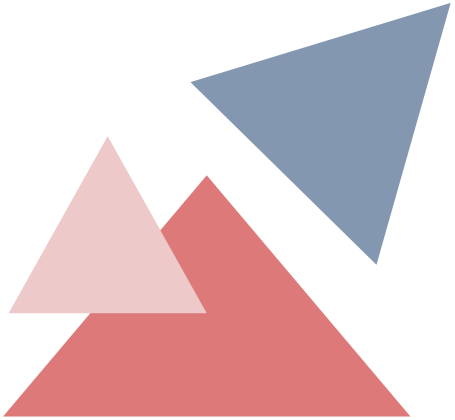




Stringr

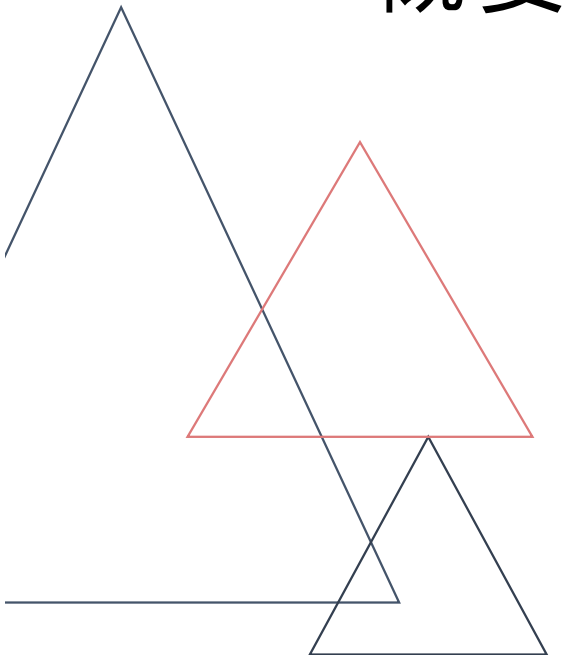
1 概要介绍

2 主要函数



PART ONE

概要介绍



字符处理 stringr

1 作用

R语言支持字符处理，内置了系列函数(grep、gsub等)，但系列函数定义混乱，对使用者极不方便。stringr包是专门用于字符处理的R包，函数定义简洁、使用方式统一，是使用率较高的R包。

2 函数

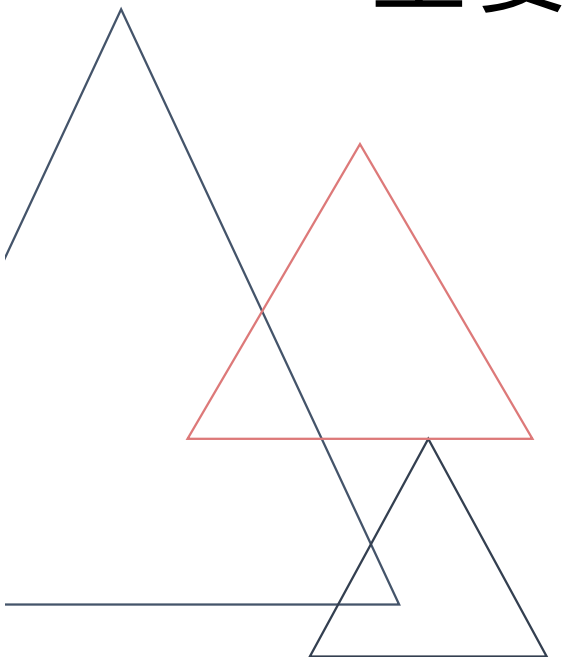
stringr包中的大部分函数具有统一风格的命名方式，以str_开头，正则表达式也完全适用该包。

3 安装

```
install.packages("stringr")
```

PART TWO

主要函数



字符串拼接——str_c

默认无向量分割符拼接

str_c("a","b")

"ab "

指定向量分隔符

str_c("a","b",sep = "_")

"a_b"

指定向量折叠符

str_c(c("a","b","c"),collapse = "_")

"a_b_c"

混合应用

str_c(c("a","b"),c("c","d"),sep = "/",collapse = "_")

"a/c_b/d"

字符计数——str_count

单个目标字符计数

```
str_count(string =
c("sql","json","java"),pattern = "s")
```

1 1 0

多个目标字符计数

```
str_count(string =
c("sql","json","java"),pattern = c("s","j","a "))
```

1 1 2

元字符查找计数（fixed包裹元字符）

```
str_count(string =
"a..b",pattern = fixed("."))
```

2

字符检查、复制、长度

字符检查
str_detect

```
str_detect(string =
c("sql","json","java"),pattern = "s")
```

TRUE TRUE FALSE

字符复制
str_dup

```
str_dup(string =
c("sql","json","java"),times = 2)
```

"sqlsql" "jsonjson" "javajava"

字符串长度
str_length

```
str_length(string = "banana")
```

6

字符提取——str_ extract、str_extract_all

提取第一个匹配到的字符

str_extract(string = "banana",pattern = "a")

"a"

提取所有匹配到的字符（返回列表）

str_extract_all(string = "banana",pattern = "a")

"a" "a" "a"

提取所有匹配到的字符（返回矩阵）

str_extract_all(string = "banana",pattern = "a",simplify = T)

"a" "a" "a"

字符串格式化、提取、匹配

字符串格式化
str_glue

```
# 定义全局变量
name <- "jack"
age <- 12

# 字符串格式化
str_glue("My name is
{name},","\nmy age is {age}.")

## My name is jack,
## my age is 12.
```

字符位置提取
str_locate和str_locate_all

```
# 返回第一个匹配到的字符的位置
str_locate(string = "banana",pattern = "a")
##      start end
## [1,]     2   2

# 返回所有匹配到的字符的位置
str_locate_all(string = "banana",pattern = "a")
## [[1]]
##      start end
## [1,]     2   2
## [2,]     4   4
## [3,]     6   6
```

字符匹配
str_match和str_match_all

```
# 返回第一个匹配到的字符（矩阵）
str_match(string = "banana",pattern = "a")
##      [,1]
## [1,] "a"

# 返回所有匹配到的字符（列表）
str_match_all(string = "banana",pattern = "a")
## [[1]]
##      [,1]
## [1,] "a"
## [2,] "a"
## [3,] "a"
```

字符串补齐、删除

字符补齐
str_pad

```
# 默认字符串左边补齐
str_pad(string = "jack",width = 6,pad = "S")
## [1] "SSjack"

# 字符串右边补齐
str_pad(string = "jack",width = 6,side = "right",pad = "S")
## [1] "jackSS"

# 字符串两边补齐
str_pad(string = "jack",width = 6,side = "both",pad = "S")
## [1] "SjackS"
```

字符删除
str_remove和str_remove_all

```
# 删除第一个匹配到的字符
str_remove(string = "banana",pattern = "a")
## [1] "bnana"

# 删除所有匹配到的字符
str_remove_all(string = "banana",pattern =
"a")
## [1] "bnn"
```

字符串替换、过滤

字符替换

str_replace、str_replace_all和str_replace_na

```
# 替换第一个匹配到的字符
str_replace(string = "banana",pattern = "a",replacement = "A")
## [1] "bAnana"

# 替换所有匹配到的字符
str_replace_all(string = "banana",pattern = "a",replacement =
"A")
## [1] "bAnAnA"

# NA替换成NA字符
str_replace_na(string = c("banana",NA))
## [1] "banana" "NA"
```

字符过滤

str_replace、str_replace_all和str_replace_na

- # 字符过滤（正向索引）

```
str_sub(string = "banana",start = 1,end = 3)
## [1] "ban"
```

- # 字符过滤（反向索引）

```
str_sub(string = "banana",start = -2,end = -1)
## [1] "na"
```

- # 字符过滤，并赋值

```
x <- "banana"
str_sub(string = x,start = 1,end = 1) <- "A"
## [1] "Aanana"
```

- # 字符串过滤（返回字符串）

```
str_subset(string = c("java","sql","python"),pattern = "^s")
## [1] "sql"
```

- # 字符串过滤（返回位置）

```
str_which(string = c("java","sql","python"),pattern = "^s")
## [1] 2
```

字符串排序、分割

字符排序
str_sort和str_order

```
# 字符向量升序排序，返回字符向量
str_sort(c("sql","json","python"))
## [1] "json"   "python" "sql"

# 字符向量降序排序，返回字符向量
str_sort(c("sql","json","python"),decreasing =
TRUE)
## [1] "sql"    "python" "json"

# 字符向量升序排序，返回索引向量
str_order(c("sql","json","pythn"))
## [1] 2 3 1
```

字符分割
str_split和str_split_fixed

```
# 字符分割，返回列表
str_split(string = "banana",pattern = "")
## [[1]]
## [1] "b" "a" "n" "a" "n" "a"

# 字符分割，返回矩阵
str_split(string = "banana",pattern = "",simplify = T)
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] "b"  "a"  "n"  "a"  "n"  "a"

# 字符分割，指定分割块数
str_split_fixed(string = "banana",pattern = "",n = 3)
##      [,1] [,2] [,3]
## [1,] "b"  "a"  "nana"
```

其他

删除字符串两边的空格
str_trim(string = " you are beautiful! ”
[1] "you are beautiful!"

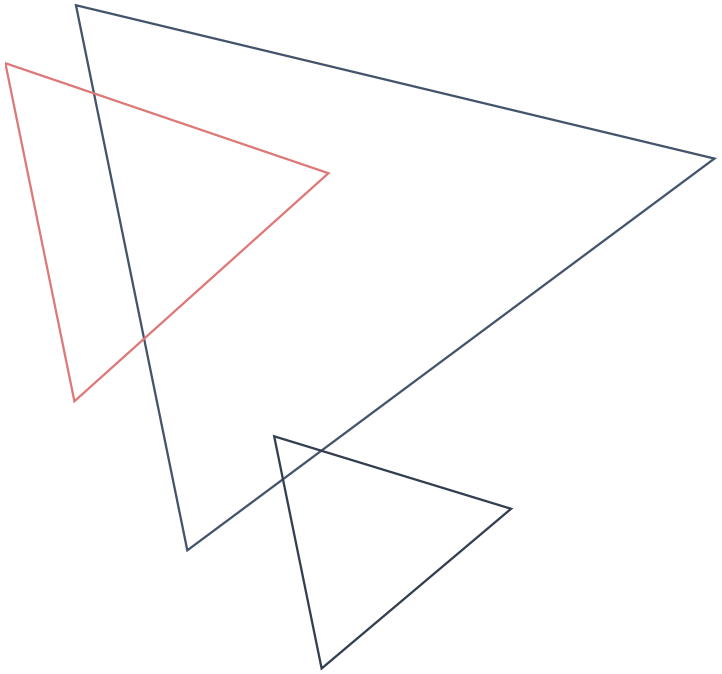
# 删除字符串中多余的空格
str_squish(string = " you are beautiful! ")
[1] "you are beautiful!"

# 字符转为小写
dog <- "The quick brown dog"
str_to_lower(dog)
[1] "the quick brown dog"

字符转为大写 str_to_upper(dog)
[1] "THE QUICK BROWN DOG"

# 字符转为标题
str_to_title(dog)
[1] "The Quick Brown Dog"

字符转为语句 str_to_sentence
[1] "The quick brown dog"

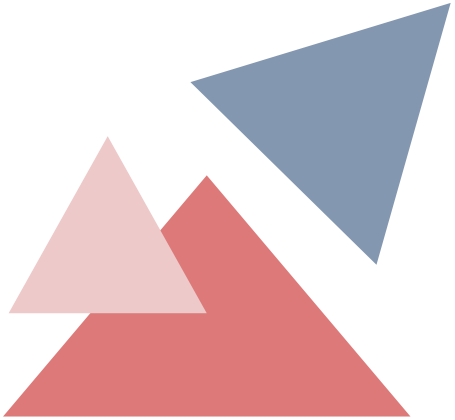


Lubridate

1 概要介绍

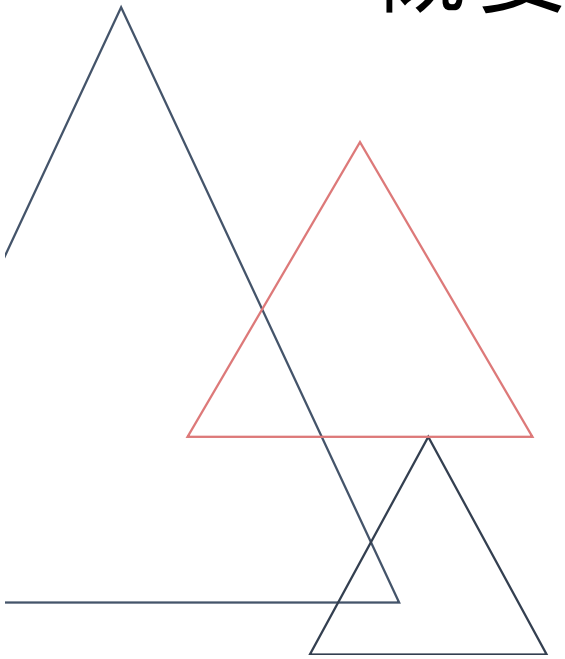
2 主要函数

3 实际运用



PART ONE

概要介绍



概要介绍

- **Hadley•Wickham：作为RStudio的Chief Scientist，他为R用户贡献了多个重量级的package（ggplot2、dplyr等等）。而今天介绍的这个lubridate包，也是由他所编写，专注于对日期时间数据（Date-time data）的处理。**

- **对于日期时间数据，R在基础包中提供了两种类型的时间数据：**
  - **一类是Date日期数据，它不包括时间和时区信息；**
  - **另一类是POSIXct/POSIXlt类型数据，其中包括了日期、时间和时区信息。**

（一般来讲，R语言中建立时序数据是通过字符型转化而来，但由于时序数据形式多样，而且R中存贮格式也是五花八门，例如Date/ts/xts/zoo/tis/fts等等。用户很容易被一系列的数据格式所迷惑，所以时序数据的转化和操作并不是非常方便。）

概要介绍

- 为此，我们引入第三方包lubridate，该包主要有两类函数：
  - 一类用于处理时点数据（time instants）
  - 另一类则用于处理时段数据（time spans）。

- 虽然这些基础功能R Base也能实现，但实现方式及其繁琐，通过下图的对比，我们可以看到同样是时间数据处理，lubridate包比R的基础包的操作是何等的简洁。

2. Motivation

To see how **lubridate** simplifies things, consider a common scenario. Given a character string, we would like to read it in as a date-time, extract the month, and change it to February (i.e. 2). On the left are the base R methods we'd use for these three tasks. On the right are the **lubridate** methods.

```
date <- as.POSIXct("01-01-2010",
 format = "%d-%m-%Y", tz = "UTC")

as.numeric(format(date, "%m")) # or
as.POSIXlt(date)$month + 1

date <- as.POSIXct(format(date,
 "%Y-2-%d"), tz = "UTC")
```

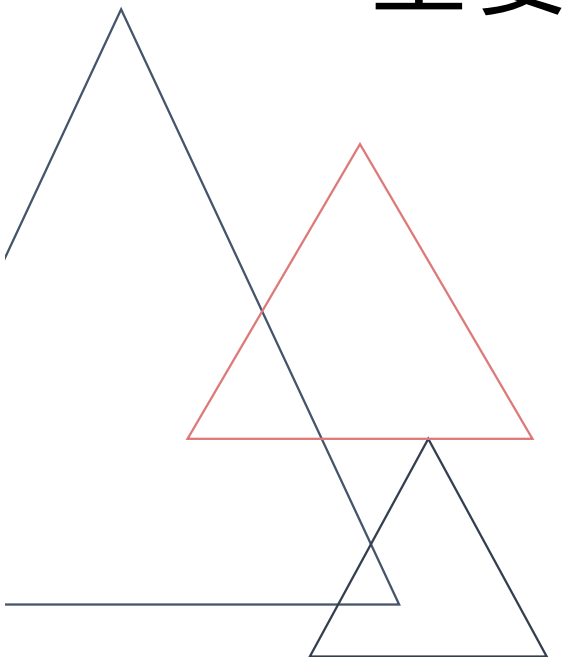
```
date <- dmy("01-01-2010")

month(date)

month(date) <- 2
```

PART TWO

主要函数



主要函数

一、解析日期与时间（Parsing dates and times）

- 首先，在使用lubridate包识别日期前，我们需要告诉它年（y）月（m）日（d）的排列顺序

Journal of Statistical Software

Order of elements in date-time	Parse function
year, month, day	`ymd()`
year, day, month	`ydm()`
month, day, year	`mdy()`
day, month, year	`dmy()`
hour, minute	`hm()`
hour, minute, second	`hms()`
year, month, day, hour, minute, second	`ymd_hms()`

主要函数

举例：

```
> ymd(20170629);myd(06201729);dmy(29062017)
[1] "2017-06-29"
[1] "2017-06-29"
[1] "2017-06-29"
```

增加：

- **具体时间的数据→加上小时（h）分钟（m）和秒（s）；**
- **具有特定时区的时间→tz选项**

```
> test_date <- ymd_hms("2017-06-29-12-01-30", tz = "Pacific/Auckland")
> test_date
[1] "2017-06-29 12:01:30 NZST"
```

注：lubriadate非常灵活，它可以“智能”的判断我们的输入格式，最好的得到标准的时间格式，甚至即使你的输入不完全，通过一个truncated选项，也可以识别不完整信息的日期输入格式。

主要函数

如果拿到的数据年月日是无序排列的？

parse_date_time()

- 它可以将格式各样的日期时间字符转换为日期时间类型的数据。该函数中有一个重要的参数，即orders，通过该参数指定可能的日期格式顺序，如年-月-日或月-日-年等顺序。

```
test_date <- c('20131113','120315','12/17/1996','09-01-01')
> parse_date_time(test_date,order = c('ymd','mdy','dmy','ymd'))

[1] "2013-11-13 UTC" "2012-03-15 UTC" "1996-12-17 UTC" "2009-01-01 UTC"
```

主要函数

二、设置与提取信息（Setting and Extracting information）

1. 精确提取

second()，minute()，hour()，day()，wday()，yday()，week()，month()，year()，tz()分别可以提取秒，分，小时，天，周的第几天，年的第几天，星期，月，年和时区的信息。

举例：

```
> test <- ymd_hms('2017/06/29/12/00/00')
> test
[1] "2017-06-29 12:00:00 UTC"
> second(test) <- 30
> test
[1] "2017-06-29 12:00:30 UTC"
```

```
> wday(test)
[1] 5
> wday(test,label = TRUE)
[1] Thurs
Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat
```

主要函数

二、设置与提取信息（Setting and Extracting information）

2. 模糊提取（取整）

模糊取整即截断函数，即将日期时间型数据取整到不同的单位，如年、季、月、日、时等。

四舍五入取整：round_date()
向下取整：floor_date()
向上取整：ceiling_date()

举例：

```
> test_date <- as.POSIXct("2017-06-29 12:34:59")
> round_date(test_date,'hour')
[1] "2017-06-29 13:00:00 CST"
> ceiling_date(test_date,'hour')
[1] "2017-06-29 13:00:00 CST"
> floor_date(test_date,'hour')
[1] "2017-06-29 12:00:00 CST"
```

主要函数

三、时区（Time Zones）

1. 显示同一个时间点在不同时区的时间（变换时区）
   - with_tz()
2. 结合某个时间点和给定时区，新建一个给定时区的时间点（固定时区）
   - force_tz()

举例：

```
> test_date <- ymd_hms("2017-06-29 09:00:00", tz = "Pacific/Auckland")
> with_tz(test_date,"America/New_York")
[1] "2017-06-28 17:00:00 EDT"
```

```
# 给定时间和时区，新建一个给定时区的对应时间点
> test_date <- ymd_hms("2017-06-29 12:00:00", tz = "America/Chicago")
> test_date
[1] "2017-06-29 12:00:00 CDT"
> test_date_1 <- force_tz(test_date,tz="Europe/London")
> test_date_1
[1] "2017-06-29 12:00:00 BST"
```

主要函数

四、时间间隔（Time Intervals）

```
> begin1 <- ymd_hms("20150903,12:00:00")
> end1 <- ymd_hms("20160804,12;30:00")
> begin2 <- ymd_hms("20151203,12:00:00")
> end2 <- ymd_hms("20160904,12;30:00")
```

```
> test_date_1 <- interval(begin1,end1)
> test_date_1
[1] 2015-09-03 12:00:00 UTC--2016-08-04 12:30:00 UTC
> test_date_2 <- interval(begin2,end2)
# 判断两段时间是否有重叠
> int_overlaps(test_date_1,test_date_2)
[1] TRUE
```

注：其他操作时间间隔的函数还包括：int_start，int_end，int_flip，int_shift，int_aligns，union，intersect和%within%等。

主要函数

五、日期时间的计算（Arithmetic with date times）

1. 时间跨度（durations和periods）
   - 时间间隔：特定的时间跨度（因为它绑定在特定时间点上）
   - 时间跨度：一般的时间跨度
     - durations
     - periods

```
# periods
> minutes(1)
[1] "1M 0S"
# durations[加前缀'd']
> dminutes(1)
[1] "60s"
```

主要函数

五、日期时间的计算（Arithmetic with date times）

1. 时间跨度（durations和periods）

- durations VS. periods
  - 为什么要这两个不同的类呢？因为时间线（timeline）并没有数字线（number line）那样可靠。
  - durations类：通常提供了更准确的运算结果。一个duration年总是等于365天。
  - periods类：随着时间线的波动而给出更理性的结果。

举例：这一特点在建立时钟时间（clock times）的模型时非常有用。比方说，durations遇到闰年时，结果就太死板，而periods给出的结果就灵活很多：

```
> leap_year(2016)
[1] TRUE
> ymd(20160101)+years(1)
[1] "2017-01-01"
> ymd(20160101)+dyears(1)
[1] "2016-12-31"
```

主要函数

五、日期时间的计算（Arithmetic with date times）

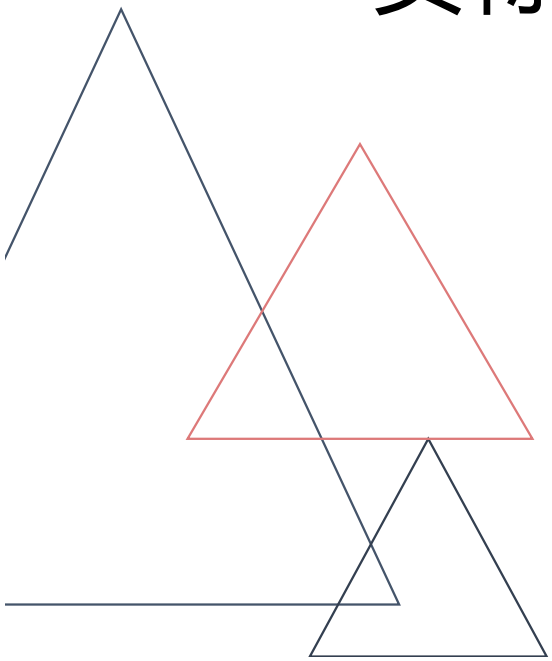
2. %m+%

- 在时间计算时，由于日期数据的特殊性，如果我们需要得到每个月的最后一天的日期数据，直接在某一个月的最后一天上加上月份很明显是错误的。为此我们引入%m+%函数：

```
> test_date_0 <- as.Date('2015-01-31')
> test_date_2 <- test_date_0 %m+% months(0:11)
> test_date_2
 [1] "2015-01-31" "2015-02-28" "2015-03-31" "2015-04-30" "2015-05-31" "2015-06-30"
 [7] "2015-07-31" "2015-08-31" "2015-09-30" "2015-10-31" "2015-11-30" "2015-12-31"
```

PART THREE

实际运用



实际运用

问题：计算2010年Thanksgiving时间

有些节日，如感恩节（美国）和阵亡将士纪念日（美国）并不发生在固定的日期。相反，他们是按照一个规则来庆祝的。例如，感恩节是在十一月的第四个星期四庆祝的。

为了得到感恩节是在2010年什么时间举行的，我们可以从2010的第一天开始计

```
R> date <- ymd("2010-01-01")
[1] "2010-01-01 UTC"
```

```
R> month(date) <- 11
[1] "2010-11-01 UTC"
```

加上10，或者直接将月份设定为11月。

```
R> wday(date, label = T, abbr = F)
[1] Monday
```

。

```
R> date <- date + days(3)
[1] "2010-11-04 UTC"
R> wday(date, label = T, abbr = F)
[1] Thursday
```

第一个星期四。

```
R> date + weeks(3)
[1] "2010-11-25 UTC"
```

星期到十一月的第四个星期四。

感谢观看

数据分析工具实践

第一组
组长：张学思
组员：陈静、陈若愚、孙一辰