# Data Analysis Using R: Chapter05

罗智超 (ROKIA.ORG)

## 1 通过本章你将学会

- 数据处理基本技能

  – 与变量有关创建变量、缺失值处理、类型转换、

  – 与记录有关

  排序、过滤记录

  – 多表操作横向合并、纵向合并、累加合并

- dplyr 的使用
- reshape2 的使用
- 日期
- 文本

## 2 创建变量的三种方法

```r
mydata$sumx <- mydata$x1 + mydata$x2
mydata$meanx <- (mydata$x1 + mydata$x2)/2
attach(mydata)
mydata$sumx <- x1 + x2
mydata$meanx <- (x1 + x2)/2
detach(mydata)
```

```
mydata <- transform(mydata, sumx = x1 + x2,
                    meanx = (x1 + x2)/2  )
```

## 3 连续变量分箱 (binning)

```
# Create 3 age categories from the age variable
attach(leadership)
leadership$agecat[age > 75] <- "Elder"
leadership$agecat[age > 45 & age <= 75] <- "Middle Aged"
leadership$agecat[age <= 45] <- "Young"
detach(leadership)

#library(smbinning)
```

## 4 变量赋值

## 5 变量重命名

```
names(leadership)[1]<-"ID"
#Redefine all the variables

names(leadership)<- c("testDate","country","gender","age","managerID","q1","q2",
                       "q3","q4", "q5")
# Maual rename
fix(leadership)
#Rename by programming
library(reshape)
leadership <- rename(leadership,c( manager="managerID",
                   date="testDate" ) )
```

# 6 缺失值

- NA: not avalable
- NaN: not a num
- 判断对象元素是否为缺失值

```
x1 <- c(1, 4, 3, NA, 7)
is.na(x1)

#[1] FALSE FALSE FALSE  TRUE FALSE
is.finite()
complete.cases()
```

- 计算包含缺失值的记录数
- 将包含缺失值的记录排除计算

```
# Obtion na.rm=TRUE remove missing values
# priors to caculations
# may be included in almost all the functions.
sum(x,na.rm=TRUE)
```

- 排序中的 NA 处理

```
sort(x1, na.last = TRUE)
```

- 删除包含缺失值得记录

```
newdata <- na.omit(leadership)
#na.omit roughly equivalent to x[!is.na(x)]
```

- 使用均值插值缺失值

```
x <- c(NA,0,2,0,2,NA,NA,NA,0,2)
ifelse(is.na(x), mean(x, na.rm = TRUE), x)

x[is.na(x)] <- mean(x, na.rm = TRUE)
```

- 使用 library mi() 进行插值

## 7 类型转换

| Test | Convert |
|---|---|
| is.numeric | as.numeric |
| is.character | as.character |
| is.vector | as.vector |
| is.matrix | as.matrix |
| is.data.frame | as.data.frame |

## 8 排序

```
# sorting examples using the leadership dataset
# sort by age
newdata <- leadership[order(-age),]

# sort by gender and age
newdata <- leadership[order(gender, age),]

#sort by gender (ascending) and age (descending)
newdata <-leadership[order(gender, -age),]
```

## 9 合并数据

## 10 纵向合并（关键字段）

```
# merge two dataframes by ID
total <- merge(dataframeA,dataframeB,by="ID")

# merge two dataframes by ID and Country
total <- merge(dataframeA,dataframeB,by=c("ID","Country"))

#by.x by.y by.all=TRUE
```

## 11 纵向合并（无关键字段）

```
#c means coloumn
total <- cbind(A, B)
```

## 12 横向合并

```
# merge two dataframes vertically
total <- rbind(dataframeA, dataframeB)
```

## 13 累计合并

```
data<-NULL
for (i in 1:10){
data<-rbind(data,down[[i]])
}
return(data)
```

## 14 变量选择

```
# select variables q1, q2, q3,
# q4, q5 from the leadership dataframe
# method 1
newdata <- leadership[, c(6:10)]

# method 2
myvars <- c("q1", "q2", "q3", "q4", "q5")
newdata <-leadership[myvars]

# method 3
myvars <- paste("q", 1:5, sep="")
newdata <- leadership[myvars]
```

## 15 删除变量

```
# exclude variables q3 and q4 three different ways  my

vars <- names(leadership) %in% c("q3", "q4")
newdata <- leadership[!myvars]

# exclude 8th and 10th variable
newdata <- leadership[c(-8,-9)]

# delete variables q3 and q4
leadership$q3 <- leadership$q4 <- NULL
```

## 16 选择记录

```
# first 5 observerations
newdata <- leadership[1:5,]

# based on variable values
newdata <- leadership[which(leadership$gender=="M"  & leadership$age > 30),]

# or
attach(leadership)
newdata <- leadership[which(gender=='M' & age > 30),]
detach(leadership)
```

## 17 提取数据集

```
# using subset function
newdata <- subset(leadership, age >= 35 | age < 24, select=c(q1, q2, q3, q4))

# using subset function (another example)
newdata <- subset(leadership, sex=="M" & age > 25, select=gender:q4)
```

## 18 随机抽样

```
# sample without replacement
mysample <- leadership[sample(1:nrow(leadership), 50, replace=FALSE),]
```

## 19 dplyr

## 20 显示数据

- Rstudio 最新版本在数据显示方面加强了很多，增加了过滤功能

- Rstudio console 中显示的数据依旧难看，所以有了 dplyr 的 tbl_df

```
library(dplyr)
library(hflights)
data(hflights)
head(hflights)

# convert to local data frame
flights <- tbl_df(hflights)

# printing only shows 10 rows and as many columns as can fit on your screen
flights
```

```
# you can specify that you want to see more rows
print(flights, n=20)

# convert to a normal data frame to see all of the columns
data.frame(head(flights))
```

## 21 匹配过滤

- 使用 base R 的方法需要一直重复输入数据框名字

```
# base R approach to view all flights on January 1
flights[flights$Month==1 & flights$DayofMonth==1, ]
```

```
# dplyr approach
# note: you can use comma or ampersand to represent AND condition
#note:dplyr generally does not preserve row names
filter(flights, Month==1, DayofMonth==1)

# use pipe for OR condition
filter(flights, UniqueCarrier=="AA" | UniqueCarrier=="UA")
```

```
# you can also use %in% operator
filter(flights, UniqueCarrier %in% c("AA", "UA"))
```

## 22 选择变量

```
# base R approach to select DepTime, ArrTime, and FlightNum columns
flights[, c("DepTime", "ArrTime", "FlightNum")]
```

```
# dplyr approach
select(flights, DepTime, ArrTime, FlightNum)

# use colon to select multiple contiguous columns, and use `contains` to match columns
# note: `starts_with`, `ends_with`, and `matches` (for regular expressions) can also be
select(flights, Year:DayofMonth, contains("Taxi"), contains("Delay"))
```

## 23 使用管道功能提升代码阅读

- 传统使用嵌套方法

```
# nesting method to select UniqueCarrier and DepDelay columns and filter for delays ove
filter(select(flights, UniqueCarrier, DepDelay), DepDelay > 60)
```

```
# chaining method
flights %>%
    select(UniqueCarrier, DepDelay) %>%
    filter(DepDelay > 60)
```

```
# create two vectors and calculate Euclidian distance between them
x1 <- 1:5; x2 <- 2:6
sqrt(sum((x1-x2)^2))
```

```r
# chaining method
(x1-x2)^2 %>% sum() %>% sqrt()
```

## 24 排序

```r
# base R approach to select UniqueCarrier and DepDelay columns and sort by DepDelay
flights[order(flights$DepDelay), c("UniqueCarrier", "DepDelay")]
```

```r
# dplyr approach
flights %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(DepDelay)
```

```r
# use `desc` for descending
flights %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(desc(DepDelay))
```

## 25 创建新变量

- Create new variables that are functions of existing variables

```r
# base R approach to create a new variable Speed (in mph)
flights$Speed <- flights$Distance / flights$AirTime*60
flights[, c("Distance", "AirTime", "Speed")]
```

```r
# dplyr approach (prints the new variable but does not store it)
flights %>%
    select(Distance, AirTime) %>%
    mutate(Speed = Distance/AirTime*60)
```

```
# store the new variable
flights <- flights %>% mutate(Speed = Distance/AirTime*60)
```

## 26 摘要分析

- 在分组分析中非常有价值

```
# base R approaches to calculate the average arrival delay to each destination
head(with(flights, tapply(ArrDelay, Dest, mean, na.rm=TRUE)))
head(aggregate(ArrDelay ~ Dest, flights, mean))
```

```
# dplyr approach: create a table grouped by Dest, and then summarise each group by taki
flights %>%
    group_by(Dest) %>%
    summarise(avg_delay = mean(ArrDelay, na.rm=TRUE))
```

- `summarise_each` 允许一次处理多个列变量
- Note: `mutate_each` is also available

```
# for each carrier, calculate the percentage of flights cancelled or diverted
flights %>%
    group_by(UniqueCarrier) %>%
    summarise_each(funs(mean), Cancelled, Diverted)

# for each carrier, calculate the minimum and maximum arrival and departure delays
flights %>%
    group_by(UniqueCarrier) %>%
    summarise_each(funs(min(., na.rm=TRUE), max(., na.rm=TRUE)), matches("Delay"))
#matches: 变量名包含 Delay
```

- Helper function `n()` counts the number of rows in a group
- Helper function `n_distinct(vector)` counts the number of unique items in that vector

```
# for each day of the year, count the total number of flights and sort in descending or
flights %>%
    group_by(Month, DayofMonth) %>%
    summarise(flight_count = n()) %>%
    arrange(desc(flight_count))

# rewrite more simply with the `tally` function
flights %>%
    group_by(Month, DayofMonth) %>%
    tally(sort = TRUE)

# for each destination, count the total number of flights and the number of distinct pl
flights %>%
    group_by(Dest) %>%
    summarise(flight_count = n(), plane_count = n_distinct(TailNum))
```

- Grouping can sometimes be useful without summarising

```
# for each destination, show the number of cancelled and not cancelled flights
flights %>%
    group_by(Dest) %>%
    select(Cancelled) %>%
    table() %>%
    head()
```

## 27 Window function

- Aggregation function (like `mean`) takes n inputs and returns 1 value
- Window function takes n inputs and returns n values
- 包括排序函数 (row_number(), min_rank (RANK in SQL), dense_rank(), cume_dist(), percent_rank(), and ntile()).

- 位移函数 offset (`lead` and `lag`),
- 累加函数 cumulative aggregate (cumsum(), cummin(), cummax() (from base R), and cumall(), cumany(), and cummean() (from dplyr)).
- 移动汇总函数 rolling aggregate
- 再循环汇总函数 Recycled aggregate

```
# for each carrier, calculate which two days of the year they had their longest departu
# note: smallest (not largest) value is ranked as 1, so you have to use `desc` to rank
# 取每个分组的排名最后两名的数据
flights %>%
    group_by(UniqueCarrier) %>%
    select(Month, DayofMonth, DepDelay) %>%
    filter(min_rank(desc(DepDelay)) <= 2) %>%
    arrange(UniqueCarrier, desc(DepDelay))
```

```
# rewrite more simply with the `top_n` function
# 取每个分组的排名前两名的数据
flights %>%
    group_by(UniqueCarrier) %>%
    select(Month, DayofMonth, DepDelay) %>%
    top_n(2) %>%
    arrange(UniqueCarrier, desc(DepDelay))

# for each month, calculate the number of flights and the change from the previous mont
flights %>%
    group_by(Month) %>%
    summarise(flight_count = n()) %>%
    mutate(change = flight_count - lag(flight_count))

# rewrite more simply with the `tally` function
flights %>%
    group_by(Month) %>%
    tally() %>%
    mutate(change = n - lag(n))
```

## 27.1 Other Useful Convenience Functions

```
# randomly sample a fixed number of rows, without replacement
flights %>% sample_n(5)

# randomly sample a fraction of rows, with replacement
flights %>% sample_frac(0.25, replace=TRUE)

# base R approach to view the structure of an object
str(flights)

# dplyr approach: better formatting, and adapts to your screen width
glimpse(flights)
```

# 28 连接数据库

- dplyr can connect to a database as if the data was loaded into a data frame

-Use the same syntax for local data frames and databases

- Only generates SELECT statements
- Currently supports SQLite, PostgreSQL/Redshift, MySQL/MariaDB, BigQuery, MonetDB
- Example below is based upon an SQLite database containing the hflights data
- Instructions for creating this database are in the databases vignette

```r
# connect to an SQLite database containing the hflights data
my_db <- src_sqlite("my_db.sqlite3")

# connect to the "hflights" table in that database
flights_tbl <- tbl(my_db, "hflights")

# example query with our data frame
flights %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(desc(DepDelay))

# identical query using the database
flights_tbl %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(desc(DepDelay))
```

- You can write the SQL commands yourself
- dplyr can tell you the SQL it plans to run and the query execution plan

```r
# send SQL commands to the database
tbl(my_db, sql("SELECT * FROM hflights LIMIT 100"))

# ask dplyr for the SQL commands
flights_tbl %>%
    select(UniqueCarrier, DepDelay) %>%
    arrange(desc(DepDelay)) %>%
    explain()
```

# 29 0.3 0.4 版本新增功能

# 30 选择变量

```
# besides just using select() to pick columns...
flights %>% select(carrier, flight)

# ...you can use the minus sign to hide columns
flights %>% select(-month, -day)
```

```
# hide a range of columns
flights %>% select(-(dep_time:arr_delay))

# hide any column with a matching name
flights %>% select(-contains("time"))
```

```
# pick columns using a character vector of column names
cols <- c("carrier", "flight", "tailnum")
flights %>% select(one_of(cols))
```

```
# select() can be used to rename columns, though all columns not mentioned are dropped
flights %>% select(tail = tailnum)

# rename() does the same thing, except all columns not mentioned are kept
flights %>% rename(tail = tailnum)
```

# 31 选择记录：filter, between, slice, sample_n, top_n, distinct

```
# filter() supports the use of multiple conditions
flights %>% filter(dep_time >= 600, dep_time <= 605)
```

```
# between() is a concise alternative for determing if numeric values fall in a range
flights %>% filter(between(dep_time, 600, 605))

# side note: is.na() can also be useful when filtering
flights %>% filter(!is.na(dep_time))
```

```
# slice() filters rows by position
flights %>% slice(1000:1005)

# keep the first three rows within each group
flights %>% group_by(month, day) %>% slice(1:3)

# sample three rows from each group
flights %>% group_by(month, day) %>% sample_n(3)

# keep three rows from each group with the top dep_delay
flights %>% group_by(month, day) %>% top_n(3, dep_delay)

# also sort by dep_delay within each group
flights %>% group_by(month, day) %>% top_n(3, dep_delay) %>% arrange(desc(dep_delay))
```

```
# unique rows can be identified using unique() from base R
flights %>% select(origin, dest) %>% unique()
```

```
# dplyr provides an alternative that is more "efficient"
flights %>% select(origin, dest) %>% distinct()

# side note: when chaining, you don't have to include the parentheses if there are no a
flights %>% select(origin, dest) %>% distinct
```

## 32 创建变量: mutate, transmute, add_rownames

```
# mutate() creates a new variable (and keeps all existing variables)
flights %>% mutate(speed = distance/air_time*60)

# transmute() only keeps the new variables
flights %>% transmute(speed = distance/air_time*60)
```

```
# example data frame with row names
mtcars %>% head()

# add_rownames() turns row names into an explicit variable
mtcars %>% add_rownames("model") %>% head()

# side note: dplyr no longer prints row names (ever) for local data frames
mtcars %>% tbl_df()
```

## 33 分组: summarise, tally, count, group_size, n_groups, ungroup

```
# summarise() can be used to count the number of rows in each group
flights %>% group_by(month) %>% summarise(cnt = n())
```

```
# tally() and count() can do this more concisely
flights %>% group_by(month) %>% tally()
flights %>% count(month)
```

```
# you can sort by the count
flights %>% group_by(month) %>% summarise(cnt = n()) %>% arrange(desc(cnt))
```

```
# tally() and count() have a sort parameter for this purpose
flights %>% group_by(month) %>% tally(sort=TRUE)
flights %>% count(month, sort=TRUE)
```

```
# you can sum over a specific variable instead of simply counting rows
flights %>% group_by(month) %>% summarise(dist = sum(distance))
```

```
# tally() and count() have a wt parameter for this purpose
flights %>% group_by(month) %>% tally(wt = distance)
flights %>% count(month, wt = distance)
```

```
# group_size() returns the counts as a vector
flights %>% group_by(month) %>% group_size()

# n_groups() simply reports the number of groups
flights %>% group_by(month) %>% n_groups()
```

```
# group by two variables, summarise, arrange (output is possibly confusing)
flights %>% group_by(month, day) %>% summarise(cnt = n()) %>% arrange(desc(cnt)) %>% pr

# ungroup() before arranging to arrange across all groups
flights %>% group_by(month, day) %>% summarise(cnt = n()) %>% ungroup() %>% arrange(des
```

# 34 创建 data_frame

`data_frame()` is a better way than `data.frame()` for creating data frames. Benefits of `data_frame()`:

- You can use previously defined columns to compute new columns.
- It never coerces column types.
- It never munges column names.
- It never adds row names.

- It only recycles length 1 input.
- It returns a local data frame (a tbl_df).

```r
# data_frame() example
data_frame(a = 1:6, b = a*2, c = 'string', 'd+e' = 1) %>% glimpse()

# data.frame() example
data.frame(a = 1:6, c = 'string', 'd+e' = 1) %>% glimpse()
```

# 35 合并数据集: left_join, right_join, inner_join, full_join, semi_join, anti_join

```r
# create two simple data frames
(a <- data_frame(color = c("green","yellow","red"), num = 1:3))
(b <- data_frame(color = c("green","yellow","pink"), size = c("S","M","L")))

# only include observations found in both "a" and "b" (automatically joins on variables
inner_join(a, b)

# include observations found in either "a" or "b"
full_join(a, b)

# include all observations found in "a"
left_join(a, b)

# include all observations found in "b"
right_join(a, b)

# right_join(a, b) is identical to left_join(b, a) except for column ordering
left_join(b, a)
```

```r
# filter "a" to only show observations that match "b"
semi_join(a, b)

# filter "a" to only show observations that don't match "b"
anti_join(a, b)
```

```r
# sometimes matching variables don't have identical names
b <- b %>% rename(col = color)

# specify that the join should occur by matching "color" in "a" with "col" in "b"
inner_join(a, b, by=c("color" = "col"))
```

## 36 输出预览: print, View

```r
# specify that you want to see more rows
flights %>% print(n = 15)
```

```r
# specify that you want to see ALL rows (don't run this!)
flights %>% print(n = Inf)
```

```r
# specify that you want to see all columns
flights %>% print(width = Inf)
```

```r
# show up to 1000 rows and all columns
flights %>% View()

# set option to see all columns and fewer rows
options(dplyr.width = Inf, dplyr.print_min = 6)

# reset options (or just close R)
options(dplyr.width = NULL, dplyr.print_min = 10)
```

## 37 相关资源

- Official dplyr reference manual and vignettes on CRAN: vignettes are well-written and cover many aspects of dplyr
- July 2014 webinar about dplyr (and ggvis) by Hadley Wickham and related slides/code: mostly conceptual, with a bit of code
- dplyr tutorial by Hadley Wickham at the useR! 2014 conference: excellent, in-depth tutorial with lots of example code (Dropbox link includes slides, code files, and data files)
- dplyr GitHub repo and list of releases
- Release announcements for version 0.3 and version 0.4
- dplyr reference manual and vignettes
- Two-table vignette covering joins and set operations
- RStudio's Data Wrangling Cheat Sheet for dplyr and tidyr

## 38 Reshape2

## 39 melt()

```r
# id include variables to be kept ,
# measured include variables need to transpose
library(reshape2)
melt(dataframe,id=c("v1","v2"),
     measured=c("v3","v4"))

melt(data, id.vars, measure.vars,
    variable.name = "variable", ..., na.rm = FALSE,
    value.name = "value")
```

```
head(airquality)
aqm<-head(melt(airquality, id=c("Month", "Day")))
head(aqm)
melt(airquality,id=c("month","day"))
names(airquality) <- tolower(names(airquality))
```

## 40 dcast()

```
dcast(dataframe,idvariable~variable,
      aggregatefun,subset=.())
```

- Use acast or dcast depending on whether you want vector/matrix/array output or data frame output.
- Data frames can have at most two dimensions.

## 41 日期处理

## 42 日期时间类

- 系统内嵌三种类：Date, POSIXct, POSIXlt.
- Date

This is the class to use if you have only dates, but no times, in your data.

```
# 创建日期
dt1 <- as.Date("2012-07-22")
dt2 <- as.Date("04/20/2011", format = "%m/%d/%Y")
dt3 <- as.Date("October 6, 2010", format = "%B %d, %Y")
# 查看所有格式:
```

```
`?`(strptime)

# 日期计算

# 计算间隔
dt1 - dt2
# 加减天数
dt2 + 10
dt2 - 10

# 创建日期向量

six.weeks <- seq(dt1, length = 6, by = "week")
six.weeks <- seq(dt1, length = 6, by = "2 weeks")
six.weeks <- seq(dt1, length = 6, by = 14)

#diff
three.dates <- as.Date(c("2010-07-22", "2011-04-20", "2012-10-06"))
diff(three.dates)
difftime(dt1, dt2, units = "weeks")

# 日期-数值转换
a<-as.Date("2014-01-03")
b<-as.integer(a)
c<-as.Date(b,origin="1960-01-01")
```

- POSIXct

如果要包含时间信息，就要使用 POSIXct 类

```
# 创建 POSIXct 类
tm1 <- as.POSIXct("2013-07-24 23:55:26")
tm2 <- as.POSIXct("25072013 08:32:07", format = "%d%m%Y %H:%M:%S")
```

```r
# 设定时区
tm3 <- as.POSIXct("2010-12-01 11:42:03", tz = "GMT")
# 时间计算
# 加减 30 秒
tm1 + 30
tm1 - 30
# 时间比较
tm2 - tm1

# 获取当前时间（默认是 POSIXct 格式）
```

- POSIXlt

该类以 List 类型存储日期时间，更方便提取元素,“ct” 代表 calendar time 日历时间，lt 代表 local time.

```r
# 创建时间
tm1.lt <- as.POSIXlt("2013-07-24 23:55:26")
unclass(tm1.lt)
unlist(tm1.lt)
# 提取元素

tm1.lt$sec
tm1.lt$wday

# 截取长度（日期、分钟）
trunc(tm1.lt, "days")
trunc(tm1.lt, "mins")
```

## 43 日期时间包

- chron

如果不需要考虑时区和 daylight savings time，可以考虑使用 chron 类，需要调用 library(chron)

```
library(chron)
tm1.c <- as.chron("2013-07-24 23:55:26")
tm2.c <- as.chron("07/25/13 08:32:07", "%m/%d/%y %H:%M:%S")
dates(tm1.c)
tm2.c > tm1.c
tm1.c + 10
tm2.c - tm1.c
difftime(tm2.c, tm1.c, units = "hours")
```

- ts
- zoo

zoo 是一个 R 语言类库，zoo 类库中定义了一个名为 zoo 的 S3 类型对象，用于描述规则的和不规则的有序的时间序列数据。zoo 对象是一个独立的对象，包括索引、日期、时间，只依赖于基础的 R 环境，zoo 是时间序列的基础，也是股票分析的基础。

```
# 构建一个 zoo 对象，以时间为索引
library(zoo)
x.Date <- as.Date("2003-02-01") + c(1, 3, 7, 9, 14) - 1
x.Date
class(x.Date)
x1 <- zoo(rnorm(5), x.Date)
class(x1)
x.Time<-as.POSIXct("2013-07-24 23:55:26")+c(1, 3, 7, 9, 14) - 1
x2<-zoo(rnorm(5),x.Time)
class(x2)
plot(x2)
# 修改 zoo 的数据部分
coredata(x2)
```

```r
# 修改 zoo 的索引部分
index(x2)
# 按时间过滤 window.zoo
window(x, start = as.Date(""), end = as.Date(""))
# 合并多个 zoo
merge(x1, x2, all = FALSE)

# 数据滚动处理 rollapply
z <- zoo(11:15, as.Date(31:35))
rollapply(z, 2, mean)
```

- xts

*zoo 作为时间序列的基础库，是面向通用的设计，可以用来定义股票数据，也可以分析天气数据。但由于业务行为的不同，我们需要更多的辅助函数，来帮助我们更高效的完成任务。xts 扩展了 zoo，提供更多的数据处理和数据变换的函数。

*xts 对象是金融时间序列的标准。quantmod getSymbols() 的提取结果就是一个 xts 对象。wind 的数据提取结果是一个 list, 其中的 data 部分包括了日期时间信息和数据，需要转换成 xts 对象。

## 44 时间序列作图包的简单历史

- `Package Date`

  *1 ts 1999-08-27* 2 lattice/grid 2002-04-29 *3 zoo 2004-10-08* 4 zoo/lattice 2006-07-06 *5 PeformanceAnalytics 2007-02-02* 6 ggplot2 2007-06-10 *7 quantmod/TTR 2007-10-07* 8 xts 2008-02-17 *9 timeSeries 2009-05-17* 10 xtsExtra 2012-05-30 *11 rCharts 2013-04-10

require(devtools) install_github('rCharts', 'ramnathv')

# 45 日期时间处理包

- lubridate

* 这个包在 POSIXct 的基础上增加了很多日期时间操作函数

```
# 创建日期时间
# 默认使用 UTC 时区
tm1.lub <- ymd_hms("2013-07-24 23:55:26")
tm2.lub <- mdy_hm("07/25/13 08:32")
tm3.lub <- ydm_hm("2013-25-07 4:00am")
tm4.lub <- dmy("26072013")

# 提取日期时间元素

year(tm1.lub)
week(tm1.lub)
wday(tm1.lub, label = TRUE)
hour(tm1.lub)
tz(tm1.lub)
second(tm2.lub) <- 7
tm2.lub
```

*Lubridate 区分四种对象: instants, intervals, durations, and periods.

*instants 是一个时点

*Intervals, durations, and periods 是三种区间表示方法

```
# 日期时间都是 instants 对象
is.instant(tm1.lub)
# 日期时间取整
round_date(tm1.lub, "minute")
round_date(tm1.lub, "day")
# 查看在不同时区的时间
```

```
with_tz(tm1.lub, "America/Los_Angeles")
# 改变时区
force_tz(tm1.lub, "America/Los_Angeles")

# 日期时间区间
in.bed <- as.interval(tm1.lub, tm2.lub)

# 判断某时间是否在某区间内
tm3.lub %within% in.bed

# 判断时间范围是否重叠
daylight <- as.interval(ymd_hm("2013-07-25 06:03"), ymd_hm("2013-07-25 20:23"))
int_overlaps(in.bed, daylight)

#duration 没有设定开始结尾时间，只是设定了一个持续时间范围，以秒为最小单位
# 创建 duration
ten.minutes <- dminutes(10)
five.days <- ddays(5)
one.year <- dyears(1)
as.duration(in.bed)


#period 没有设定开始结尾时间，只是设定了一个持续时间范围，与 duration 表现形式不一样

three.weeks <- weeks(3)
four.hours <- hours(4)
sabbatical <- months(6) + days(12)
```

## 46 日期时间其他资源链接

-date and time tutorials for R: *http://www.stat.berkeley.edu/classes/s133/dates.html* http://science.nature.nps.gov/

im/datamgmt/statistics/r/fundamentals/dates.cfm *http://en.wikibooks.org/wiki/R_Programming/Times_and_Dates

- lubridate: *http://www.jstatsoft.org/v40/i03/paper
- time zone and daylight saving time info: *http://www.timeanddate.com/* http://en.wikipedia.org/wiki/List_of_tz_database_time_zones *http://www.twinsun.com/tz/tz-link.htm* Also see the R help file at ?Sys.timezone

## 47 日期时间小结

- 如果只有日期，就用 Date
- 如果有时间，通常 POSIXct 是最好选择
- POSIXlt 提取日期时间元素最方便
- 如果不需要处理时区和 daylight savings time(类似夏令时),chron 也很方便

## 48 文本处理