Processing Dates and Times with Open Source R

Revolution Analytics

1. Dates in R
2. Date class
3. POSIXct and POSIXlt
4. Additional Packages
5. Summary

Overview

In this module we will discuss some of the essential date-related concepts in R and how to use R's date classes and functions.

Outline

1 Dates in R

2 Date class

3 POSIXct and POSIXlt

4 Additional Packages

5 Summary

Date classes in R

- There are three built-in date and time classes available in R
  1. `Date` class
  2. `POSIXct` class
  3. `POSIXlt` class
- The input and formatting of the dates may depend on the objects' class

Date classes in R

Date Represents the number of **days** since the beginning of 1970 (no time information).

POSIXct Represents the number of **seconds** since the beginning of 1970.

POSIXlt Representats date and time information in a named `list`.

Outline

1 Dates in R

2 Date class

3 POSIXct and POSIXlt

4 Additional Packages

5 Summary

Simple manipulation from strings

- `Date` classes are usually created from strings of characters using the standard coercion function.

Simple manipulation from strings

```r
SampleDate1 <- as.Date("2010-07-31")
SampleDate2 <- as.Date("08 / 01 / 2012", format = "%m / %d / %Y")
```

Format symbols

- We can check the format symbols like %Y , %m, %B, etc. to control input:

```
help(as.Date)
help(strptime)
```

Exercise

Look at the help for `as.Date()`, and describe the default `format` that the function uses to convert into a `Date` object.

Date calculations

- Dates can be used in Calculations intuitively.

```
diff <- SampleDate2 - SampleDate1
diffWeeks <- difftime(SampleDate2, SampleDate1, units = "weeks")
```

Reading Data

Let’s read in a vector of dates that we can use to illustrate some additional features.

```
dataPath <- "../data"
Dates101Csv <- file.path(dataPath, "101Dates.csv")
Dates101 <- read.csv(Dates101Csv, header = TRUE, as.is = TRUE)$Date
Dates101 <- as.Date(Dates101, format = "%m/%d/%Y")
head(Dates101)
```

```
## [1] "2036-10-19" "1992-09-09" "1988-09-14" "2012-10-27" "1935-12-03"
## [6] "2024-11-24"
```

Exercise

Now we have a vector of dates. Can extract the earliest and the most recent dates in the vector?

Date summary()

We can also perform summaries on dates

```
summary(Dates101)
```

```
##         Min.      1st Qu.       Median         Mean      3rd Qu.
## "1900-06-14" "1947-07-18" "1983-04-01" "1980-07-30" "2015-08-04"
##         Max.
## "2049-12-01"
```

Date calculations and sequences

- Intervals in a sequence or vector of dates can be obtained by lagged differences

```
dateDiff <- diff(Dates101)
head(dateDiff)
```

```
## Time differences in days
## [1] -16111  -1456   8809 -28088  32499 -32615
```

Date calculations and sequences

- Generating a sequence of dates may be necessary (say for time series analysis later)

```
DateSequence1 <- seq(SampleDate2, length = 50, by = "week")
DateSequence2 <- seq(SampleDate2, length = 50, by = "year")
DateSequence3 <- seq(SampleDate2, length = 50, by = "2 years")
```

Integer representation internally in R

- Dates are internally represented as integers inside the system
- Specifically, the number of days since January 1, 1970 (see help for `Date`)

```
unclass(SampleDate1)
```

```
## [1] 14821
```

Questions?

Are there any questions about the `Date` class?

Outline

1. Dates in R
2. Date class
3. POSIXct and POSIXlt
4. Additional Packages
5. Summary

POSIXct manipulation

- The `POSIXct` class is applicable when dealing with both a date and time entity
- Time zones can be included

```r
SampleDate1 <- as.POSIXct("2014-02-20 23:50:26")
SampleDate2 <- as.POSIXct("02 202012 12:15:26", format = "%m %d%Y %H:%M:%S")
SampleDate3 <- as.POSIXct("05112000 02:16:10", format = "%m%d%Y %H:%M:%S",
  tz = "EST")
```

Date Calculations

- Computations and comparisons are also possible for POSIXct objects

```
SampleDate2 <= SampleDate3
```

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

```
SampleDate1 + 100
```

```
## [1] "2014-02-20 23:52:06 EST"
```

```
diff <- SampleDate1 - SampleDate2
```

System Time Stamps

```
class(Sys.time())
```

```
## [1] "POSIXct" "POSIXt"
```

```
class(Sys.Date())
```

```
## [1] "Date"
```

difftime and POSIXct

```
difftime(SampleDate1, SampleDate2, tz = "GMT", units = "secs")
```

```
## Time difference of 63200100 secs
```

POSIXlt

- We can think of the inherent difference between `POSIXct` and `POSIXlt` in the following manner:

POSIXct	“ct” refers to “calendar time”
POSIXlt	“lt” refers to “local time” or broken-down time. In R, it is broken down into a named list.

We also have to keep in mind that the reference point for the POSIX standard is 12:00 AM GMT, 01 Jan 1970

POSIXlt

The `POSIXlt` class represents the components of date-time objects as a list with named elements:

```
* sec
* min
* hour
* mday
* etc.
```

POSIXlt Examples

```
as.POSIXlt("1999-02-28 11:13:25")
```

```
## [1] "1999-02-28 11:13:25 EST"
```

```
SampleDate2 <- as.POSIXlt("08 / 01 / 2012 >>> 10>10>10", format = "%m / %d / %Y >>> %H>%M>%S")
```

POSIXlt

- A POSIXlt object is composed of several date-time objects:

```
names(unclass(SampleDate2))
```

```
##  [1] "sec"    "min"    "hour"   "mday"   "mon"    "year"   "wday"
##  [8] "yday"   "isdst"  "zone"   "gmtoff"
```

```
SampleDate2$sec
```

```
## [1] 10
```

Exercise

How are most of the fields indexed? Specifically, if you extract the `mon` from a `POSIXlt` object, what month name would a value of 2 correspond to?

trunc()

trunc() has methods for dates and times.

```
trunc(SampleDate3, "day")
```

```
## [1] "2000-05-11 EST"
```

```
trunc(SampleDate3, "mins")
```

```
## [1] "2000-05-11 02:16:00 EST"
```

Summary

We have reviewed the `base` classes that represent dates and times in R, and we discussed how we might operate on each of them.

Any questions?

Outline

1. Dates in R
2. Date class
3. POSIXct and POSIXlt
4. Additional Packages
5. Summary

Additional Packages

There are a number of additional packages that facilitate the processing of dates

We will mention two:

- `chron`
- `lubridate`

chron

chron is a good option if we don't need to deal with timezones

chron stands for "Chronological objects which can handle dates and times"

```
library(chron)
## library(help = chron)
```

Manipulation in chron

```
as.chron("2014-02-20 23:50:26")
class(SampleDate1)
dates(SampleDate1)
```

Note that chron doesn’t adjust for daylight savings time

The lubridate package

- The lubridate package is a more intuitive wrapper for POSIXct with easier syntax

```
library(lubridate)
```

lubridate Usage

- There are direct date functions in the package that can be used to input dates:

```
lubri_read1 <- ymd_hms("2010-02-24 02:45:56")
(lubri_read2 <- mdy_hm("08/13/12 09:23"))
```

```
## [1] "2012-08-13 09:23:00 UTC"
```

```
lubri_read3 <- ydm_hm("2014-29-01 5:00am")
(lubri_read4 <- dmy("13082012"))
```

```
## [1] "2012-08-13 UTC"
```

lubridate Usage

Components of the object can be extracted and reassigned

```
year(lubri_read1)
```

```
## [1] 2010
```

```
week(lubri_read2)
```

```
## [1] 33
```

```
wday(lubri_read1, label = TRUE)
```

```
## [1] Wed
## Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat
```

lubridate Usage

```
tz(lubri_read1)
```

```
## [1] "UTC"
```

```
second(lubri_read1)
```

```
## [1] 56
```

```
minute(lubri_read2)
```

```
## [1] 23
```

`lubridate` Usage

`lubridate` takes note of four types of objects:

instant A point on the time-line.

interval The time between two instants

duration An amount of time without a specific beginning/end

period A time span recorded in a unit larger than seconds (e.g. decades)

An Instant

- An instant is a specific moment in time. The other types are different ways of recording time spans
- Date-time objects are read as instants

```
is.instant(lubri_read2)
```

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

```
is.interval(lubri_read2)
```

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

Time Zones

- `lubridate` uses UTC (Universal Coordinated Time) as the default timezone
- We can view or change the timezone of an instance when necessary

```
with_tz(lubri_read1, "America/New_York")
```

```
## [1] "2010-02-23 21:45:56 EST"
```

```
force_tz(lubri_read2, "America/Los_Angeles")
```

```
## [1] "2012-08-13 09:23:00 PDT"
```

Date Manipulation

The date manipulation steps we discussed can also be done in `lubridate`:

```
round_date(lubri_read1, "minute")
```

```
## [1] "2010-02-24 02:46:00 UTC"
```

```
round_date(lubri_read2, "day")
```

```
## [1] "2012-08-13 UTC"
```

System Time

```
today()
```

```
## [1] "2015-03-20"
```

```
now()
```

```
## [1] "2015-03-20 08:19:25 EDT"
```

Intervals

Intervals measure the span of time that happens between two specific instants.

```
(span34 <- new_interval(lubri_read4, lubri_read3))
```

```
## [1] 2012-08-13 UTC--2014-01-29 05:00:00 UTC
```

```
(span23 <- lubri_read2 %--% lubri_read3)
```

```
## [1] 2012-08-13 09:23:00 UTC--2014-01-29 05:00:00 UTC
```

Interval Usage

We can extract an interval's beginning and end.

```
int_start(span34)
```

```
## [1] "2012-08-13 UTC"
```

```
int_end(span23)
```

```
## [1] "2014-01-29 05:00:00 UTC"
```

Interval Usage

We can extract an interval's duration (in seconds).

```
int_length(span34)
```

```
## [1] 46155600
```

Interval Usage

We can shift an interval by some **duration**.

```
int_shift(span34, dyears(3))
```

```
## [1] 2015-08-13 UTC--2017-01-28 05:00:00 UTC
```

Interval Usage

We can check whether an instant happened inside an interval

```
lubri_read1 %within% span34
```

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

Interval Overlap

We can check whether two intervals overlap

```
interval1 <- new_interval(ymd_hm("2014-08-23 06:03"), ymd_hm("2014-08-25 11:02"))
int_overlaps(interval1, span34)
```

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

```
int_overlaps(span23, span34)
```

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

Duration Usage

A **duration** is essentially a time span. It has a measured length, stored in seconds internally

```
(FiftyMins <- dminutes(50))
```

```
## [1] "3000s (~50 minutes)"
```

```
(ThreeYears <- dyears(3))
```

```
## [1] "94608000s (~3 years)"
```

```
(IntervalDuration <- as.duration(interval1))
```

```
## [1] "190740s (~2.21 days)"
```

Duration Usage

Arithmetic operations can be done on these as well

```
lubri_read2 - FiftyMins
```

```
## [1] "2012-08-13 08:33:00 UTC"
```

```
ThreeYears + dyears(10)
```

```
## [1] "409968000s (~12.99 years)"
```

```
IntervalDuration/as.duration(span34)
```

```
## [1] 0.004132543
```

period Usage

A **period** is also a time span like durations, but its units are larger than seconds

```
(FourHours <- hours(4))
```

```
## [1] "4H 0M 0S"
```

```
(TwoWeeks <- weeks(2))
```

```
## [1] "14d 0H 0M 0S"
```

periods and Calculations

Calculations can be done on periods as well.

```
FourHours + TwoWeeks
```

```
## [1] "14d 4H 0M 0S"
```

`intervals` and `durations` are more precise instruments.

Outline

1. Dates in R
2. Date class
3. POSIXct and POSIXlt
4. Additional Packages
5. Summary

Summary

- When dealing with dates only, use `Date`
- `POSIXct` usually is best when dealing with dates and times together
- `POSIXlt` allows us to extract the components of the date-time object
- `chron` is the simplest approach when timezones and DST can be ignored
- `lubridate` is one of the best packages for handling dates and times

Other packages and functions

- The `timeDate` package, targeted towards datasets and analyses in finance
- The `fame` package, for additional time and date information
- `Sys.timezone`, `Sys.time`, `Sys.Date`, `date`, `Sys.setlocale`, and other similar system functions that can be used to check and/or alter date, time, and location settings

Review Questions

- What are the different types of date-time objects disussed in the module?
- How can we use the `lubridate` package to simplify working with dates and times in R?
- Can you think of data date-time formats which may present issues when analyzed?

Questions

Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com
1.855.GET.REVO
Twitter: @RevolutionR