Handling date-times in R

Cole Beck

August 30, 2012

1 Introduction

Date-time variables are a pain to work with in any language. We'll discuss some of the common issues and how to overcome them.

Before we examine the combination of dates and times, let's focus on dates. Even by themselves dates can be a pain. One idea is to refuse to use them. Generally dates are interally stored as integers, and depending on the operations you'll need, you could choose to do the same. As an added bonus, converting dates to integers may be useful in de-identifying your dataset.

One easy way to do this is by having a lookup table.

```
date.lookup <- format(seq(as.Date("2000-01-01"), as.Date("2010-12-31"), by = "1 day"))
match("2004-01-19", date.lookup)

## [1] 1480

date.lookup[1337]

## [1] "2003-08-29"
```

If that works for you, then that works for me and we're done here... let's assume you actually want dates though.

2 Date Class

The simplest data type to use for dates is the "Date" class. As mentioned, these will be internally stored as integers. In my example, day one was 2000-01-01. The specific date used to index your dates is called the origin. Typically programming languages use a default origin of 1970-01-01, though it is really day zero, not day one (negative values are perfectly valid).

```
# create a date
as.Date("2012-08-30")

## [1] "2012-08-30"

# specify the format
as.Date("08/30/2012", format = "%m/%d/%Y")

## [1] "2012-08-30"

# use a different origin, for instance importing values from Excel
as.Date(41149, origin = "1900-01-01")

## [1] "2012-08-30"
```

```
# take a difference
Sys.Date() - as.Date("1970-01-01")

## Time difference of 15582 days

# alternate method with specified units
difftime(Sys.Date(), as.Date("1970-01-01"), units = "days")

## Time difference of 15582 days

# see the internal integer representation
unclass(Sys.Date())

## [1] 15582
```

3 POSIXt Date-Time Class

Dates are pretty simple and most of the operations that we could use for them will also apply to date-time variables. There are at least three good options for date-time data types: built-in POSIXt, chron package, lubridate package. I rarely support using external packages (spoiler: I use POSIXt), if for no other reason than I don't like forcing dependencies on my code. But it can also be burdensome to write code that someone else has already written, so add-on packages are worth examining.

There are two POSIXt types, POSIXct and POSIXlt. "ct" can stand for calendar time, it stores the number of seconds since the origin. "lt", or local time, keeps the date as a list of time attributes (such as "hour" and "mon").

```
# current time as POSIXct
unclass(Sys.time())

## [1] 1.346e+09

# and as POSIXlt
unclass(as.POSIXlt(Sys.time()))

## $sec
## [1] 9.166
##
## $min
## [1] 54
##
## $hour
## [1] 13
##
## $mday
## [1] 30
##
## $mon
## [1] 7
##
## $year
## [1] 112
##
## $wday
## [1] 4
```

```
##
## $yday
## [1] 242
##
## $isdst
## [1] 1
##
## attr(,"tzone")
## [1] ""    "CST" "CDT"
```

3.1 Format

Thus far we've made the bold assumption that we want our date-times in the format YYYY-MM-DD. It's not hard to set the format when creating or printing date-times. In fact it's not a bad idea to always set the format so that you can be certain that your input/output looks like you expect. View the strftime help file for an idea on how to specify format.

```
# create POSIXct variables
as.POSIXct("080406 10:11", format = "%y%m%d %H:%M")

## [1] "2008-04-06 10:11:00 CDT"

as.POSIXct("2008-04-06 10:11:01 PM", format = "%Y-%m-%d %I:%M:%S %p")

## [1] "2008-04-06 22:11:01 CDT"

as.POSIXct("08/04/06 22:11:00", format = "%m/%d/%y %H:%M:%S")

## [1] "2006-08-04 22:11:00 CDT"

# convert POSIXct variables to character strings
format(as.POSIXct("080406 10:11", format = "%y%m%d %H:%M"), "%m/%d/%Y %I:%M %p")

## [1] "04/06/2008 10:11 AM"

as.character(as.POSIXct("080406 10:11", format = "%y%m%d %H:%M"), format = "%m-%d-%y %H:%M")

## [1] "04-06-08 10:11"
```

3.2 Example 1

We've finally covered the basics, let's run some practical examples.

```
# when do I turn 1 billion seconds old?
billbday <- function(bday, age = 10^9, format = "%Y-%m-%d %H:%M:%S") {
    x <- as.POSIXct(bday, format = format) + age
    togo <- round(difftime(x, Sys.time(), units = "days"))
    if (togo > 0) {
        msg <- sprintf("You will be %s seconds old on %s, which is %s days from now.", age,
            format(x, "%Y-%m-%d"), togo)
    } else {
        msg <- sprintf("You turned %s seconds old on %s, which was %s days ago.", age, format(x,
            "%Y-%m-%d"), -1 * togo)
    }
```

```
    if (age > 125 * 365.25 * 86400)
        msg <- paste(msg, "Good luck with that.")
    print(msg)
    format(x, "%Y-%m-%d")
}

billbday("1981-04-13 15:00:00")

## [1] "You will be 1e+09 seconds old on 2012-12-20, which is 112 days from now."

## [1] "2012-12-20"
```

3.3 Data.frames Hate You

That was easy enough, let's use a data.frame.

```
dts <- data.frame(day = c("20081101", "20081101", "20081101", "20081101", "20081101", "20081102",
    "20081102", "20081102", "20081102", "20081103"), time = c("01:20:00", "06:00:00", "12:20:00",
    "17:30:00", "21:45:00", "01:15:00", "06:30:00", "12:50:00", "20:00:00", "01:05:00"), value = c("5",
    "5", "6", "6", "5", "5", "6", "7", "5", "5"))

dts1 <- paste(dts$day, dts$time)
dts2 <- as.POSIXct(dts1, format = "%Y%m%d %H:%M:%S")
dts3 <- as.POSIXlt(dts1, format = "%Y%m%d %H:%M:%S")
dts.all <- data.frame(dts, ct = dts2, lt = dts3)
# lt changed to POSIXct!
str(dts.all)

## 'data.frame': 10 obs. of  5 variables:
##  $ day  : Factor w/ 3 levels "20081101","20081102",..: 1 1 1 1 1 2 2 2 2 3
##  $ time : Factor w/ 10 levels "01:05:00","01:15:00",..: 3 4 6 8 10 2 5 7 9 1
##  $ value: Factor w/ 3 levels "5","6","7": 1 1 2 2 1 1 2 3 1 1
##  $ ct   : POSIXct, format: "2008-11-01 01:20:00" "2008-11-01 06:00:00" ...
##  $ lt   : POSIXct, format: "2008-11-01 01:20:00" "2008-11-01 06:00:00" ...
```

Not even a warning? Stupid date-times, I don't care if the documentation states that you should use ct's for data.frames. Let's try again.

```
dts.all <- dts
dts.all$ct <- dts2
dts.all$lt <- dts3
str(dts.all)

## 'data.frame': 10 obs. of  5 variables:
##  $ day  : Factor w/ 3 levels "20081101","20081102",..: 1 1 1 1 1 2 2 2 2 3
##  $ time : Factor w/ 10 levels "01:05:00","01:15:00",..: 3 4 6 8 10 2 5 7 9 1
##  $ value: Factor w/ 3 levels "5","6","7": 1 1 2 2 1 1 2 3 1 1
##  $ ct   : POSIXct, format: "2008-11-01 01:20:00" "2008-11-01 06:00:00" ...
##  $ lt   : POSIXlt, format: "2008-11-01 01:20:00" "2008-11-01 06:00:00" ...

unclass(dts.all$ct)

##  [1] 1.226e+09 1.226e+09 1.226e+09 1.226e+09 1.226e+09 1.226e+09 1.226e+09 1.226e+09
##  [9] 1.226e+09 1.226e+09
```

```
## attr(,"tzone")
## [1] ""

unclass(dts.all$lt)

## $sec
##  [1] 0 0 0 0 0 0 0 0 0 0
##
## $min
##  [1] 20  0 20 30 45 15 30 50  0  5
##
## $hour
##  [1]  1  6 12 17 21  1  6 12 20  1
##
## $mday
##  [1] 1 1 1 1 1 2 2 2 2 3
##
## $mon
##  [1] 10 10 10 10 10 10 10 10 10 10
##
## $year
##  [1] 108 108 108 108 108 108 108 108 108 108
##
## $wday
##  [1] 6 6 6 6 6 0 0 0 0 1
##
## $yday
##  [1] 305 305 305 305 305 306 306 306 306 307
##
## $isdst
##  [1] 1 1 1 1 1 1 0 0 0 0
##
```

That worked? Suspicious behaviour now may yield suspicious behaviour later. Let's try rounding our times.

```
# round sets to POSIXlt and fails!
dts.all[, "ct"] <- round(dts.all[, "ct"], units = "hours")

## Warning:  provided 9 variables to replace 1 variables

# force back to POSIXct
dts.all[, "ct"] <- as.POSIXct(round(dts2, units = "hours"))

# rounding our POSIXlt column also fails
dts.all[, "lt"] <- round(dts3, units = "hours")

## Warning:  provided 9 variables to replace 1 variables

# while this works
dts.all$lt <- round(dts3, units = "hours")
```

Let's try to replace one element of one row.

```
# this fails
dts.all[1, "lt"] <- as.POSIXlt("2008-11-01 01:00:00")

## Warning:  provided 9 variables to replace 1 variables

## Error:  'origin' must be supplied

# this works
dts.all$lt[1] <- as.POSIXlt("2008-11-01 01:00:00")
# this works?
dts.all[1, "lt"] <- as.POSIXct("2008-11-01 01:00:00")
```

There's magic in the $. But it seems we can use this trick to force things (that shouldn't be forced?).

So maybe there's a reason we're supposed to use POSIXct times in data.frames. While tricks are discouraged, and it might take some effort, it's certainly nice to have access to our list attributes.

```
time1 <- dts.all$lt[1]
time2 <- dts.all$lt[2]
# presumably do something with the data from time1 to time2
while (time1 < time2) {
    time1$hour <- time1$hour + 1
}
print(sprintf("%s -- %s", time1, time2))

## [1] "2008-11-01 06:00:00 -- 2008-11-01 06:00:00"
```

3.4 Daylight Savings and Time Zones [also hate you]

```
# another example
time1 <- dts.all$lt[5]
time2 <- dts.all$lt[7]
while (time1 < time2) {
    time1$hour <- time1$hour + 1
    # notice how hour is the only attribute to change
    print(unlist(time1))
}

##   sec   min  hour  mday   mon  year  wday  yday isdst
##     0     0    23     1    10   108     6   305     1
##   sec   min  hour  mday   mon  year  wday  yday isdst
##     0     0    24     1    10   108     6   305     1
##   sec   min  hour  mday   mon  year  wday  yday isdst
##     0     0    25     1    10   108     6   305     1
##   sec   min  hour  mday   mon  year  wday  yday isdst
##     0     0    26     1    10   108     6   305     1
##   sec   min  hour  mday   mon  year  wday  yday isdst
##     0     0    27     1    10   108     6   305     1
##   sec   min  hour  mday   mon  year  wday  yday isdst
##     0     0    28     1    10   108     6   305     1
##   sec   min  hour  mday   mon  year  wday  yday isdst
##     0     0    29     1    10   108     6   305     1
##   sec   min  hour  mday   mon  year  wday  yday isdst
##     0     0    30     1    10   108     6   305     1
```

```
##   sec   min  hour  mday   mon  year  wday  yday isdst
##     0     0    31     1    10   108     6   305     1
##   sec   min  hour  mday   mon  year  wday  yday isdst
##     0     0    32     1    10   108     6   305     1

print(sprintf("%s -- %s", time1, time2))

## [1] "2008-11-02 08:00:00 -- 2008-11-02 07:00:00"


# DST and tzone are confusing they're still equal
time1 == time2

## [1] TRUE

# combining them seems to work
c(time1, time2)

## [1] "2008-11-02 07:00:00 CST" "2008-11-02 07:00:00 CST"

# default to right tzone
as.POSIXlt(as.POSIXct(time1))

## [1] "2008-11-02 07:00:00 CST"
```

Remember our first solution, where we decided to skip dates altogther and convert everything to integer values? It sure would be nice if we didn't have to worry about daylight savings or local time zones. We can use universal time (UTC) for that. Actually, like specifying the format, it's a good idea to always specify your time zone even if you don't use UTC.

```
# universal time
dts4 <- round(as.POSIXlt(dts1, format = "%Y%m%d %H:%M:%S", tz = "UTC"), units = "hours")
dts4

##  [1] "2008-11-01 01:00:00 UTC" "2008-11-01 06:00:00 UTC" "2008-11-01 12:00:00 UTC"
##  [4] "2008-11-01 18:00:00 UTC" "2008-11-01 22:00:00 UTC" "2008-11-02 01:00:00 UTC"
##  [7] "2008-11-02 07:00:00 UTC" "2008-11-02 13:00:00 UTC" "2008-11-02 20:00:00 UTC"
## [10] "2008-11-03 01:00:00 UTC"

# central standard time
round(as.POSIXlt(dts1, format = "%Y%m%d %H:%M:%S", tz = "CST"), units = "hours")

##  [1] "2008-11-01 01:00:00 CST" "2008-11-01 06:00:00 CST" "2008-11-01 12:00:00 CST"
##  [4] "2008-11-01 18:00:00 CST" "2008-11-01 22:00:00 CST" "2008-11-02 01:00:00 CST"
##  [7] "2008-11-02 07:00:00 CST" "2008-11-02 13:00:00 CST" "2008-11-02 20:00:00 CST"
## [10] "2008-11-03 01:00:00 CST"
```

3.5 Example 2

Let's examine a function that will fill in values when we have missing dates.

```
# POSIXct data.frame
mydata.ct <- data.frame(date = dts.all$ct, value = dts.all$value)
# POSIXlt data.frame
mydata.lt <- data.frame(date = NA, value = dts.all$value)
```

```
mydata.lt$date <- dts.all$lt
# POSIX.lt data.frame using UTC
mydata.lt.utc <- data.frame(date = NA, value = dts.all$value)
mydata.lt.utc$date <- dts4

fill.dates <- function(x) {
    dates <- seq(x[1, "date"], x[nrow(x), "date"], by = "hours")
    x2 <- data.frame(date = rep(NA, length(dates)), value = NA)
    x2$date <- as.POSIXlt(dates)
    x2$value[match(as.character(x$date), as.character(x2$date))] <- x$value
    for (i in which(is.na(x2$value))) x2[i, "value"] <- x2[i - 1, "value"]
    x2
}

data.ct <- fill.dates(mydata.ct)
data.lt <- fill.dates(mydata.lt)
data.lt.utc <- fill.dates(mydata.lt.utc)
cbind(data.ct[21:30, ], data.lt[21:30, ], data.lt.utc[21:30, ])
```

```
##                    date value                date value                date value
## 21 2008-11-01 21:00:00     2 2008-11-01 21:00:00     2 2008-11-01 21:00:00     2
## 22 2008-11-01 22:00:00     1 2008-11-01 22:00:00     1 2008-11-01 22:00:00     1
## 23 2008-11-01 23:00:00     1 2008-11-01 23:00:00     1 2008-11-01 23:00:00     1
## 24 2008-11-02 00:00:00     1 2008-11-02 00:00:00     1 2008-11-02 00:00:00     1
## 25 2008-11-02 01:00:00     1 2008-11-02 01:00:00     1 2008-11-02 01:00:00     1
## 26 2008-11-02 01:00:00     1 2008-11-02 01:00:00     1 2008-11-02 02:00:00     1
## 27 2008-11-02 02:00:00     1 2008-11-02 02:00:00     1 2008-11-02 03:00:00     1
## 28 2008-11-02 03:00:00     1 2008-11-02 03:00:00     1 2008-11-02 04:00:00     1
## 29 2008-11-02 04:00:00     1 2008-11-02 04:00:00     1 2008-11-02 05:00:00     1
## 30 2008-11-02 05:00:00     1 2008-11-02 05:00:00     1 2008-11-02 06:00:00     1
```

4 Chron Package

Let's examine the chron package. "chron" is short for chronological, not chronic pain, which would be logical given the amount of pain programmers have had to deal with implementing date-times over the years (last joke, I promise). It's not a base package, so you'll have to install it, as will everyone else that runs your code.

It's important to note that chron expects numeric values to be time since origin. If you want 101010 to mean 2010-10-10, you'll need to set it as a character string first (otherwise it's 2246-07-23).

```
library(chron)
# fails on a factor
chron(dts$day, format = "ymd")
```

```
## Error:  object dates.  must be numeric or character
```

```
chron(as.character(dts$day), format = "ymd")
```

```
##  [1] 081101 081101 081101 081101 081101 081102 081102 081102 081102 081103
```

```
chron.dts <- chron(date = as.character(dts$day), time = dts$time, format = c("ymd", "h:m:s"))
cdts.all <- dts
cdts.all$cd <- chron.dts
# I hate two-digit year format; methods to display 4 are awful
format(as.POSIXlt(cdts.all$cd, tz = "UTC"), "%Y%m%d %H:%M:%S")
```

```
##  [1] "20081101 01:20:00" "20081101 06:00:00" "20081101 12:20:00" "20081101 17:30:00"
##  [5] "20081101 21:45:00" "20081102 01:15:00" "20081102 06:30:00" "20081102 12:50:00"
##  [9] "20081102 20:00:00" "20081103 01:05:00"

# options(chron.year.abb=FALSE) # don't use options as a global variable
```

So what does chron have to offer? In my opinion, very little, it mostly replicates what's already there.

5 Lubridate Package

Lubridate is another package you may wish to install. It's documentation boasts that it "makes working with dates fun instead of frustrating". It does include many functions unavailable to us with POSIXt objects. More importantly the basic stuff is easy to do.

```
library(lubridate)
ymd_hms("2012-12-31 23:59:59")

## [1] "2012-12-31 23:59:59 UTC"

ldate <- mdy_hms("12/31/2012 23:59:59")
ldate + seconds(1)

## [1] "2013-01-01 UTC"

month(ldate) <- 8
```

Understanding the different objects used to represent time is the key to using lubridate.

5.1 Instants

An instant is a specific moment in time.

```
now()

## [1] "2012-08-30 13:54:09 CDT"

# last day of the month
ceiling_date(now(), unit = "month") - days(1)

## [1] "2012-08-31 CDT"

# is it morning?
am(now())

## [1] FALSE
```

5.2 Intervals, Durations, Periods

An interval is the time in between two instants. Know it exists, but assume you don't care. Durations are exact time spans, measured in seconds. It is the absolute time between two events. Periods are used to handle relative time measurements between two events.

```
# how do you want to handle leap years?
ldate - dyears(1)
```

```
## [1] "2011-09-01 23:59:59 UTC"

ldate - years(1)

## [1] "2011-08-31 23:59:59 UTC"

# or DST?
force_tz(ymd_hms(as.character(dts.all$lt[5])), "America/Chicago") + dhours(6)

## [1] "2008-11-02 03:00:00 CST"

force_tz(ymd_hms(as.character(dts.all$lt[5])), "America/Chicago") + hours(6)

## [1] "2008-11-02 04:00:00 CST"

# within is a nice function to use with intervals
now() %within% new_interval(ymd("2012-07-01"), ymd("2013-06-30"))

## [1] TRUE
```

I don't know about fun, but you could make a very strong argument to use lubridate to handle your date-time variables.

6 Bonus Example: Determine Your Format

This simple function will try to determine your date-time format. You'll have problems if the format isn't consistent. This function could easily be extended, but as is, it may be helpful.

```
# FUNCTION guessDateFormat @x vector of character dates/datetimes @returnDates return
# actual dates rather than format convert character datetime to POSIXlt datetime, or at
# least guess the format such that you could convert to datetime
guessDateFormat <- function(x, returnDates = FALSE, tzone = "") {
    x1 <- x
    # replace blanks with NA and remove
    x1[x1 == ""] <- NA
    x1 <- x1[!is.na(x1)]
    if (length(x1) == 0)
        return(NA)
    # if it's already a time variable, set it to character
    if ("POSIXt" %in% class(x1[1])) {
        x1 <- as.character(x1)
    }
    dateTimes <- do.call(rbind, strsplit(x1, " "))
    for (i in ncol(dateTimes)) {
        dateTimes[dateTimes[, i] == "NA"] <- NA
    }
    # assume the time part can be found with a colon
    timePart <- which(apply(dateTimes, MARGIN = 2, FUN = function(i) {
        any(grepl(":", i))
    }))
    # everything not in the timePart should be in the datePart
    datePart <- setdiff(seq(ncol(dateTimes)), timePart)
    # should have 0 or 1 timeParts and exactly one dateParts
    if (length(timePart) > 1 || length(datePart) != 1)
        stop("cannot parse your time variable")
```

```
    timeFormat <- NA
    if (length(timePart)) {
        # find maximum number of colons in the timePart column
        ncolons <- max(nchar(gsub("[^:]", "", na.omit(dateTimes[, timePart]))))
        if (ncolons == 1) {
            timeFormat <- "%H:%M"
        } else if (ncolons == 2) {
            timeFormat <- "%H:%M:%S"
        } else stop("timePart should have 1 or 2 colons")
    }
    # remove all non-numeric values
    dates <- gsub("[^0-9]", "", na.omit(dateTimes[, datePart]))
    # sep is any non-numeric value found, hopefully / or -
    sep <- unique(na.omit(substr(gsub("[0-9]", "", dateTimes[, datePart]), 1, 1)))
    if (length(sep) > 1)
        stop("too many seperators in datePart")
    # maximum number of characters found in the date part
    dlen <- max(nchar(dates))
    dateFormat <- NA
    # when six, expect the century to be omitted
    if (dlen == 6) {
        if (sum(is.na(as.Date(dates, format = "%y%m%d"))) == 0) {
            dateFormat <- paste("%y", "%m", "%d", sep = sep)
        } else if (sum(is.na(as.Date(dates, format = "%m%d%y"))) == 0) {
            dateFormat <- paste("%m", "%d", "%y", sep = sep)
        } else stop("datePart format [six characters] is inconsistent")
    } else if (dlen == 8) {
        if (sum(is.na(as.Date(dates, format = "%Y%m%d"))) == 0) {
            dateFormat <- paste("%Y", "%m", "%d", sep = sep)
        } else if (sum(is.na(as.Date(dates, format = "%m%d%Y"))) == 0) {
            dateFormat <- paste("%m", "%d", "%Y", sep = sep)
        } else stop("datePart format [eight characters] is inconsistent")
    } else {
        stop(sprintf("datePart has unusual length: %s", dlen))
    }
    if (is.na(timeFormat)) {
        format <- dateFormat
    } else if (timePart == 1) {
        format <- paste(timeFormat, dateFormat)
    } else if (timePart == 2) {
        format <- paste(dateFormat, timeFormat)
    } else stop("cannot parse your time variable")
    if (returnDates)
        return(as.POSIXlt(x, format = format, tz = tzone))
    format
}

# generate some dates
mydates <- format(as.POSIXct(sample(31536000, 20), origin = "2011-01-01", tz = "UTC"), "%m/%d/%Y %H:%M")
mydates

##  [1] "02/07/2011 06:51" "11/21/2011 17:03" "09/17/2011 22:42" "02/16/2011 13:45"
##  [5] "12/14/2011 19:11" "09/08/2011 09:22" "12/06/2011 14:06" "02/02/2011 11:00"
##  [9] "03/27/2011 06:12" "01/05/2011 15:09" "04/15/2011 04:17" "10/20/2011 14:20"
```

```
## [13] "11/13/2011 21:46" "02/26/2011 03:24" "12/29/2011 11:02" "03/17/2011 02:24"
## [17] "02/27/2011 13:51" "06/27/2011 08:36" "03/14/2011 10:54" "01/28/2011 14:14"

guessDateFormat(mydates)

## [1] "%m/%d/%Y %H:%M"
```