Dates and Times

Greg Ridgeway (gridge@upenn.edu)
Ruth Moyer (moyruth@upenn.edu)

July 26, 2018

Introduction

Working with dates and times is a lot different than working with the more familiar numbers. Months have different number of days. Sometimes we count hours of the day up to 12 and then start over. Sometimes we count hours up to 24 and then start over. Some years have 366 days. We have 24 times zones around the world. Twice a year we switch clocks for Daylight Saving Time, except in some places like Arizona. Arithmetic, such as adding one month to a date, is poorly defined. Which date is one month after January 31st? Is it February 28th? Or is it March 3rd?

Fortunately, software for working with dates exist to make these tasks easier. Unfortunately, every system seems to make their own design decisions. Excel stores dates as the number of days since January 0, 1900... that's not a typo... they count from January 0, 1900. Linux systems count days since January 1, 1970. SPSS stores times as the number of seconds since midnight October 14, 1582, the adoption date of the Gregorian calendar. Much of the world did not adopt the calendar in 1582. The American colonies did not adopt the Gregorian calendar until 1752 along with Great Britain. So beware if you are a historian digging through centuries old data. Aligning dates can become very messy.

R has had a variety of attempts at providing a means for managing dates. We are going to use the `lubridate` package that address just about everything you might need to do with dates and times.

`lubridate` is not part of R by default. You will need to install it. Simply run

```
install.packages("lubridate")
```

and R will hit the web, download the `lubridate` package and any supporting packages it needs (and it does need a few), and installs them. This is a one time event. Once you have `lubridate` on your machine you will not need to reinstall it every time you need it.

Some of our students, particularly on Macs, have encountered trouble installing some packages for R. R will sometimes try to download the source code for the packages and compile them from scratch on your machine. Sometimes that goes well and other times it requires that you have other tools installed on your machine. Any easy solution is to run

```
install.packages("lubridate", type="mac.binary")
```

instead to insist that R finds and installs a ready-to-use version of the packages.

Working with dates

While `lubridate` is now installed, once per R session you will need to load `lubridate`.

```
library(lubridate)
```

If you close R and restart it, then you'll need to run this line again.

Let's reload the sample of Chicago crime data discussed in the introductory notes, available on the R4Crim github site.

```
load("chicago crime 20141124-20141209.RData")
```

Let's extract five dates from the `chicagoCrime` dataset.

```
i <- c(1,2500,5000,7500,10000)   # select 5 rows to get the dates
chicagoCrime$Date[i]
```

```
[1] "12/09/2014 11:54:00 PM" "12/05/2014 11:00:00 PM"
[3] "12/02/2014 10:58:00 AM" "11/28/2014 02:35:00 PM"
[5] "11/24/2014 12:30:00 AM"
```

As you can see the dates include the date in month/day/year format and the time on a 12 hour AM/PM clock. R has no idea that these values represent dates. You are familiar with this date formatting, but R just thinks they are strings of characters. Use `substring()` to just extract the date part, the first 10 characters of `Date`.

```
textDate <- substring(chicagoCrime$Date[i],1,10)
textDate
```

```
[1] "12/09/2014" "12/05/2014" "12/02/2014" "11/28/2014" "11/24/2014"
```

Now let's use the `mdy()` function from the `lubridate` package to tell R that these are not just strings of characters, but they actually represent months, days, and years.

```
b <- mdy(textDate)
is(b)
b
```

```
[1] "Date"     "oldClass"
[1] "2014-12-09" "2014-12-05" "2014-12-02" "2014-11-28" "2014-11-24"
```

`b` now stores those five dates in a format that recognizes the month, day, and year. `is(b)` tells us that R is storing `b` as a date. There are different functions for other date formats depending on the ordering of the day, month and year, like `dmy()` and `ymd()` and even `mdy_hms()` for month, day, year, hours, minutes, seconds format.

Now that R knows these are dates, the `lubridate` package provides a lot of functions to help you work with dates.

```
year(b)
month(b)
month(b,label=TRUE)
month(b,label=TRUE,abbr=FALSE)
wday(b,label=TRUE)
```

```
[1] 2014 2014 2014 2014 2014
[1] 12 12 12 11 11
```

```
[1] Dec Dec Dec Nov Nov
12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
[1] December December December November November
12 Levels: January < February < March < April < May < June < ... < December
[1] Tue Fri Tue Fri Mon
Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

Subtraction will tell you the time between two dates. How many days since December 1, 2014? How many days have passed from the values in `b` to today? The `now()` function give you the date and time, well... right now.

```
b-mdy("12/01/2014")
date(now())-b
```

```
Time differences in days
[1]  8  4  1 -3 -7
Time differences in days
[1] 1325 1329 1332 1336 1340
```

We can add time to the dates as well

```
b + dyears(1) # adds 365 days, does not increase year by 1
b + ddays(31)
```

```
[1] "2015-12-09" "2015-12-05" "2015-12-02" "2015-11-28" "2015-11-24"
[1] "2015-01-09" "2015-01-05" "2015-01-02" "2014-12-29" "2014-12-25"
```

Now let's go ahead and create a new column in our Chicago dataset containing properly stored dates.

```
chicagoCrime$realdate <- mdy_hms(chicagoCrime$Date)
chicagoCrime[1:5,c("Date","realdate")]   # show the dates in the first five rows
```

```
                  Date            realdate
1 12/09/2014 11:54:00 PM 2014-12-09 23:54:00
2 12/09/2014 11:45:00 PM 2014-12-09 23:45:00
3 12/09/2014 11:42:00 PM 2014-12-09 23:42:00
4 12/09/2014 11:42:00 PM 2014-12-09 23:42:00
5 12/09/2014 11:40:00 PM 2014-12-09 23:40:00
```

`lubridate` has converted the date and time formats to a more standardized form, one that is easier to use on a computer.

The default timezone is Coordinated Universal Time abbreviated UTC, which is the same as Greenwich Mean Time. Interestingly, the abbreviation CUT would make more sense in English, but TCU would make more sense in French, so the compromise was to universally abbreviate as UTC. Since all of these crimes occurred in Chicago, let's explicitly set the timezone to Central Time. The function `OlsonNames()` will give you a list of all possible time zones you can use.

```
chicagoCrime$realdate <- force_tz(chicagoCrime$realdate, "America/Chicago")
chicagoCrime$realdate[1:5]   # show just the first five dates
```

```
[1] "2014-12-09 23:54:00 CST" "2014-12-09 23:45:00 CST"
```

```
[3] "2014-12-09 23:42:00 CST" "2014-12-09 23:42:00 CST"
[5] "2014-12-09 23:40:00 CST"
```

Now when printed you can see that the timezone is set to Central Standard Time. R will automatically handle Daylight Saving Time. Note that an August date reports Central Daylight Time.

```
force_tz(mdy_hms("8/1/2014 12:00:00"), "America/Chicago")
```

```
[1] "2014-08-01 12:00:00 CDT"
```

We can actually find out when Daylight Saving Time ends. Generate all November dates and convert them to Chicago time.

```
b <- paste0("11/",1:30,"/2016 12:00:00")
mdy_hms(b,tz="America/Chicago")
```

```
 [1] "2016-11-01 12:00:00 CDT" "2016-11-02 12:00:00 CDT"
 [3] "2016-11-03 12:00:00 CDT" "2016-11-04 12:00:00 CDT"
 [5] "2016-11-05 12:00:00 CDT" "2016-11-06 12:00:00 CST"
 [7] "2016-11-07 12:00:00 CST" "2016-11-08 12:00:00 CST"
 [9] "2016-11-09 12:00:00 CST" "2016-11-10 12:00:00 CST"
[11] "2016-11-11 12:00:00 CST" "2016-11-12 12:00:00 CST"
[13] "2016-11-13 12:00:00 CST" "2016-11-14 12:00:00 CST"
[15] "2016-11-15 12:00:00 CST" "2016-11-16 12:00:00 CST"
[17] "2016-11-17 12:00:00 CST" "2016-11-18 12:00:00 CST"
[19] "2016-11-19 12:00:00 CST" "2016-11-20 12:00:00 CST"
[21] "2016-11-21 12:00:00 CST" "2016-11-22 12:00:00 CST"
[23] "2016-11-23 12:00:00 CST" "2016-11-24 12:00:00 CST"
[25] "2016-11-25 12:00:00 CST" "2016-11-26 12:00:00 CST"
[27] "2016-11-27 12:00:00 CST" "2016-11-28 12:00:00 CST"
[29] "2016-11-29 12:00:00 CST" "2016-11-30 12:00:00 CST"
```

Looks like by noon on November 6, 2016, Chicago was back to Central Standard Time.

Exercises

1. At what hour does Daylight Saving Time end? (Hint: Try using `dminutes()` to add time to the date DST ends)
2. Thanksgiving occurs on the fourth month in November. On what date will Thanksgiving fall in 2020? Hints:
   - Try listing all dates in November
   - Use `wday()` to get the weekday
   - find the fourth Thursday
3. Make a function that takes as input a year and returns the date of Thanksgiving in that year. Here's a template to start

```
tday <- function(year)
{

```

```
   return( )
}
```

Solutions to the exercises

1. At what hour does Daylight Saving Time end?

```
mdy("11/6/2016", tz="America/Chicago") + dminutes(1:180)
```

```
  [1] "2016-11-06 00:01:00 CDT" "2016-11-06 00:02:00 CDT"
  [3] "2016-11-06 00:03:00 CDT" "2016-11-06 00:04:00 CDT"
  [5] "2016-11-06 00:05:00 CDT" "2016-11-06 00:06:00 CDT"
  [7] "2016-11-06 00:07:00 CDT" "2016-11-06 00:08:00 CDT"
  [9] "2016-11-06 00:09:00 CDT" "2016-11-06 00:10:00 CDT"
 [11] "2016-11-06 00:11:00 CDT" "2016-11-06 00:12:00 CDT"
 [13] "2016-11-06 00:13:00 CDT" "2016-11-06 00:14:00 CDT"
 [15] "2016-11-06 00:15:00 CDT" "2016-11-06 00:16:00 CDT"
 [17] "2016-11-06 00:17:00 CDT" "2016-11-06 00:18:00 CDT"
 [19] "2016-11-06 00:19:00 CDT" "2016-11-06 00:20:00 CDT"
 [21] "2016-11-06 00:21:00 CDT" "2016-11-06 00:22:00 CDT"
 [23] "2016-11-06 00:23:00 CDT" "2016-11-06 00:24:00 CDT"
 [25] "2016-11-06 00:25:00 CDT" "2016-11-06 00:26:00 CDT"
 [27] "2016-11-06 00:27:00 CDT" "2016-11-06 00:28:00 CDT"
 [29] "2016-11-06 00:29:00 CDT" "2016-11-06 00:30:00 CDT"
 [31] "2016-11-06 00:31:00 CDT" "2016-11-06 00:32:00 CDT"
 [33] "2016-11-06 00:33:00 CDT" "2016-11-06 00:34:00 CDT"
 [35] "2016-11-06 00:35:00 CDT" "2016-11-06 00:36:00 CDT"
 [37] "2016-11-06 00:37:00 CDT" "2016-11-06 00:38:00 CDT"
 [39] "2016-11-06 00:39:00 CDT" "2016-11-06 00:40:00 CDT"
 [41] "2016-11-06 00:41:00 CDT" "2016-11-06 00:42:00 CDT"
 [43] "2016-11-06 00:43:00 CDT" "2016-11-06 00:44:00 CDT"
 [45] "2016-11-06 00:45:00 CDT" "2016-11-06 00:46:00 CDT"
 [47] "2016-11-06 00:47:00 CDT" "2016-11-06 00:48:00 CDT"
 [49] "2016-11-06 00:49:00 CDT" "2016-11-06 00:50:00 CDT"
 [51] "2016-11-06 00:51:00 CDT" "2016-11-06 00:52:00 CDT"
 [53] "2016-11-06 00:53:00 CDT" "2016-11-06 00:54:00 CDT"
 [55] "2016-11-06 00:55:00 CDT" "2016-11-06 00:56:00 CDT"
 [57] "2016-11-06 00:57:00 CDT" "2016-11-06 00:58:00 CDT"
 [59] "2016-11-06 00:59:00 CDT" "2016-11-06 01:00:00 CDT"
 [61] "2016-11-06 01:01:00 CDT" "2016-11-06 01:02:00 CDT"
 [63] "2016-11-06 01:03:00 CDT" "2016-11-06 01:04:00 CDT"
 [65] "2016-11-06 01:05:00 CDT" "2016-11-06 01:06:00 CDT"
 [67] "2016-11-06 01:07:00 CDT" "2016-11-06 01:08:00 CDT"
 [69] "2016-11-06 01:09:00 CDT" "2016-11-06 01:10:00 CDT"
 [71] "2016-11-06 01:11:00 CDT" "2016-11-06 01:12:00 CDT"
 [73] "2016-11-06 01:13:00 CDT" "2016-11-06 01:14:00 CDT"
```

```
 [75] "2016-11-06 01:15:00 CDT" "2016-11-06 01:16:00 CDT"
 [77] "2016-11-06 01:17:00 CDT" "2016-11-06 01:18:00 CDT"
 [79] "2016-11-06 01:19:00 CDT" "2016-11-06 01:20:00 CDT"
 [81] "2016-11-06 01:21:00 CDT" "2016-11-06 01:22:00 CDT"
 [83] "2016-11-06 01:23:00 CDT" "2016-11-06 01:24:00 CDT"
 [85] "2016-11-06 01:25:00 CDT" "2016-11-06 01:26:00 CDT"
 [87] "2016-11-06 01:27:00 CDT" "2016-11-06 01:28:00 CDT"
 [89] "2016-11-06 01:29:00 CDT" "2016-11-06 01:30:00 CDT"
 [91] "2016-11-06 01:31:00 CDT" "2016-11-06 01:32:00 CDT"
 [93] "2016-11-06 01:33:00 CDT" "2016-11-06 01:34:00 CDT"
 [95] "2016-11-06 01:35:00 CDT" "2016-11-06 01:36:00 CDT"
 [97] "2016-11-06 01:37:00 CDT" "2016-11-06 01:38:00 CDT"
 [99] "2016-11-06 01:39:00 CDT" "2016-11-06 01:40:00 CDT"
[101] "2016-11-06 01:41:00 CDT" "2016-11-06 01:42:00 CDT"
[103] "2016-11-06 01:43:00 CDT" "2016-11-06 01:44:00 CDT"
[105] "2016-11-06 01:45:00 CDT" "2016-11-06 01:46:00 CDT"
[107] "2016-11-06 01:47:00 CDT" "2016-11-06 01:48:00 CDT"
[109] "2016-11-06 01:49:00 CDT" "2016-11-06 01:50:00 CDT"
[111] "2016-11-06 01:51:00 CDT" "2016-11-06 01:52:00 CDT"
[113] "2016-11-06 01:53:00 CDT" "2016-11-06 01:54:00 CDT"
[115] "2016-11-06 01:55:00 CDT" "2016-11-06 01:56:00 CDT"
[117] "2016-11-06 01:57:00 CDT" "2016-11-06 01:58:00 CDT"
[119] "2016-11-06 01:59:00 CDT" "2016-11-06 01:00:00 CST"
[121] "2016-11-06 01:01:00 CST" "2016-11-06 01:02:00 CST"
[123] "2016-11-06 01:03:00 CST" "2016-11-06 01:04:00 CST"
[125] "2016-11-06 01:05:00 CST" "2016-11-06 01:06:00 CST"
[127] "2016-11-06 01:07:00 CST" "2016-11-06 01:08:00 CST"
[129] "2016-11-06 01:09:00 CST" "2016-11-06 01:10:00 CST"
[131] "2016-11-06 01:11:00 CST" "2016-11-06 01:12:00 CST"
[133] "2016-11-06 01:13:00 CST" "2016-11-06 01:14:00 CST"
[135] "2016-11-06 01:15:00 CST" "2016-11-06 01:16:00 CST"
[137] "2016-11-06 01:17:00 CST" "2016-11-06 01:18:00 CST"
[139] "2016-11-06 01:19:00 CST" "2016-11-06 01:20:00 CST"
[141] "2016-11-06 01:21:00 CST" "2016-11-06 01:22:00 CST"
[143] "2016-11-06 01:23:00 CST" "2016-11-06 01:24:00 CST"
[145] "2016-11-06 01:25:00 CST" "2016-11-06 01:26:00 CST"
[147] "2016-11-06 01:27:00 CST" "2016-11-06 01:28:00 CST"
[149] "2016-11-06 01:29:00 CST" "2016-11-06 01:30:00 CST"
[151] "2016-11-06 01:31:00 CST" "2016-11-06 01:32:00 CST"
[153] "2016-11-06 01:33:00 CST" "2016-11-06 01:34:00 CST"
[155] "2016-11-06 01:35:00 CST" "2016-11-06 01:36:00 CST"
[157] "2016-11-06 01:37:00 CST" "2016-11-06 01:38:00 CST"
[159] "2016-11-06 01:39:00 CST" "2016-11-06 01:40:00 CST"
[161] "2016-11-06 01:41:00 CST" "2016-11-06 01:42:00 CST"
[163] "2016-11-06 01:43:00 CST" "2016-11-06 01:44:00 CST"
[165] "2016-11-06 01:45:00 CST" "2016-11-06 01:46:00 CST"
[167] "2016-11-06 01:47:00 CST" "2016-11-06 01:48:00 CST"
[169] "2016-11-06 01:49:00 CST" "2016-11-06 01:50:00 CST"
```

```
[171] "2016-11-06 01:51:00 CST" "2016-11-06 01:52:00 CST"
[173] "2016-11-06 01:53:00 CST" "2016-11-06 01:54:00 CST"
[175] "2016-11-06 01:55:00 CST" "2016-11-06 01:56:00 CST"
[177] "2016-11-06 01:57:00 CST" "2016-11-06 01:58:00 CST"
[179] "2016-11-06 01:59:00 CST" "2016-11-06 02:00:00 CST"
```

What happens after "2016-11-06 01:59:00 CDT"? You can see that it ends when the clock strikes two in the morning. If you want R to really do all the work, then the code gets a little more complicated and requires more digging in the help and lubridate code.

```
a <- mdy("11/6/2016", tz="America/Chicago") + dminutes(1:180)
tzone <- substring(format.POSIXct(a,usetz = TRUE), 21)
a[max(which(tzone=="CDT"))]
a[min(which(tzone=="CST"))]
```

```
[1] "2016-11-06 01:59:00 CDT"
[1] "2016-11-06 01:00:00 CST"
```

2. On what date will Thanksgiving fall in 2020?

```
a <- mdy(paste0("11/",1:30,"/2020"))
a[wday(a,label=TRUE)=="Thurs"][4]
```

```
[1] NA
```

3. Make a function that takes as input a year and returns the date of Thanksgiving in that year.

```
tday <- function(year)
{
   a <- mdy(paste0("11/",1:30,"/",year))
   return( a[wday(a,label=TRUE)=="Thurs"][4] )
}
tday(2020)
lapply(2020:2100, tday)
```

```
[1] NA
[[1]]
[1] NA

[[2]]
[1] NA

[[3]]
[1] NA

[[4]]
[1] NA

[[5]]
[1] NA
```

```
[[6]]
[1] NA

[[7]]
[1] NA

[[8]]
[1] NA

[[9]]
[1] NA

[[10]]
[1] NA

[[11]]
[1] NA

[[12]]
[1] NA

[[13]]
[1] NA

[[14]]
[1] NA

[[15]]
[1] NA

[[16]]
[1] NA

[[17]]
[1] NA

[[18]]
[1] NA

[[19]]
[1] NA

[[20]]
[1] NA

[[21]]
[1] NA
```

```
[[22]]
[1] NA

[[23]]
[1] NA

[[24]]
[1] NA

[[25]]
[1] NA

[[26]]
[1] NA

[[27]]
[1] NA

[[28]]
[1] NA

[[29]]
[1] NA

[[30]]
[1] NA

[[31]]
[1] NA

[[32]]
[1] NA

[[33]]
[1] NA

[[34]]
[1] NA

[[35]]
[1] NA

[[36]]
[1] NA

[[37]]
[1] NA
```

```
[[38]]
[1] NA

[[39]]
[1] NA

[[40]]
[1] NA

[[41]]
[1] NA

[[42]]
[1] NA

[[43]]
[1] NA

[[44]]
[1] NA

[[45]]
[1] NA

[[46]]
[1] NA

[[47]]
[1] NA

[[48]]
[1] NA

[[49]]
[1] NA

[[50]]
[1] NA

[[51]]
[1] NA

[[52]]
[1] NA

[[53]]
[1] NA
```

```
[[54]]
[1] NA

[[55]]
[1] NA

[[56]]
[1] NA

[[57]]
[1] NA

[[58]]
[1] NA

[[59]]
[1] NA

[[60]]
[1] NA

[[61]]
[1] NA

[[62]]
[1] NA

[[63]]
[1] NA

[[64]]
[1] NA

[[65]]
[1] NA

[[66]]
[1] NA

[[67]]
[1] NA

[[68]]
[1] NA

[[69]]
[1] NA
```

```
[[70]]
[1] NA

[[71]]
[1] NA

[[72]]
[1] NA

[[73]]
[1] NA

[[74]]
[1] NA

[[75]]
[1] NA

[[76]]
[1] NA

[[77]]
[1] NA

[[78]]
[1] NA

[[79]]
[1] NA

[[80]]
[1] NA

[[81]]
[1] NA
```