Formatting Dates Using lubridate

Danielle Quinn

Tuesday, November 03, 2015

One of the most common issues that new (and experienced!) R users have is formatting dates both prior to data import and during analyses. The simplest approach to dealing with dates is to ensure that the flat file (csv or txt) that is being read into R contains a *separate column for each component of date and time*. So, if date is a variable in your dataset, convert the date to three separate columns (day, month, year) prior to import, and if date and time are variables, convert those to six separate columns (second, minute, hour, day, month, year).

If you're using Excel to prepare your data for import, you can use the `=YEAR()` function, for example, to extract the year value from a cell formatted as a date. *Note:* If you ever try this and it results in an obscure date like "1900-01-01", it means that you just need to format your year column differently. If you're stuck, here's a link to a good explanation of how to deal with this!

http://www.excelfunctions.net/YearFunction.html

Here's a simple dataset that we'll use for this tutorial:
https://raw.githubusercontent.com/DanielleQuinn/RLessons/master/FormattingDates/date_data.txt

Load `lubridate` Package

```
library(lubridate)
```

Import Data

```
mydata<-read.delim("date_data.txt")
```

As you can see, the date and time information has been separated into individual columns prior to the data being imported into R

```
head(mydata)
```

```
  year month day hour minute second
1 2014     1  15   13     45     34
2 2014     1  15   22      0      0
3 2014     1  16   14     45     34
4 2014     1  16   23      0      0
5 2014     1  17   15     45     36
6 2014     1  17    0      0      0
```

We're going to be using the `ymd()` and `ymd_hms()` functions. These functions are designed to take character and numeric vectors, and convert it to a POSIXct object. Essentially, a

POSIXct object is recognized by R as being a date or date and time and will be handled as such. The format of the vectors that these functions can recognize is flexible, but it's generally a good idea to use a standardized method. Here, we're going to set up the input as a character vector formatted as "YYYY-MM-DD" (for `ymd()`) or "YYYY-MM-DD-HH-MM-SS" (for `ymd_hms()`).

Using `ymd()` to format dates

For now, we'll only be concerned with creating a column of dates consisting of year, month, and day, and exclude the time components.

Use the values of year, month, and day to create a character string that is formatted as "YYYY-MM-DD"

To do this, use the `paste()` function:

```
paste(mydata$year, mydata$month, mydata$day, sep="-")
```

```
 [1] "2014-1-15" "2014-1-15" "2014-1-16" "2014-1-16" "2014-1-17"
 [6] "2014-1-17" "2014-1-18" "2014-1-18" "2014-1-19" "2014-1-19"
[11] "2014-1-20" "2014-1-20" "2014-1-21" "2014-1-21" "2014-1-22"
[16] "2014-1-22" "2014-1-23" "2014-1-23"
```

Note: The `sep` argument in the `paste()` function determines what character(s) are used to separate each piece that is being pasted together.

Now we can wrap that piece of code in the `ymd()` function to create a vector of dates, properly formatted as POSIXct.

```
form.dates<-ymd(paste(mydata$year, mydata$month, mydata$day, sep="-"))
form.dates
```

```
 [1] "2014-01-15 UTC" "2014-01-15 UTC" "2014-01-16 UTC" "2014-01-16 UTC"
 [5] "2014-01-17 UTC" "2014-01-17 UTC" "2014-01-18 UTC" "2014-01-18 UTC"
 [9] "2014-01-19 UTC" "2014-01-19 UTC" "2014-01-20 UTC" "2014-01-20 UTC"
[13] "2014-01-21 UTC" "2014-01-21 UTC" "2014-01-22 UTC" "2014-01-22 UTC"
[17] "2014-01-23 UTC" "2014-01-23 UTC"
```

```
str(form.dates)
```

```
 POSIXct[1:18], format: "2014-01-15" "2014-01-15" "2014-01-16" "2014-01-16"
...
```

We can use this vector to create a new column in `mydata`

```
mydata$date<-form.dates
str(mydata)
```

```
'data.frame':   18 obs. of  7 variables:
 $ year  : int  2014 2014 2014 2014 2014 2014 2014 2014 2014 2014 ...
 $ month : int  1 1 1 1 1 1 1 1 1 1 ...
 $ day   : int  15 15 16 16 17 17 18 18 19 19 ...
 $ hour  : int  13 22 14 23 15 0 16 1 17 2 ...
```

```
 $ minute: int  45 0 45 0 45 0 45 0 45 0 ...
 $ second: int  34 0 34 0 36 0 38 0 40 0 ...
 $ date  : POSIXct, format: "2014-01-15" "2014-01-15" ...
```

Using `ymd_hms()` to format date-times

Let's create a second column that includes both date and time, both properly formatted.

Use the values of year, month, day, hour, minute, and secon to create a character string that is formatted as "YYYY-MM-DD-HH-mm-SS", wrap it in the `ymd_hms()` function and create a new column in `mydata`. .

```
mydata$datetime<-ymd_hms(paste(mydata$year, mydata$month, mydata$day,
mydata$hour, mydata$minute, mydata$second, sep="-"))
str(mydata)
```

```
'data.frame':   18 obs. of  8 variables:
 $ year    : int  2014 2014 2014 2014 2014 2014 2014 2014 2014 2014 ...
 $ month   : int  1 1 1 1 1 1 1 1 1 1 ...
 $ day     : int  15 15 16 16 17 17 18 18 19 19 ...
 $ hour    : int  13 22 14 23 15 0 16 1 17 2 ...
 $ minute  : int  45 0 45 0 45 0 45 0 45 0 ...
 $ second  : int  34 0 34 0 36 0 38 0 40 0 ...
 $ date    : POSIXct, format: "2014-01-15" "2014-01-15" ...
 $ datetime: POSIXct, format: "2014-01-15 13:45:34" "2014-01-15 22:00:00" ...
```

There are lots of other ways to format dates in R; some are more efficient given the particular format of the original data. Here's a link to a good overview of some of the common approaches:

http://biostat.mc.vanderbilt.edu/wiki/pub/Main/ColeBeck/datestimes.pdf

Other Handy Functions in `lubridate`

Time Zones

```
head(mydata$datetime)
```

```
[1] "2014-01-15 13:45:34 UTC" "2014-01-15 22:00:00 UTC"
[3] "2014-01-16 14:45:34 UTC" "2014-01-16 23:00:00 UTC"
[5] "2014-01-17 15:45:36 UTC" "2014-01-17 00:00:00 UTC"
```

At the moment, the `datetime` column is in UTC, or Coordinated Universal Time because it's the default when lubridate parses dates. To change this, you can use the function `force_tz()`:

```
mydata$datetime<-force_tz(mydata$datetime, tzone="EST") # Set as Eastern
Standard Time
mydata$datetime
```

```
 [1] "2014-01-15 13:45:34 EST" "2014-01-15 22:00:00 EST"
 [3] "2014-01-16 14:45:34 EST" "2014-01-16 23:00:00 EST"
 [5] "2014-01-17 15:45:36 EST" "2014-01-17 00:00:00 EST"
```

```
 [7] "2014-01-18 16:45:38 EST" "2014-01-18 01:00:00 EST"
 [9] "2014-01-19 17:45:40 EST" "2014-01-19 02:00:00 EST"
[11] "2014-01-20 18:45:42 EST" "2014-01-20 03:00:00 EST"
[13] "2014-01-21 19:45:44 EST" "2014-01-21 04:00:00 EST"
[15] "2014-01-22 20:45:46 EST" "2014-01-22 05:00:00 EST"
[17] "2014-01-23 21:45:48 EST" "2014-01-23 06:00:00 EST"
```

Set time zone based on location:

```
mytime<-ymd_hms("2015-08-14-05-30-00", tz="America/Halifax")
mytime
```

```
[1] "2015-08-14 05:30:00 ADT"
```

Check corresponding time in another location:

```
with_tz(mytime, "America/Vancouver")
```

```
[1] "2015-08-14 01:30:00 PDT"
```

Extracting Information From Date-Times

Extract the minute information from `mytime`

```
minute(mytime)
```

```
[1] 30
```

Change the minute to 34

```
minute(mytime)<-34
mytime
```

```
[1] "2015-08-14 05:34:00 ADT"
```

Find out the weekday a date falls on

```
wday(mytime) # As a number
```

```
[1] 6
```

```
wday(mytime, label=TRUE) # As a name
```

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

Or the month

```
month(mytime, label=TRUE)
```

```
[1] Aug
12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

Time Intervals

Set up two dates

```
date1<-ymd_hms("2011-09-23-03-45-23")
date2<-ymd_hms("2011-10-03-21-02-19")
```

How much time has passed?

```
difftime(date2,date1)

Time difference of 10.72009 days

difftime(date2, date1, unit="mins") # In minutes

Time difference of 15436.93 mins

difftime(date2, date1, unit="secs") # In seconds

Time difference of 926216 secs
```

Leap Years

Check to see if a year was a leap year

```
leap_year(2011)

[1] FALSE

leap_year(2012)

[1] TRUE
```