Rbit005

6/21/2014

Elgin Perry

eperry@chesapeake.net

**Date and Date-Time Objects**

In this Rbit, we look at processing dates. In an ASCII data file, dates are typically stored as character strings such as “6/21/2014”, “21Jun2014”, or “2014-06-21”. Of these, R likes the last one best. The character string representation is not conducive to mathematical operations such as computing a time difference. Even simple data management tasks such a sorting data may not work as expected using the character string representation. To resolve these issues, most computer packages represent dates internally as the number of days from a fixed point in time. Date-time variables are represented as seconds from a point in time. Along with these internal representations are special formatting tools that convert the internal numeric date to a conventional character string representation when you ask the package to display the date. The base R-language uses this strategy as well as some other options for dealing with dates. In addition there are add-on packages that offer more tools for processing dates. In this Rbit, I look only at the tools in base-R. I encourage you to the add on packages such as ‘chron’, ‘date’, and ‘timeDate’ if you need more than you find here.

The base r-language has one type of date object of class ‘’Date” and two types of date-time objects of classes POSIXct and POSIXlt. I have no idea what those names mean. The “Date” class is number of days from a fixed point in time. The “POSIXct” class is a number of seconds from a fixed point in time. The POSIXlt class is a ‘list’ object. We have not yet talked about lists. We will deal with this as a special case and generalize about lists later.

We begin with class “Date”. Here we create a character string date using R’s favorite date format, convert it to a date using as.Date(), ask R to show the converted variable, and look at the internal numeric representation of the converted variable.

d1.char <- '2014-06-21'

d1.char

class(d1.char)

d1 <- as.Date(d1.char)

d1

class(d1)

as.numeric(d1)



Note that when R displays d1.char it looks the same as the display of d1, but the class() of these two objects is different. When we use as.numeric() on d1 R displays the internal numeric value of d1 which is the number of days since 1970-01-01. When we use as.numeric() on d1.char R basically says it doesn’t know what we are asking for and returns an NA.

If the character version of the date in not in R’s favorite format, you will have to provide a format descriptor. Here are some examples.



If your character date has delimiters such as “/” or “-“, these are placed between the format descriptors. Note that “Y” is for a 4 digit year while “y” is for 2 digit year. For all format descriptors see ?strptime.

Once you have dates converted to the internal numeric representation, you can use operators with dates. Here are examples:



For a full list of operators see ?Ops.Date. That’s all for the class ‘dates’.

Now I turn to the date-time objects of classes POSIXct and POSIXlt. I mentioned above the POSIXlt is a list object. For now, just think of a list as a sequence of elements similar to a vector. The difference is that for a vector, the elements must be all the same class such as all numeric or all character. In a list, the elements can have different classes. A list can have elements that are a mixture of numeric and character. Even complex objects such as linear model objects can be stored in a list. The POSIXlt object is created with strptime(). Here we create the object, ask R to show it to us, and then using unlist(), we ask R to show all of the elements of the list.

dt1.char<-"2014-06-21 12:31:24"

dt1.char

dt1 <- strptime(dt1.char,"%Y-%m-%d %H:%M:%S")

dt1

unlist(dt1)



We see that dt1 is a list with 11 elements. Most of these are self-explanatory. Some things are unusual. Months (mon) are numbered 0-11 so that June is mon=5. The element year is the year since 1900. The element wday is day-of-the-week numbered 0-6. The element yday is day of the year. The element gmtoff is the offset in minutes from Greenwich Mean Time, but I’m not sure why it is NA in this case.

To extract elements from a list, use a double bracket notation ([[ ]]) instead of the single bracket notation ([ ]) used with vectors. To extract yday from this list do this:

# extract an element of the POSIXlt list

doy <- dt1[[8]]

doy

#or

doy <- dt1$yday

doy



The class POSIXct is internally the number of seconds since the beginning of 1970 (in the UTC time zone). As far as I know, it is only possible to get to POSIXct through POSIXlt. For example, nesting strptime() within as.POSIXct():

dt2.char<-"2014-06-21 12:31:34"

dt2.char

dt2 <- as.POSIXct(strptime(dt2.char,"%Y-%m-%d %H:%M:%S"))

dt2

as.numeric(dt2)

unlist(dt2)



It is surprising to me, but it seems that R can deal with mixing POSIXlt and POSIXct in operations.

class(dt1)

class(dt2)

dt2-dt1

dt2 > dt1



It might have something to do with this class POSIXt which is common to both, but I prefer to think it’s magic.

The last thing I will cover here is reading data from a file and coverting dataframe columns to data class columns. For this I will use a 5-day subset of the Mattawoman creek ConMon data which has been distributed with the file name ‘MAT\_5day.csv’. Read the data as usual

datafile <- paste(ProjRoot,"MAT\_5day.csv",sep='');

mat <- read.table(datafile, header=TRUE, sep=",", na.strings="NA", dec=".", strip.white=TRUE,stringsAsFactors = FALSE)

mat[1:10,]



We see that the dataframe has three columns with date-time information, Date, Time, and Date.Time. To add an object with a datetime class to the data frame, it would be simplest to convert Date.Time. However, to show some power of R, I will illustrate this using the Date and Time columns. Note that in the examples above, the arguments of the date function were always single dates. Here we show that these functions handle vector arguments by converting the entire column of the data frame with one command.

mat$date.time <- strptime(paste(mat$Date,mat$Time),"%m/%d/%Y %H:%M")

mat[1:10,]



Here are some functions for extracting parts of the data-time object.

# extracting parts of date.time object

as.Date(mat$date.time[1:5])

months(mat$date.time[1:5])

months(mat$date.time[1:5],abbreviate=TRUE)

years(mat$date.time[1:5])

library(chron) #date functions

years(mat$date.time[1:5])



Note that months() is a function in base-R. It would seem logical that years() would also be a function in base-R, but on first attempt R just says in can’t find years(). When I load the ‘chron’ library, which I have previously installed, then a years() function is available. This just one example of a useful extension to base-R provided by an add-on package.

I hope this is enough to get you started on using date and date-time objects.