Introduction to the mondate Package

Dan Murphy

2015-11-11

Base R provides two broad classes for representing dates-- POSIXt and Date. Objects of these classes conform to international standards[[1]](#footnote-1) by marking a calendar "day" with the instant of time that begins the day; i.e., 00.00. It is straightforward to calculate elapsed time in units of "days" using those objects. However, business-use cases often express elapsed time in units of business "months" or "years" and the definition of "month" or "year" in units of calendar "days" does not unambiguously satisfy this business requirement.

The "mondate" package adopts a different perspective. It is based on three principles:

1. A "month" is marked with the instant in time that separates that month from the following month.
2. A "day" is marked with the instant in time that separates that day from the following day.
3. A "year" equals twelve "months" *by definition.*

mondate objects constructed according to these principles make it straightforward to calculate elapsed time in units of business "months" and "years."

The purpose of this paper is to demonstrate the usefulness of the "mondate" package in everyday and business situations that express time in units of months or years. Technical looks "under the mondate “hood" will only be touched on for clarification.

The four major benefits of the mondate package are:

1. Date Aging
2. Date Formatting
3. Date Sequencing
4. Date Cutting

## 1. Date Aging

The "age" of an event plays many important roles in business use cases. By definition, Date objects are measured in units of "days" and POSIXt objects in units of "seconds". But sometimes it is more convenient to measure elapsed time in units of "months" or "years". This is where mondate comes in.

#### Example 1

If my "birth event" took place on February 29, 1996 then my age on February 28, 2006 was 10:

require(mondate)

## Loading required package: mondate  
##   
## Attaching package: 'mondate'  
##   
## The following object is masked from 'package:base':  
##   
## as.difftime

YearsBetween("1996-02-29", "2006-02-28")

## [1] 10  
## attr(,"timeunits")  
## [1] "years"

or in the US

YearsBetween("2/29/1996", "2/28/2006")

## [1] 10  
## attr(,"timeunits")  
## [1] "years"

and

MonthsBetween("2/29/1996", "2/28/2006")

## [1] 120  
## attr(,"timeunits")  
## [1] "months"

which also results when subtracting two mondates

m1 <- mondate.ymd(1996, 2)  
m2 <- mondate.ymd(2006, 2)  
m2 - m1

## Time difference of 120 months

#### Example 2

Suppose for Accounts Receivable aging, ABC Company has a policy of recognizing all invoices to have been sent on the last day of the month. This code calculates the age of all November 2015 invoices as of the end of the year and the end of the following year:

asof <- mondate.ymd(2015:2016)  
ages <- asof – mondate(“2015-11-30”)  
print(ages)

## Time differences in months  
## [1] 2 14

#### Example 3

The last example in this section is actuarial in nature. Suppose ABC Insurance Co. stores the date of insured losses in the variable (or data base field) DateOfLoss. Here are 10 random dates of loss after the end of 2010:

# generate 10 random dates after 2010  
set.seed(1)  
z <- rexp(10, .1)  
DateOfLoss <- as.Date(mondate.ymd(2010) + z)  
names(DateOfLoss) <- paste0("Claim", 1:10)  
print(DateOfLoss)

## Claim1 Claim2 Claim3 Claim4 Claim5   
## "2011-08-18" "2011-12-26" "2011-02-13" "2011-02-12" "2011-05-12"   
## Claim6 Claim7 Claim8 Claim9 Claim10   
## "2013-05-30" "2012-01-10" "2011-06-12" "2011-10-18" "2011-02-14"

Here are the four quarter-end "as-of dates" in 2013:

# Quarter-ends in 2013  
asof <- mondate.ymd(2013, 3 \* 1:4)  
names(asof) <- paste0("Q", 1:4)  
print(asof)

## Q1 Q2 Q3 Q4   
## 03/31/2013 06/30/2013 09/30/2013 12/31/2013

Comment:  
"names" were assigned to DateOfLoss and asof so that R will automatically embellish the data.frame below with row and column headers.

Here are the ages of the 10 losses as of each quarter end, stored in a data.frame with each claim's date of loss:

# a matrix of ages in units of months  
Ages <- round(sapply(asof, `-` , DateOfLoss), 1)  
# code ages as "not available" if the evaluation date preceeds   
# the Date of Loss (one instance)  
Ages[Ages <= 0] <- NA  
data.frame(DateOfLoss, Ages)

## DateOfLoss Q1 Q2 Q3 Q4  
## Claim1 2011-08-18 19.5 22.5 25.5 28.5  
## Claim2 2011-12-26 15.2 18.2 21.2 24.2  
## Claim3 2011-02-13 25.6 28.6 31.6 34.6  
## Claim4 2011-02-12 25.6 28.6 31.6 34.6  
## Claim5 2011-05-12 22.6 25.6 28.6 31.6  
## Claim6 2013-05-30 NA 1.1 4.1 7.1  
## Claim7 2012-01-10 14.7 17.7 20.7 23.7  
## Claim8 2011-06-12 21.6 24.6 27.6 30.6  
## Claim9 2011-10-18 17.5 20.5 23.5 26.5  
## Claim10 2011-02-14 25.5 28.5 31.5 34.5

### Date Arithmetic

mondates can act arithmetically in (almost always) the same way their underlying numeric can act.[[2]](#footnote-2) In particular, use subtraction to measure the magnitude of the interval between two dates in units of months.

For example, the following two calculations yield the same result:

mondate("2015-12-31") - mondate("2014-12-31")

## Time difference of 12 months

mondate("2015-12-31") - as.Date("2015-01-01")

## Time difference of 12 months

Why are the results identical even though the subtrahends would appear to be a day apart? The answer is that the two objects, as.Date("2015-01-01") and mondate("2014-12-31") *represent the same instant in time*, i.e., the moment that separates events occuring in 2014 from events occurring in 2015. This points out a new feature in mondate v1.0.

** Dates can be subracted directly from mondates.**

## 2. Date Formatting

mondate enables dates to be read and displayed in more than one format. The current built-in formats that are automatically recognized are

1. USa: "%m/%d/%Y"
2. USb: "%m-%d-%Y"
3. EUa: "%Y-%m-%d"
4. EUb: "%Y/%m/%d"

The order can be changed and new formats added using base::options for display ("writing") and set.mondate.displayFormats for "reading".

### "writing": dynamic format display

By default, mondate looks at your value of Sys.getlocale("LC\_TIME") at startup. If "United States" appears, then USa format is selected, otherwise, EUa is selected. This default can be changed globally for all mondate objects in the session or for individual objects.

#### Example 4

This vignette is being written in the US, so today's date, November 11, 2015, will be represented using the first format above by default:

mondate(Sys.Date())

## mondate: timeunits="months"  
## [1] 11/11/2015

The default can be changed to the international standard format[[3]](#footnote-3) "YYYY-MM-DD" using base::options and the name mondate.default.displayFormat:

options(mondate.default.displayFormat = "%Y-%m-%d")  
mondate(Sys.Date())

## mondate: timeunits="months"  
## [1] 2015-11-11

#### Example 5

French users of the format "dd/mm/YYYY" can establish that the default be:

options(mondate.default.displayFormat = "%d/%m/%Y")  
mondate(Sys.Date())

## mondate: timeunits="months"  
## [1] 11/11/2015

 The options approach modifies the default display format for **all** mondates in the R session. To set the display format for **just one** instance of a mondate object, use the displayFormat argument during the object's creation.

#### Example 6

Here we create the first 6 month-ends of 2015 to be displayed in the French format above despite the fact that the default format is first changed to the ISO standard:

options(mondate.default.displayFormat = "%Y-%m-%d")  
mondate(Sys.Date())

## mondate: timeunits="months"  
## [1] 2015-11-11

m <- mondate.ymd(2015, 1:6, displayFormat = "%d/%m/%Y")  
print(m)

## [1] 31/01/2015 28/02/2015 31/03/2015 30/04/2015 31/05/2015 30/06/2015

More creative formats can be used, for instance to display just the year and month, as was done in "Example 3" above.

### "reading": dynamic format detection

As previously mentioned, the mondate package has the four formats paste(get.mondate.displayFormats(), collapse=", ") for detecting whether a character string represents a "date." To inform mondate of another format for converting character to date, use set.mondate.displayFormats[[4]](#footnote-4) with the value(s) of your choice.

#### Example 7

To set the French format "dd/mm/yyyy" as Priority One for detecting dates, add that format to the *head* of the current list of detectable formats. The  
code below accomplishes that.[[5]](#footnote-5):

set.mondate.displayFormats(c("%d/%m/%Y",   
 get.mondate.displayFormats()),   
 clear = TRUE)

Continuing, suppose dates in a spreadsheet are saved to a csv file in France and the read.csv function results in this data.frame:

data <- data.frame(  
 cbind(Invoice=c("A", "B", "C"),  
 datechar = c("28/11/2015", "29/11/2015", "30/11/2015")))  
print(data)

## Invoice datechar  
## 1 A 28/11/2015  
## 2 B 29/11/2015  
## 3 C 30/11/2015

The character string of dates can be converted automatically to Date objects via mondate as follows

data$InvoiceDate <- as.Date(mondate(data$datechar))  
print(data)

## Invoice datechar InvoiceDate  
## 1 A 28/11/2015 2015-11-28  
## 2 B 29/11/2015 2015-11-29  
## 3 C 30/11/2015 2015-11-30

For more information on which codes to use when formatting dates, see the R help page for the strptime function. To add addtional defaults according to your value of Sys.getlocale("LC\_TIME"), contact the author[[6]](#footnote-6). (All are welcome to visit the package's public repository at <https://github.com/chiefmurph/mondate>.)

## 3. Date Sequencing

Sequences of dates in units of days or weeks is easily accomplished using the base R's Date class:

seq(as.Date("2015-11-01"), by = "day", length.out = 5)

## [1] "2015-11-01" "2015-11-02" "2015-11-03" "2015-11-04" "2015-11-05"

Month-sequences can similarly be generated with Dates, which does work well for most dates. Results can be disappointing, however, for dates near the end of the month. Compare these two sequences starting from the first and last days of January:

seq(as.Date("2015-01-01"), by = "month", length.out = 5)

## [1] "2015-01-01" "2015-02-01" "2015-03-01" "2015-04-01" "2015-05-01"

seq(as.Date("2015-01-31"), by = "month", length.out = 5)

## [1] "2015-01-31" "2015-03-03" "2015-03-31" "2015-05-01" "2015-05-31"

All dates in the first sequence are the first days of the month, but some dates in the second sequence "leak" into subsequent months. This behavior is well documented in R help[[7]](#footnote-7):

Using "month" first advances the month without changing the day: if this results in an invalid day of the month, it is counted forward into the next month

Perhaps the major purpose of the mondate package is to avoid this shortcoming.[[8]](#footnote-8)

#### Example 8

Sequences of month ends can be accomplished in various "mondate" ways. Here are two:

seq(mondate("2015-01-31"), by = "month", length.out = 5)

## mondate: timeunits="months"  
## [1] 2015-01-31 2015-02-28 2015-03-31 2015-04-30 2015-05-31

mondate.ymd(2015, 1:5)

## mondate: timeunits="months"  
## [1] 01/31/2015 02/28/2015 03/31/2015 04/30/2015 05/31/2015

The display format in the first sequence inherits from the format of the character representation of the beginning date. The display format in the second sequence is based on the author's locale (see "Date Formatting" section above). Also note that each of the objects generated above are of class "mondate".

 It is often more convenient to generate month sequences from Date objects, **and produce Date objects**, without having to resort to a mondate object in between. For that purpose the seqmondate generic function was written.

### seqmondate

seqmondate(x) generates sequences of class(x) for a variety of classes: Date, POSIXt, and mondate. For any other class(x) seqmondate(x) will produce a sequence of mondates, if possible. By default, 'by = "month"' is assumed.

#### Example 9

The first sequence below generates the same month-ends as in Example 8 but this time the objects generated are Dates.

(d <- seqmondate(as.Date("2015-01-01"), length = 5))

## [1] "2015-01-01" "2015-02-01" "2015-03-01" "2015-04-01" "2015-05-01"

class(d)

## [1] "Date"

#### Example 10: Year-ends

It has been said that there are always multiple ways to do things in R and this is no exception. Here are two ways to generate sequences of year-end dates.

seqmondate("2010-12-31", by = "year", length = 6)

## mondate: timeunits="months"  
## [1] 2010-12-31 2011-12-31 2012-12-31 2013-12-31 2014-12-31 2015-12-31

mondate.ymd(2010:2015)

## mondate: timeunits="months"  
## [1] 12/31/2010 12/31/2011 12/31/2012 12/31/2013 12/31/2014 12/31/2015

 If month- and year-end dates are intended to represent "as of" dates, it is preferable to create them as mondate objects rather than, say, Date objects if those dates will be used for "date aging" in units of months/years.

## 4. Date Cutting

**Sidebar on "cut" for numerics**  
A cut of a numeric 'x' is a collection of (half-open,half-closed] intervals that "cover" 'x'. By "cover" is meant that every value in 'x' is contained in some interval[[9]](#footnote-9), with the exception that R excludes the minimum value of 'x' by default.[[10]](#footnote-10) By default, the right endpoint is assumed to be closed.[[11]](#footnote-11)

A cut in R is represented by a factor. The cut function elegantly enunciates the numeric intervals by clearly identifying the (open,closed] borders in the factor's levels.

R Definition: A cut of a set of dates 'x' by "months" is a collection of contiguous months such that every date in x is contained in some month.

This correspondence between a date and its neighboring members in its 'cut' can be an important factor in the statistical analysis of events occurring during similar time periods.

There is a cut method for mondates when the 'breaks' argument is

* numeric and so determines the borders between intervals, or
* character and so identifies that the cover is to be a set of day-, week-, month-, year-, or quarter-intervals.

First we will define some cuts. Then we will see how one might use a cut.

#### Example 11

Because a mondate is fundamentally a numeric[[12]](#footnote-12), the following two commands -- the first on numeric, the second on mondate -- are fundamentally the same. The only difference is how the cuts' levels display.

cut(seq(from = 180.5, to = 185.5, by = .5), breaks = 180:186)

## [1] (180,181] (180,181] (181,182] (181,182] (182,183] (182,183] (183,184]  
## [8] (183,184] (184,185] (184,185] (185,186]  
## Levels: (180,181] (181,182] (182,183] (183,184] (184,185] (185,186]

cut(seq(from = mondate(180.5), to = mondate(185.5), by = .5), breaks = 180:186)

## [1] (12/31/2014,01/31/2015] (12/31/2014,01/31/2015]  
## [3] (01/31/2015,02/28/2015] (01/31/2015,02/28/2015]  
## [5] (02/28/2015,03/31/2015] (02/28/2015,03/31/2015]  
## [7] (03/31/2015,04/30/2015] (03/31/2015,04/30/2015]  
## [9] (04/30/2015,05/31/2015] (04/30/2015,05/31/2015]  
## [11] (05/31/2015,06/30/2015]  
## 6 Levels: (12/31/2014,01/31/2015] ... (05/31/2015,06/30/2015]

In the month intervals above, if one were to label the interval with one of the endpoints, it seems natural to choose the closed endpoint. That is the 'mondate' convention when 'breaks' is character. This bears repeating:

The 'mondate' convention is to label a *character cut* (breaks = "days", "months", ...) with the **closed endpoint** of the interval. As with cut.default, the closed endpoint is determined by the argument right: when TRUE the right endpoint labels the interval, when FALSE the left endpoint labels the interval.

We begin with examples of mondate cuts, with breaks being numeric and character.

#### Example 12

The following two commands generate the same cut. The first explicitly sets the break points with the month-ends beginning 2014-12-31 and ending six months later. The second implicitly sets the same break points. As with cut.default, the labels of the first cut clearly enunciate the (open,closed] monthly intervals. The labels of the second cut only display the closed endpoint.

cut(seq(mondate("2015-01-15"), mondate("2015-06-15"), by = .5),   
 breaks = mondate.ymd(2014) + 0:6)

## [1] (2014-12-31,2015-01-31] (2014-12-31,2015-01-31]  
## [3] (2015-01-31,2015-02-28] (2015-01-31,2015-02-28]  
## [5] (2015-02-28,2015-03-31] (2015-02-28,2015-03-31]  
## [7] (2015-03-31,2015-04-30] (2015-03-31,2015-04-30]  
## [9] (2015-04-30,2015-05-31] (2015-04-30,2015-05-31]  
## [11] (2015-05-31,2015-06-30]  
## 6 Levels: (2014-12-31,2015-01-31] ... (2015-05-31,2015-06-30]

cut(seq(mondate("2015-01-15"), mondate("2015-06-15"), by = .5), breaks = "month",  
 include.lowest = TRUE)

## [1] 2015-01-31 2015-01-31 2015-02-28 2015-02-28 2015-03-31 2015-03-31  
## [7] 2015-04-30 2015-04-30 2015-05-31 2015-05-31 2015-06-30  
## 6 Levels: 2015-01-31 2015-02-28 2015-03-31 2015-04-30 ... 2015-06-30

In the case that breaks is character it is unfortunate to have to set include.lowest = TRUE, opposite its default value FALSE.[[13]](#footnote-13) Other cut arguments as well have default values that may seem counterintuitive for cutting dates.

Perhaps the most troubling default is right = TRUE for Date objects because it violates the basic principle that Date objects begin on, and can be considered synonymous with, the instant beginning of the day, i.e., the *left* endpoint. For those and other reasons a new cutmondate generic exists in mondate v1.0.

 The cutmondate work on Date, mondate, and other objects with arguments that are more appropriate for their class.

Additionally, three new arguments were added to cut.mondate, which we will cover in due course.

We now turn our attention to the cutmondate methods.

### cutmondate

The 'cutmondate' collection of methods are most effective when 'breaks' defines a cover in terms of months or multiple months.

 When the object being cut is a Date or POSIXt, the breakpoints are assumed to begin the period, right = FALSE by default, and the levels are labeled by the first date in the period. If necessary, set labels to the right endpoint with right = TRUE.

#### Example 13

Here we regenerate the same DateOfLoss dates from Example 3, and cut them into month intervals.

set.seed(1)  
z <- rexp(10, .1)  
monDOL <- mondate.ymd(2010) + z  
DateOfLoss <- as.Date(monDOL)  
print(DateOfLoss)

## [1] "2011-08-18" "2011-12-26" "2011-02-13" "2011-02-12" "2011-05-12"  
## [6] "2013-05-30" "2012-01-10" "2011-06-12" "2011-10-18" "2011-02-14"

cutmondate(DateOfLoss)

## [1] 2011-08-01 2011-12-01 2011-02-01 2011-02-01 2011-05-01 2013-05-01  
## [7] 2012-01-01 2011-06-01 2011-10-01 2011-02-01  
## 28 Levels: 2011-02-01 2011-03-01 2011-04-01 2011-05-01 ... 2013-05-01

The "28 Levels" says that it takes 28 contiguous months to cover 'DateOfLoss'. Note that the levels are labeled with the first day of each month because in this case right=FALSE by default. Specify right=TRUE and the levels are labeled by the last day of the month, which occurs by default in the second, mondate case below:

cutmondate(DateOfLoss, right = TRUE)

## [1] 2011-08-31 2011-12-31 2011-02-28 2011-02-28 2011-05-31 2013-05-31  
## [7] 2012-01-31 2011-06-30 2011-10-31 2011-02-28  
## 28 Levels: 2011-02-28 2011-03-31 2011-04-30 2011-05-31 ... 2013-05-31

cutmondate(monDOL)

## [1] 08/31/2011 12/31/2011 02/28/2011 02/28/2011 05/31/2011 05/31/2013  
## [7] 01/31/2012 06/30/2011 10/31/2011 02/28/2011  
## 28 Levels: 02/28/2011 03/31/2011 04/30/2011 05/31/2011 ... 05/31/2013

Before tackling the final examples, it is important to point out three new arguments for cut.mondate (and therefore for cutmondate) that do not appear in cut.default or cut.Date:

* startmonth
* startyear
* attr.breaks = FALSE

See the help for cut.mondate for details behind these arguments. We will show use of the first and third.

### Fiscal Years

 The 'startmonth' argument enables *fiscal year cuts*!

#### Example 14

Suppose ABC's fiscal year is July 1 through June 30. The dates of loss in the previous examples can be cut into fiscal years by setting startmonth = 7.

Here we show two ways to cut DateOfLoss by fiscal year, the choice depending on whether the company identifies its fiscal year with the beginning day or ending day of the period.

cutmondate(DateOfLoss, breaks = "year", startmonth = 7)

## [1] 2011-07-01 2011-07-01 2010-07-01 2010-07-01 2010-07-01 2012-07-01  
## [7] 2011-07-01 2010-07-01 2011-07-01 2010-07-01  
## Levels: 2010-07-01 2011-07-01 2012-07-01

cutmondate(DateOfLoss, breaks = "year", startmonth = 7, right = TRUE)

## [1] 2012-06-30 2012-06-30 2011-06-30 2011-06-30 2011-06-30 2013-06-30  
## [7] 2012-06-30 2011-06-30 2012-06-30 2011-06-30  
## Levels: 2011-06-30 2012-06-30 2013-06-30

Continuing, suppose ABC Company refers to fiscal year by the first calendar year. (For example, FY 2015/2016 is referred to as the “2015 year.”) The dates can be cut and the appropriate labels automatically generated in the single function call

cutmondate(mondate(DateOfLoss, displayFormat = "%Y"),   
 breaks = "year", right = FALSE, startmonth = 7)

## [1] 2011 2011 2010 2010 2010 2012 2011 2010 2011 2010  
## Levels: 2010 2011 2012

In the final example we aggregate and plot data by fiscal year.

#### Example 15

ABC Insurance Co. records loss amounts associated with the dates of loss at regular intervals. Suppose the amounts as of 2016-06-30 are

(LossAmount <- round(rnorm(10, 1000, 100), -1))

## [1] 890 970 970 960 1030 910 1040 880 980 1040

ABC's actuaries want to aggregate loss by age. The C-Suite wants aggregations by fiscal year. Everyone wants visuals! No problem.

First, cut the loss dates into fiscal years (FY), but this time also use attr.breaks = TRUE.

 To make the break points available for subsequent date aging, set attr.breaks = TRUE in cutmondate.

FY <- cutmondate(mondate(DateOfLoss, displayFormat = "%Y"),   
 breaks = "year", right = FALSE, startmonth = 7,  
 attr.breaks = TRUE)  
asof <- mondate.ymd(2016, 6)  
age <- asof - attr(FY, "breaks")[FY]  
(data <- data.frame(DateOfLoss, LossAmount, FY, FYage = age))

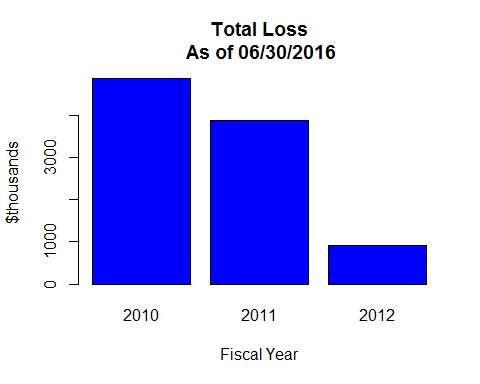
## DateOfLoss LossAmount FY FYage  
## 1 2011-08-18 890 2011 60 months  
## 2 2011-12-26 970 2011 60 months  
## 3 2011-02-13 970 2010 72 months  
## 4 2011-02-12 960 2010 72 months  
## 5 2011-05-12 1030 2010 72 months  
## 6 2013-05-30 910 2012 48 months  
## 7 2012-01-10 1040 2011 60 months  
## 8 2011-06-12 880 2010 72 months  
## 9 2011-10-18 980 2011 60 months  
## 10 2011-02-14 1040 2010 72 months

Calculate loss totals -- here we use aggregate -- and plot those totals -- here we use base R graphics. The first plot is by FY, the second by FY age.

(LossByFY <- aggregate(LossAmount ~ FY, data, sum))

## FY LossAmount  
## 1 2010 4880  
## 2 2011 3880  
## 3 2012 910

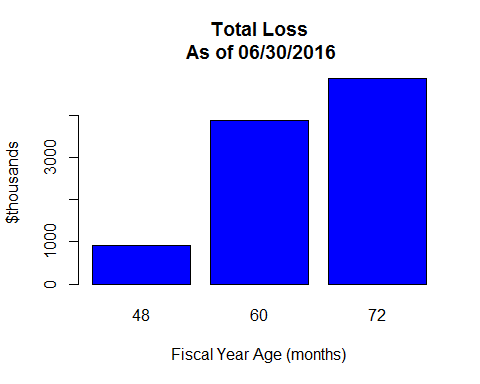
barplot(LossByFY$LossAmount, names.arg = LossByFY$FY,  
 ylab = "$thousands",  
 xlab = "Fiscal Year",  
 main = paste("Total Loss\n As of", asof),  
 col = "blue")



# and  
(LossByFYage <- aggregate(LossAmount ~ FYage, data, sum))

## FYage LossAmount  
## 1 48 months 910  
## 2 60 months 3880  
## 3 72 months 4880

barplot(LossByFYage$LossAmount, names.arg = LossByFYage$FYage,  
 ylab = "$thousands",  
 xlab = "Fiscal Year Age (months)",  
 main = paste("Total Loss\n As of", asof),  
 col = "blue")



## Summary

The mondate package represents dates in a way that enables date aging and sequencing in a mathematically "invertible" manner.[[14]](#footnote-14)

A mondate object is not appropriate for all situations. For example, a mondate halfway through the month of February falls on the close of business of the 14th day (in most cases) but falls on the 15th day for April. If a time period other than month or year is more suited to the situation, use an R object other than mondate.

Factors that associate similar events by date can be created by R's cut methods. A cut method exists for mondate objects but the preferable function to use for cutting those and other date-representing objects (Date, POSIXt) is cutmondate because the arguments default to values intuitively appropriate for the object. The "startmonth" arguments allows the creation of fiscal year cuts.

## Thank you

Many thanks to the R Development team for their work on Date and POSIXt objects and methods.

A special thank you goes out to Gabor Grothendieck for his suggestion of, and help with, cut.mondate.

Finally, the "mondate perspective" was motivated by Damien Laker in his somewhat obscure 2008 paper *Time Calculations for Annualizing Returns: The Need for Standardization*[[15]](#footnote-15) where he states the obvious :-)

"Annualization calculations based on whole months never wind up accidentally calculating that a year is anything other than a year long."

Mr. Laker's *Recommended Method* is based on two cases:[[16]](#footnote-16)

1. "For any period that starts and finishes on the last day of a month, the time calculation can be done entirely in months."
2. "In cases where the start date or end date is not the last day of a month, it will be necessary to count a partial month."

The mondate package embraces this end-of-business, month-centric, a-year-equals-twelve-months perspective.

1. Refer to <http://www.iso.org/iso/home/standards/iso8601.htm> [↑](#footnote-ref-1)
2. The underlying numeric measures the number of months since the close of business 1999-12-31 (".mondate.origin"). The reader is encouraged to experiment with mondate arithmetic. Please inform the package maintainer if the results are not intuitive. [↑](#footnote-ref-2)
3. ibid. [↑](#footnote-ref-3)
4. This function sets the options value of mondate.displayFormats [↑](#footnote-ref-4)
5. This example is given in ?set.mondate.displayFormats [↑](#footnote-ref-5)
6. chiefmurphy at gmail [↑](#footnote-ref-6)
7. see ?seq.POSIXt [↑](#footnote-ref-7)
8. "Under the hood" mondate represent dates relative to the percent of the month that has transpired by the close of business that day. Kudos to Damien Laker! See the "Thank You" section at the end. [↑](#footnote-ref-8)
9. thus, not an "open cover" in the topological sense [↑](#footnote-ref-9)
10. unless you set include.lowest = TRUE [↑](#footnote-ref-10)
11. change the interval to left-closed by setting right = FALSE [↑](#footnote-ref-11)
12. the "mondate class" is defined via setClass("mondate", contains = "numeric", etc. [↑](#footnote-ref-12)
13. Excluding the minimum value of 'x' would be somewhat *random* -- forgive the colloquialism -- given that other values of 'x' are likely to in the same time interval. In the case of character breaks, include.lowest=FALSE throws an error. [↑](#footnote-ref-13)
14. Invertible in the sense that retracing a monthly sequence from the end should produce the same sequence but in reverse order. That is not always possible with Date sequences by "month". Consider that  
    seq(as.Date("2015-01-31"), by = "month", length.out = 2)  
    yields  
    "2015-01-31" "2015-03-03"  
    whereas  
    seq(as.Date("2015-03-03"), by = "-1 month", length.out = 2)  
    yields  
    "2015-03-03" "2015-02-03" [↑](#footnote-ref-14)
15. The Journal of Performance Measurement, Summer 2008 [↑](#footnote-ref-15)
16. Ibid. Although the terms "start on" and "finishes on" are not specifically defined in Mr. Laker's paper, the spirit is intuitively understood and reflected in the package. [↑](#footnote-ref-16)