Working with Unspecified, Approximate, Uncertain, Sets and Ranges of Dates with messydates

James Hollway
Graduate Institute of
International and Development Studies

Henrique Sposito
Graduate Institute of
International and Development Studies

Abstract

This paper presents the **messydates** package for R, which facilitates working with 'messy' dates. Messy dates are common when studying historical and sometimes even current phenomena, and can create various technical problems for the data analyst. The paper highlights these problems and offers practical advice on how to solve them using **messydates**. The paper also introduces a conceptual framework for resolving messydates into more familiar date classes in R ready for analysis.

Keywords: dates, ISO, R.

1. Introduction

Dates are often messy. Whether historical (or ancient), future, or even recent, we often only know approximately when an event occurred, that it happened within a particular period, an unreliable source means a date should be flagged as uncertain, or sources offer multiple, competing dates.

messydates implements for R the Extended Date/Time Format (EDTF) annotations set by the International Organization for Standardization (ISO) outlined in ISO 8601-2_2019(E). The standardised extended format allow for unambiguous interpretation of dates and guarantee interoperability. These include notation for:

- unspecified date(component)s, e.g. `2012-XX-01` for the first of some unknown month in 2012 or `2012-01` for some unknown day in January 2012
- approximate date(component)s, e.g. `2012-01-12~` for approximately the 12th of Jan-

uary 2012
- uncertain date(component)s, e.g. `2012-01-12?` where this data point is based on an unreliable source
- sets of dates, e.g. `{2012-01-01,2012-01-12}` where the date can be both 1 January 2012 and 12 January 2012
- ranges of dates, e.g. `2012-01-01..2012-01-12` for all dates between the 1 January 2012 and 12 January 2012 inclusive

messydates contains a set of tools for constructing and coercing into and from the `mdate` class. This date class allows regular dates to be annotated to express unspecified date components, approximate or uncertain date components, date ranges, and sets of dates.

The package also includes a function for unpacking or expanding sets or ranges of dates into all dates consistent with how the date or set of dates is specified or annotated. Methods are also offered that can be used to make explicit how researchers convert date imprecision into precise dates for analysis, such as getting the `min()`, `max()`, or even a `random()` date from among the dates consistent with a set or range of dates. This greatly facilitates research transparency as well as robustness checks.

1.1. Motivation

As researchers, we often recognize this messiness but are forced to force non-existent precision on data so we can proceed with analysis. For example, if we only know something happened in a given month or year, we might just opt for the start of that month (e.g. `2021-07-01`) or year (`2021-01-01`), assuming that to err on the earlier (or later) side is a justifiable bias. However, this can create issues for inference in which sequence or timing is important. The goal of **messydates** is to help with this problem by retaining and working with various kinds of date imprecision.

1.2. Relationship to other packages

messydates offers a new date class, but one that comes with methods for converting from and into common date classes such as `Date`, `POSIXct`, and `POSIXlt`. It is thus fully compatible with packages such as **lubridate** (Grolemund and Wickham 2011) and **anytime** (Eddelbuettel 2019).

2. R code

2.1. A new class

messydates contains a set of tools for constructing and coercing into and from the `mdate` class. This date class implements ISO 8601-2:2019(E) and allows regular dates to be annotated to express unspecified date components, approximate or uncertain date components, date ranges, and sets of dates. The function `as_messydate()` handles the coercion to `mdate` class.

```
R> tibble::tribble(~Example, ~Date,
+                  "A normal date", as.character(Sys.Date()),
```

```
+                "A future date", "2599-12-31",
+                "A written date", "This is the first day of February, two thousand and t
+                "A historical date", "476",
+                "An era date", "33 BC",
+                "An approximate date", "2012-01-12~",
+                "An uncertain date", "2001-01-01?",
+                "An unspecified date", "2012-01",
+                "A censored date", "..2012-01-12",
+                "A range of dates", "2019-11-01:2020-01-01",
+                "A set of dates", "2021-5-26, 2021-11-19, 2021-12-4") %>%
+   dplyr::mutate(base = as.Date(Date),
+                lubridate = lubridate::as_date(Date),
+                anytime = anytime::anydate(Date),
+                messydates = messydates::as_messydate(Date))
```

```
# A tibble: 11 x 6
   Example             Date         base        lubridate  anytime    messydates
   <chr>               <chr>        <date>      <date>     <date>     <mdate>
 1 A normal date       2022-07-15   2022-07-15  2022-07-15 2022-07-15 2022-07-1~
 2 A future date       2599-12-31   2599-12-31  2599-12-31 2599-12-31 2599-12-3~
 3 A written date      This is the ~ NA         NA         NA         2021-02-0~
 4 A historical date   476          NA          NA         NA         0476      ~
 5 An era date         33 BC        NA          NA         NA         -0033     ~
 6 An approximate date 2012-01-12~  2012-01-12  2012-01-12 2012-01-12 2012-01-1~
 7 An uncertain date   2001-01-01?  2001-01-01  2001-01-01 2001-01-01 2001-01-0~
 8 An unspecified date 2012-01      NA          2020-12-01 2012-01-01 2012-01   ~
 9 A censored date     ..2012-01-12 NA          2012-01-12 NA         ..2012-01~
10 A range of dates    2019-11-01:2~ 2019-11-01 2019-11-01 2019-11-01 2019-11-0~
11 A set of dates      2021-5-26, 2~ 2021-05-26 NA         2021-05-26 {2021-05-~
```

2.2. Annotate

Some datasets have, for example, an arbitrary cut off point for start and end points, but these are often coded as precise dates when they are not necessarily the real start or end dates. The annotate functions helps annotate uncertainty and approximation to dates. Inaccurate start or end dates can be represented by an affix indicating "on or before", if used as a prefix (e.g. `..1816-01-01`), or indicating "on or after", if used as a suffix (e.g. `2016-12-31..`). Approximate dates are indicated by adding a `~` to year, month, or day components, as well as groups of components or whole dates to estimate values that are possibly correct (e.g. `2003-03-03~`). Day, month, or year, uncertainty can be indicated by adding a `?` to a possibly dubious date (e.g. `1916-10-10?`) or date component (e.g. `1916-?10-10`).

```
R> tibble::tibble(Beg = as_messydate(c("1816-01-01", "1916-01-01", "2016-01-01")),
+                End = as_messydate(c("1816-12-31", "1916-12-31", "2016-12-31"))) %>%
+   dplyr::mutate(on_or_before = ifelse(Beg <= "1816-01-01", on_or_before(Beg), Beg),
+                on_or_after = ifelse(End >= "2016-01-01", on_or_after(End), End),
```

```
+                    as_approximate = ifelse(End >= "2016-01-01", on_or_after(End), End),
+                    as_uncertain = ifelse(End == "1916-12-31", as_uncertain(End), End))

# A tibble: 3 x 6
  Beg        End         on_or_before on_or_after as_approximate as_uncertain
  <mdate>    <mdate>     <chr>        <chr>       <chr>          <chr>
1 1816-01-01 1816-12-31 ..1816-01-01 1816-12-31  1816-12-31     1816-12-31
2 1916-01-01 1916-12-31 1916-01-01   1916-12-31  1916-12-31     1916-12-31?
3 2016-01-01 2016-12-31 2016-01-01   2016-12-31.. 2016-12-31..  2016-12-31
```

2.3. Expand

Expand functions transform date ranges, sets of dates, and unspecified or approximate dates (annotated with '..', '{ , }', 'XX' or '~') into lists of dates. As these dates may refer to several possible dates, the function "opens" these values to include all the possible dates implied.

```
R> dates_expand <- as_messydate(c("2001-01-01", "2001-01", "2001-01-01..2001-02-01",
+                        "{2001-01-01,2001-02-01}", "2001-XX-01"))
R> expand(dates_expand)

[[1]]
[1] "2001-01-01"

[[2]]
 [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
 [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[11] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[16] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"
[21] "2001-01-21" "2001-01-22" "2001-01-23" "2001-01-24" "2001-01-25"
[26] "2001-01-26" "2001-01-27" "2001-01-28" "2001-01-29" "2001-01-30"
[31] "2001-01-31"

[[3]]
 [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
 [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[11] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[16] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"
[21] "2001-01-21" "2001-01-22" "2001-01-23" "2001-01-24" "2001-01-25"
[26] "2001-01-26" "2001-01-27" "2001-01-28" "2001-01-29" "2001-01-30"
[31] "2001-01-31" "2001-02-01"

[[4]]
[1] "2001-01-01" "2001-02-01"

[[5]]
 [1] "2001-01-01" "2001-02-01" "2001-03-01" "2001-04-01" "2001-05-01"
```

```
 [6] "2001-06-01" "2001-07-01" "2001-08-01" "2001-09-01" "2001-10-01"
[11] "2001-11-01" "2001-12-01"
```

2.4. Contract

The `contract()` function operates as the opposite of `expand()`. It contracts a list of dates into the abbreviated annotation of **messydates**.

```
R> contract(expand(dates_expand))
```

```
 'mdate' chr [1:5] "2001-01-01" "2001-01" "2001-01-01..2001-02-01" ...
```

2.5. Coerce from messydates

Coercion functions coerce objects of `mdate` class to common date classes such as `Date`, `POSIXct`, and `POSIXlt`. Since `mdate` objects can hold multiple individual dates, an additional function must be passed as an argument so that multiple dates are "resolved" into a single date. For example, one might wish to use the earliest possible date in a range, or set, of expanded dates (`min`), or the latest possible date (`max`), or some notion of a central tendency (`mean`, `median`, or `modal`), or even a `random` selection from amongst the candidate dates.

```
R> tibble::tibble(min = as.Date(dates_expand, min),
+                 max = as.Date(dates_expand, max),
+                 median = as.Date(dates_expand, median),
+                 mean = as.Date(dates_expand, mean),
+                 modal = as.Date(dates_expand, modal),
+                 random = as.Date(dates_expand, random))
```

```
# A tibble: 5 x 6
  min        max        median     mean       modal      random
  <date>     <date>     <date>     <date>     <date>     <date>
1 2001-01-01 2001-01-01 2001-01-01 2001-01-01 2001-01-01 2001-01-01
2 2001-01-01 2001-01-31 2001-01-16 2001-01-16 2001-01-01 2001-01-24
3 2001-01-01 2001-02-01 2001-01-17 2001-01-16 2001-01-01 2001-01-03
4 2001-01-01 2001-02-01 2001-02-01 2001-01-16 2001-01-01 2001-01-01
5 2001-01-01 2001-12-01 2001-07-01 2001-06-16 2001-01-01 2001-09-01
```

2.6. Additional functionality

Several other functions are also offered in the **messydates** package.

For example, one can check various logical tests for messy date objects. `is_messydate()` tests whether the object inherits the `mdate` class. `is_intersecting()` tests whether there is any intersection between two messy dates. `is_element()` similarly tests whether a messy date can be found within a messy date range or set. `is_similar()` tests whether two dates contain similar components. `is_precise()` tests for whether date is precise.

```
R> is_messydate(as_messydate("2001-01-01"))
```

```
[1] TRUE
```

```
R> is_messydate(as.Date("2001-01-01"))
```

```
[1] FALSE
```

```
R> is_intersecting(as_messydate("2001-01"), as_messydate("2001-01-01..2001-02-22"))
```

```
[1] TRUE
```

```
R> is_intersecting(as_messydate("2001-01"), as_messydate("2001-02-01..2001-02-22"))
```

```
[1] FALSE
```

```
R> is_element(as_messydate("2001-01-01"), as_messydate("2001-01"))
```

```
[1] TRUE
```

```
R> is_element(as_messydate("2001-01-01"), as_messydate("2001-02"))
```

```
[1] FALSE
```

```
R> is_similar(as_messydate("2001-06-02"), as_messydate("2001-02-06"))
```

```
[1] TRUE
```

```
R> is_similar(as_messydate("2001-06-22"), as_messydate("2001-02-06"))
```

```
[1] FALSE
```

```
R> is_precise(as_messydate("2001-06-02"))
```

```
[1] TRUE
```

```
R> is_precise(as_messydate("2001-02"))
```

```
[1] FALSE
```

Additionally, one can perform intersection (`md_intersect()`) and union (`md_union()`) on, inter alia, messy date class objects. Or 'join' that retains all elements, even if duplicated, with `md_multiset`.

```
R> md_intersect(as_messydate("2001-01-01..2001-01-20"),as_messydate("2001-01"))
```

```
 [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
 [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[11] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[16] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"
```

```
R> md_union(as_messydate("2001-01-01..2001-01-20"),as_messydate("2001-01"))
```

```
 [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
 [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[11] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[16] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"
[21] "2001-01-21" "2001-01-22" "2001-01-23" "2001-01-24" "2001-01-25"
[26] "2001-01-26" "2001-01-27" "2001-01-28" "2001-01-29" "2001-01-30"
[31] "2001-01-31"
```

```
R> md_multiset(as_messydate("2001-01-01..2001-01-20"),as_messydate("2001-01"))
```

```
 [1] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
 [6] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[11] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[16] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"
[21] "2001-01-01" "2001-01-02" "2001-01-03" "2001-01-04" "2001-01-05"
[26] "2001-01-06" "2001-01-07" "2001-01-08" "2001-01-09" "2001-01-10"
[31] "2001-01-11" "2001-01-12" "2001-01-13" "2001-01-14" "2001-01-15"
[36] "2001-01-16" "2001-01-17" "2001-01-18" "2001-01-19" "2001-01-20"
[41] "2001-01-21" "2001-01-22" "2001-01-23" "2001-01-24" "2001-01-25"
[46] "2001-01-26" "2001-01-27" "2001-01-28" "2001-01-29" "2001-01-30"
[51] "2001-01-31"
```

As well, some arithmetic operations are available for messydates. For instance, one can add or subtract one year to all messy dates in a vector.

```
R> tibble::tibble(date = dates_expand,
+                 add = dates_expand + "1 day",
+                 subtract = dates_expand - "1 year")
```

```
# A tibble: 5 x 3
  date                   add                    subtract
  <mdate>                <mdate>                <mdate>
1 2001-01-01             2001-01-02             2000-01-02
2 2001-01                2001-01-02..2001-02-01 2000-01-02..2000-02-01
3 2001-01-01..2001-02-01 2001-01-02..2001-02-02 2000-01-02..2000-02-02
4 {2001-01-01,2001-02-01} {2001-01-02,2001-02-02} {2000-01-02,2000-02-02}
5 2001-XX-01             2001-XX-02             2000-XX-02
```

2.7. Case Study - 2001 Battles

Dates, even for some recent events, can be messy. Take the dates of battles in 2001 according to Wikipedia included in **messydates**. The dates of these battles are often approximate (i.e. the day in which a battle started or ended is unknown) or come from unreliable sources (i.e. the date might not be trustworthy).

```
R> battles <- messydates::battles
R> battles
```

```
# A tibble: 20 x 3
   Battle                             Date                 Parties
   <chr>                              <mdate>              <chr>
 1 Operation MH-2                     2001-03-08           MK-National Li~
 2 2001 Bangladesh-India border clashes 2001-04-16..2001-04-20 BD-ID
 3 Operation Vaksince                 2001-05-25           MK-National Li~
 4 Alkhan-Kala operation              2001-06-22..2001-06-28 RU-Chechen Rep~
 5 Battle of Vedeno                   2001-08-13..2001-08-26 RU-Chechen Ins~
 6 Operation Crescent Wind            2001-10-7..2001-12?  US/UK-Taliban
 7 Operation Rhino                    2001-10-19..2001-10-20 US-Taliban
 8 Battle of Mazar-e-Sharif           2001-11-09           US/Northern Al~
 9 Siege of Kunduz                    2001-11-11..2001-11-23 US/Northern Al~
10 Battle of Herat                    2001-11-12           US/Northern Al~
11 Battle of Kabul                    2001-11-13..2001-11-14 US/Northern Al~
12 Battle of Tarin Kowt               2001-11-13..2001-11-14 US/Eastern All~
13 Operation Trent                    2001-11-~15..2001-11-~30 US/UK-Taliban/~
14 Battle of Kandahar                 2001-11-22..2001-12-07 US/AU/Eastern ~
15 Battle of Qala-i-Jangi             2001-11-25..2001-12-01 US/UK/Northern~
16 Battle of Tora Bora                2001-12-12..2001-12-17 US/Northern Al~
17 Battle of Shawali Kowt             2001-12-03           US/Eastern All~
18 Battle of Sayyd Alma Kalay         2001-12-04           US/Eastern All~
19 Battle of Amami-Oshima             2001-12-22           JP-KP
20 Tsotsin-Yurt operation             2001-12-30..2002-01-03 RU-Chechen Ins~
```

messydates facilitates working with these dates as we can, for example, check which dates are precise, get the median values for imprecise dates, and find the longest battle in 2001.

```
R> messydates::is_precise(battles$Date)
```

```
 [1]  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE
[13] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE
```

```
R> as.Date(battles$Date, median)
```

```
 [1] "2001-03-08" "2001-04-18" "2001-05-25" "2001-06-25" "2001-08-20"
 [6] "2001-11-19" "2001-10-20" "2001-11-09" "2001-11-17" "2001-11-12"
[11] "2001-11-14" "2001-11-14" "2001-11-23" "2001-11-30" "2001-11-28"
[16] "2001-12-15" "2001-12-03" "2001-12-04" "2001-12-22" "2002-01-01"
```

```
R> as.numeric(as.Date(battles$Date, max) - as.Date(battles$Date, min))
```

```
 [1]  0  4  0  6 13 85  1  0 12  0  1  1 15 15  6  5  0  0  0  4
```

Getting the timing can be important for researchers, however, when faced with date imprecision, researchers usually have to chose between making arbitrary choices (e.g. adding "-01-01" to all incomplete dates) or work imprecise dates (e.g. keep the year only). Yet, both choices may lead to bias results. This is especially true if researchers' are looking to generate inferences. Let's assume, for instance, certain researcher is interested in the relationship between the United States (US) being a party to a conflict and the duration of the conflict in 2001. The researcher hypothesizes that conflicts involving the US have a shorter duration because the US has the most powerful military in the world. A relationship could be mediated by the number of parties involved in the conflict. Using **messydates**, we can create two different date variables in the battles data to be our dependent variables representing conflict time, one variable with an arbitrary cut of point and the other variable with random values for uncertain or approximate dates. The ability to resolve uncertain or approximate dates differently is particularly useful for checking the robustness of results. As our independent variable, we create a dummy variable for whether the US was involved in the conflict. As our control, we code the number of actors in the conflict. With these variables we build two linear regression models, one using the arbitrary dates and the other the random dates.

```
R> set.seed(1301)
R> battles <- battles %>%
+   mutate(arbitrary = as.numeric(as.Date(Date, max) - as.Date(Date, min)),
+          random = ifelse(grepl("\\?|\\~", Date),
+                          abs(as.Date(Date, random) - as.Date(Date, random)),
+                          arbitrary),
+          US_party = ifelse(grepl("US", Parties), 1, 0),
+          n_actors = c(2, 2, 2, 2, 2, 3, 2, 4, 4, 4, 3, 3, 4, 4, 5, 4, 3, 3, 2, 2))
R> lm(arbitrary ~ US_party + n_actors, battles)


Call:
lm(formula = arbitrary ~ US_party + n_actors, data = battles)

Coefficients:
(Intercept)      US_party      n_actors
      8.815        10.802        -2.479

R> lm(random ~ US_party + n_actors, battles)


Call:
lm(formula = random ~ US_party + n_actors, data = battles)

Coefficients:
(Intercept)      US_party      n_actors
      0.538        -1.410         1.660
```

Notice how the regression coefficients change when we pick random values within the range for the uncertain and approximate dates in the battles data, in comparison to setting arbitrary cut off points. Although not statistically significant, the coefficient for US being a party in a conflict change considerably, going from being positive to becoming negative, when we we use random values within that range. In this case, it is hard to say whether there is a positive or negative relationship between the US being a part of a battle in 2001 and the time of the conflict.

3. Acknowledgements

References

Eddelbuettel D (2019). "Anytime: Easier Date and Time Conversion." p. 4.

Grolemund G, Wickham H (2011). "Dates and Times Made Easy with Lubridate." **40**(3), 1–25. ISSN 1548-7660. doi:10.18637/jss.v040.i03. URL http://www.jstatsoft.org/v40/i03/.

Affiliation:

James Hollway
Graduate Institute of
International and Development Studies
Chemin Eugène-Rigot 2A
PO Box 1672
1211 Geneva 1
Switzerland
E-mail: james.hollway@graduateinstitute.ch
URL: http://jameshollway.com