dm

Henrik Renlund

April 16, 2018

Contents

1 About

`dm` is a package for functions relating to data management.

2 `dm` and related functions

2.1 The problem we want to solve

A statistical report sometimes build an *clinical data base* (CDB) from multiple sources, collects variables that might need to be renamed and (if categorical) recoded (and possibly transformed), the documentation of which is *significantly boring*.

The `dm` functions is an interactive-ish way of creating an CDB which both inspects the chosen variables and "documents" the process.

2.2 The elevator pitch

. . . assuming LaTeX

1. point to variables (from possibly different sources), one at the time, with `dm` (along with possible renaming, recoding and transformation). (Use `dm_find` to look for candidate variables.) This gives a summary of the variable pointed to[1], and the information is stored in a list somewhere.

2. create the CDB by `dm_create`.

3. get easy-to-print documentation of where variables came from (`dm_doc2latex`) and what recodings have been done (`dm_recode2latex`).

So, the point really is to get (3) "for free" in a way that is connected to the creation of the CDB.

2.3 The stuck-in-an-elevator pitch

If most variables are picked form the same source, this can be set in options.

```
opts_dm$set('default_db' = 'MyDataBase')
```

If that is done, `dm` only needs a `var` argument, the name of the var you want to add. But you can use

- `var`, name of variable in source

- `name`, optional, if you want a new name for the variable (else it is set to `var`)

- `db`, name of data frame (or similar) where `var` exists (else will look at the default location, if set)

- `recode`, a list that specifies the recoding. This is the `L` argument for the `recode` function that this package provides (see the help for that functions)

[1]Typically one wants to to this procedure anyway to sanity check all variables that are to be included.

- `transf` a function for transforming (this might be something like a character-to-date function like `ymd` from the lubridate package)
- `comment` if you want to keep some comment about the variable
- `label` if you want to give the variable a "label" (i.e. the value of the label attribute)
- `keep.label` if `var` already has a label in `db`, should this be kept? (only if no `label` is provided)

Then as `dm` is evaluated, information about the variable is printed (to see range, levels and such).

```
dm(var = 'gEndEr', name = 'gender', label = "Perceived Gender")
    ## is followed by information being printed
```

The information about the options is stored in a list (by default "dm_doc" in an environment "dm_envir").[2] The key is the 'name' element, so as long as that is not changed, you can rerun the function with new arguments if something went wrong

```
dm('gEndEr', 'gender', label = "Biological Gender") ## overwrites
    ## the 'gender' entry
```

Else, kill all documentation and start again

```
dm_doc(kill = TRUE, prompt = FALSE) ## or possibly kill only this
  ## entry dm:::dm_doc_set('gender', NULL)
```

The documentation can be accessed

```
myDoc <- dm_doc()
print(myDoc)  ## N.B not all information is printed
```

Once all variables are created you can either store the "documentation" (and point to it later) or go on to create the CDB with `dm_create`. Specify a set if individuals (vector of id's) and, if necessary a vector of how individuals are indentified in different data frames. If the `doc` argument is not provided it will just look in `dm_doc()`.

```
id_key = c('MyDataBase' = 'id', 'Other1' = 'ID', 'Other2' = 'idno')
CDB <- dm_create(set = MyDataBase$id, id.name = id_key)
```

Now you have an CDB and you can print `dm_doc()` to show where all variables come from. You can get all recodings from

[2]This is due to it begin poor for functions practice to write to objects in the global environment.

```r
lapply(dm_doc(), FUN = function(x) x$recode_table)
```

There are also convience functions `dm_recode2latex` and `dm_doc2latex` which will print all tables and documentation, respectively, in LaTeX format.

2.4 Toy Example

We create some toy data

```r
n <- 200
BL <- data.frame(
    id = structure(sprintf("id%d", 1:n),
                   label = "identification"),
    aalder = structure(round(rnorm(n, 50, 10)),
                       label = "Age at some time",
                       foo = 'whatever'),
    vikt = rpois(n, 50),
    gr = sample(c('A', 'A2', 'B', 'C', 'D', 'd1', 'unknown'),
                n, TRUE),
    koon = structure(sample(c('M', 'K'), n, T),
                     label = "The Sex"),
    nar = as.Date("2001-01-01") + runif(n, 0, 3650),
    stringsAsFactors = FALSE
)
BL$vikt[sample(1:n, 15)] <- NA
BL$gr[sample(1:n, 10)] <- NA
m <- .9*n
COMP <- data.frame(
    ID = structure(sample(BL$id, m),
                   label = "identification"),
    foo = rbinom(m, 1, .2),
    bar = structure(rexp(m, 1/150),
                    label = "Time passed")
)
```

There are some functions to help look for relevant variables.

```r
db_info(BL) ## prints names and 'label' attributes

##   source variable            label     class
## 1     BL       id   identification character
## 2     BL   aalder Age at some time   numeric
## 3     BL     vikt                    integer
## 4     BL       gr                  character
## 5     BL     koon          The Sex character
## 6     BL      nar                       Date
```

```
dm_find(pattern = 'time') ## looks in names and labels

## dm_find found:

##   source variable              label   class
## 2     BL   aalder Age at some time numeric
## 3   COMP      bar       Time passed numeric
```

Most variables of interest are in `BL` so set this as default.

```
opts_dm$set('default_db' = 'BL')
```

Next, we add the first variable (and view the output). We've chosen a variable with a fairly complex recoding to also illustrate the use the `recode` argument (also see the help for the `recode` function that is being utilized). `L` is a list where each entry has a vector of levels that will aquire the name of that entry, where the order of entries will be the order of the levels.

```
## 'gr' will be recoded in a more complex way
L <- list('A' = 'A2',
          'B' = NULL, ## placeholder to get the order right
          'CD' = c('C', 'D', 'd1'),
          'Unknown' = c('unknown', NA))
## # this would also work:
## L <- list('A' = c('A', 'A2'),
##           'B' = 'B',
##           'CD' = c('C', 'D', 'd1', 'something not in data'),
##           'Unknown' = c('unknown', NA))
dm('gr', recode = L, label = 'Group')

## ---------------------------------------------------------------
## Adding data base 'BL' entry 'gr' as variable 'gr'
## A variable of class: character
## There are 10 (5 percent) missing
##       and 7 (3.7 percent) unique values
## Since there are less than 20 unique vales we tabulate them:
##
##
##       A      A2       B       C       D      d1 unknown    <NA>
##      22      31      27      28      25      26      31      10
##
## Cross-tabulating the recoding:
##
##          gr
## gr          A  B CD Unknown
##   A        22  0  0       0
##   A2      31  0  0       0
##   B        0 27  0       0
##   C        0  0 28       0
##   D        0  0 25       0
##   d1       0  0 26       0
##   unknown  0  0  0      31
##   <NA>     0  0  0      10
```

Next, we add some more variables (but hide the output)

```
dm('aalder', 'Age')
dm('nar', 'When', comment = "wtf?")
dm('foo', 'event', db = 'COMP',
   recode = list('No' = '0', 'Yes' = 1),
   label = "An event at some time")
dm('bar', 'time', db = 'COMP', transf = log)
dm('koon', 'Gender',
   recode = list('Male' = 'M', 'Female' = 'K'))
```

When we are done, we create the CDB with

```
CDB <- dm_create(set = BL$id,
                 id.name = c('BL' = 'id', 'COMP' = 'ID'))

## Fixing variable no.1: gr
## Fixing variable no.2: Age
## Fixing variable no.3: When
## Fixing variable no.4: event
## Fixing variable no.5: time
## Fixing variable no.6: Gender

db_info(CDB) ## look at what we've created

##   source variable                 label   class
## 1    CDB       id                        factor
## 2    CDB       gr                 Group  factor
## 3    CDB      Age      Age at some time numeric
## 4    CDB     When                          Date
## 5    CDB    event An event at some time  factor
## 6    CDB     time           Time passed numeric
## 7    CDB   Gender               The Sex  factor
```

We can view, or get the information

```
## myDoc <- dm_doc()
dm_doc() ## only prints partial information in the doc
```

```
##      name    var   db transf                  label comment
## 1      gr     gr   BL                         Group
## 2     Age aalder   BL              Age at some time
## 3    When    nar   BL                                  wtf?
## 4   event    foo COMP         An event at some time
## 5    time    bar COMP    log            Time passed
## 6  Gender   koon   BL                       The Sex
```

You can store the information from the 'print' of the `dm_doc()` and the recodings with, respectively,

```
pdoc <- print(dm_doc(), print = FALSE)
rtables <- lapply(dm_doc(), FUN = function(x) x$recode_table)
```

and manipulate to output format of your choice.
If we are using LaTeX, we can get the code for this with

```
## dm_doc2latex(doc = myDoc)
dm_doc2latex(caption = "Variables and their origin.")
```

Table 1: Variables and their origin.

name	var	db	label	comment
`gr`	`gr`	**BL**	Group	
`Age`	`aalder`	**BL**	Age at some time	
`When`	`nar`	**BL**		*wtf?*
`event`	`foo`	**COMP**	An event at some time	
`time`	`bar`	**COMP**	Time passed	
`Gender`	`koon`	**BL**	The Sex	

and all recode-information with

```
## dm_recode2latex(doc = myDoc)
dm_recode2latex()
```

Table 2: Recoding of data base entry `gr` into `gr`.

old ↓ new →	A	B	CD	Unknown
A	22	0	0	0
A2	31	0	0	0
B	0	27	0	0
C	0	0	28	0
D	0	0	25	0
d1	0	0	26	0
unknown	0	0	0	31
NA	0	0	0	10

Table 3: Recoding of data base entry `foo` into `event`.

old ↓ new →	No	Yes
0	148	0
1	0	32

Table 4: Recoding of data base entry `koon` into `Gender`.

old ↓ new →	Male	Female
K	0	83
M	117	0

3 dmf and related functions

3.1 The problem we want to solve

A clinical data base is sometimes filtered to get an *analytical data base*, the documentation of which is, again, somewhat tedious. As with `dm`, we essentially want to do make the documentation parallell with coding so it is easy to update if the coding is updated.

3.2 Toy example continuation

Now, `dmf` plays the role of `dm`.

```
dmf(f = CDB$gr != 'Unknown', name = 'crit_knowngr',
    comment = "group must be known")

## includes 79.5 perc. (159/200 rows)

dmf(f = CDB$Age >= 20 & CDB$Age <= 80, name = "crit_age",
    comment = "ages between 20 and 80")

## includes 99 perc. (198/200 rows)

dmf(f = CDB$When >= as.Date("2002-01-01") &
        CDB$When <= as.Date("2009-12-31"),
    name = "crit_date",
    comment = "study period 2002-2009")

## includes 79 perc. (158/200 rows)
```

Ok, lets look at the documentation. As with `print.dm_doc` the print method obscures the real structure of the object somewhat.

```
dm_filter()

##        criteria incl excl seq.incl seq.excl
## 1    Population  200    0      200        0
## 2 crit_knowngr  159   41      159       41
## 3      crit_age  198    2      157        2
## 4     crit_date  158   42      119       38
```

We see how many rows each criteria includes/excludes and the inclusion/exclusion of these criteria when applied sequentially. We can change the order of the sequence, by

```
print(dm_filter(), seq = c(2,3,1))
```

```
##       criteria incl excl seq.incl seq.excl
## 1   Population  200    0      200        0
## 2     crit_age  198    2      198        2
## 3    crit_date  158   42      156       42
## 4 crit_knowngr  159   41      119       37
```

3.2.1 Experimental functions

Filter description via a list

```
dm_filter2latexlist()
```

- `crit_knowngr` 'group must be known' includes 79.5 perc. (159 rows)
- `crit_age` 'ages between 20 and 80' includes 99 perc. (198 rows)
- `crit_date` 'study period 2002-2009' includes 79 perc. (158 rows)

119 of 200 are included.

Filter cluster description

```
dm_filter2dist(plot = TRUE)
```

```
##           crit_knowngr  crit_age
## crit_age     0.5308642
## crit_date    0.9259259 0.5432099
```

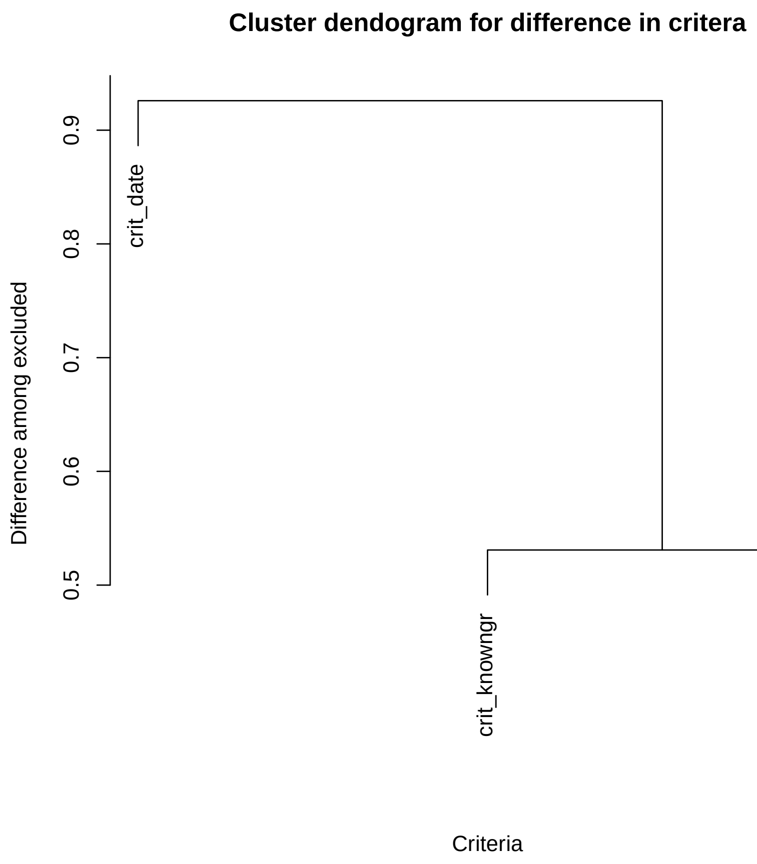

Figure 1: Test of cluster description

4 `grepict`

(**`grep`** by **i**ndividual, (possibly) **c**onstrained by **t**ime.)

4.1 The problem we want to solve

Suppose you have a set of units in a data set, and another data set where each unit may occur none or serveral times, each row associated with a date and one or more variables that contains some kind of information you want to search. For each unit we want to find matches in this information, possibly within some specified time range.

The function was specifically written to deal with this situation: the units are selected to be part of some study cohort, possible with different start and end times. Another data set exists which contains the medical records of (some possibly larger) population. At least two tasks are commonly associated with creating an clinical data base:

- Find the medical history of each individual, i.e. look for codes pertaining to different diagnosis that appear before the individual is entered into the cohort.
- Find diagnosis that appear after the individual is entered into the study (and possibly before some end date).

4.2 An example

4.2.1 Generate data

Our cohort will consists of some individuals entering a study during the year 2010, with no longer than one year follow-up.

- Anna enters 2010-01-01, with no medical history or outcomes,
- Barry enter 2010-02-01 (due to registered 'foo X' at this time), with a previous 'foo', but no outcome,
- Christina enters 2010-03-01, with no medical history but a later 'bar',
- David enters 2010-04-01, with a medical history of both 'bar' and 'foo', as well as a later 'quuz',
- Esteban enters 2010-05-01 (due to registered 'foo Y' at this time), with no medical history and an outcome 'bar' *after* the end of follow-up.

Moreover, the medical records are to be found in two different variables.

```
POP <- data.frame(
    id = c('Anna', 'Barry', 'Christina',
           'David', 'Esteban'),
    enter = as.Date(c('2010-01-01', '2010-02-01', '2010-03-01',
                      '2010-04-01', '2010-05-01'))
)
RECORDS <- data.frame(
    identity = c('Barry', 'Barry',
                 'Christina',
                 'David', 'David', 'David',
                 'Esteban', 'Esteban',
                 'Other', 'Other'),
    what1 = c('headache', 'foo X',
              'bar type I',
              'nausea', 'percutaneous foo', 'quuz',
              '', 'enui',
              'other foo', 'other bar'),
    what2 = c('mild foo', NA,
              'bar type II',
              'severe bar', 'subcutaneous foo', NA,
              'bar-ish', 'foo Y',
              'yet other foo', 'yet other bar'),
    what.date = as.Date(c('2010-01-07', '2010-02-01',
                          '2010-07-23',
                          '1998-06-27', '1996-10-12', '2011-01-18',
                          '2011-05-03', '2010-05-01',
                          '1999-12-01', '2010-06-01'))
)
options('knitr.kable.NA' = '')
```

The data is tabulated below

```
POP

##          id      enter
## 1      Anna 2010-01-01
## 2     Barry 2010-02-01
## 3 Christina 2010-03-01
## 4     David 2010-04-01
## 5   Esteban 2010-05-01

RECORDS

##     identity            what1            what2  what.date
## 1      Barry         headache         mild foo 2010-01-07
## 2      Barry            foo X             <NA> 2010-02-01
## 3  Christina       bar type I      bar type II 2010-07-23
## 4      David           nausea       severe bar 1998-06-27
## 5      David percutaneous foo subcutaneous foo 1996-10-12
## 6      David             quuz             <NA> 2011-01-18
## 7    Esteban                          bar-ish 2011-05-03
## 8    Esteban             enui            foo Y 2010-05-01
## 9      Other        other foo    yet other foo 1999-12-01
## 10     Other        other bar    yet other bar 2010-06-01
```

4.2.2 Medical history

Now we'll find the medical history of this cohort. We will need to point to the relevant variables in the different data sets, in `RECORDS` we need to point to `identity`, `date` and `what.date`. In `POP` we need to point to `id` and specify the search interval 'begin' and 'end'. In this case, we search as far back as we can, which will happen if we set 'begin' to `NULL`. We'll search all the way until the beginning of the study (which is coded in `enter` in the data frame). Actually, we will search strictly before `enter` (so as to not confuse the reason for entry into the study with medical history), by specifying `include = c(TRUE, FALSE)` which indicates that the lower bound is inclusive, but he upper bound is not.

There are options for the output format, but typicall we want a *stacked* and *long* format (which will be default).

```
searchString <- c('Foo' = 'foo', 'Bar' = 'bar', 'Quuz' = 'quuz')

tm <- grepict(
    pattern = searchString, ## what to search for
    x = c('what1', 'what2'), ## search variables in 'data'
    data = RECORDS, ## data set to search in
    id = 'identity', ## name of id variable in 'data'
    date = 'what.date', ## name of date variable in 'data'
    units = POP, ## data set, or vector, containing individuals
    units.id = 'id', ## name of id variable in 'units'
    begin = NULL, ## earliest date to search from
    end = 'enter', ## name of lates date to search
    include = c(TRUE, FALSE), ## include lower bound but not upper
    ## long = TRUE, ## long output format is default
    ## stack = TRUE, ## stacked results are default
    verbose = FALSE ## give calculation progression info?
)
```

`grepict` will return a data frame with many variables and, with this configuration, at least one row per individual and search string, and possibly as many as one per search string times variable searched in. Output (names are fixed):

- `id` the identifier

- `begin` first date searched from (inclusive)
- `end` last date searched untill (inclusive)
- `date` the date of the match
- `event` indicator for match
- `time` days between `begin` and `date`
- `match` that which matched
- `match.in` name of variable of match
- `pattern` pattern searched for
- `alias` name of pattern searched for
- `first.id` indicator for the first match for each individual and pattern
- `first.id_date` indicator for the first match for each individual, date and pattern

A few of these are tabulated below.

```
tm[, c('id', 'event', 'alias', 'match', 'match.in', 'first.id')]

##            id event alias            match match.in first.id
## 1       Barry     1   Foo         mild foo    what2        1
## 2       David     1   Foo percutaneous foo    what1        1
## 3       David     1   Foo subcutaneous foo    what2        0
## 4   Christina     0   Foo             <NA>     <NA>        1
## 5     Esteban     0   Foo             <NA>     <NA>        1
## 6        Anna     0   Foo             <NA>     <NA>        1
## 7       David     1   Bar       severe bar    what2        1
## 8       Barry     0   Bar             <NA>     <NA>        1
## 9   Christina     0   Bar             <NA>     <NA>        1
## 10    Esteban     0   Bar             <NA>     <NA>        1
## 11       Anna     0   Bar             <NA>     <NA>        1
## 12      Barry     0  Quuz             <NA>     <NA>        1
## 13 Christina     0  Quuz             <NA>     <NA>        1
## 14      David     0  Quuz             <NA>     <NA>        1
## 15    Esteban     0  Quuz             <NA>     <NA>        1
## 16       Anna     0  Quuz             <NA>     <NA>        1
```

For the history, we typically only care whether at least one instance of each search term is found. Also, we might want to transform this to a wide format.

```
tmp <- subset(tm, first.id == 1, select = c('id', 'event', 'alias'))
(medhist <- reshape(tmp, idvar = 'id',
                    timevar = c('alias'), direction = 'wide'))

##            id event.Foo event.Bar event.Quuz
## 1       Barry         1         0          0
## 2       David         1         1          0
## 4   Christina         0         0          0
## 5     Esteban         0         0          0
## 6        Anna         0         0          0

names(medhist) <- gsub("event", "prior", names(medhist),
                       fixed = TRUE)
```

Now, we have a data frame containing the relevant medical history

```
medhist

##            id prior.Foo prior.Bar prior.Quuz
## 1       Barry         1         0          0
## 2       David         1         1          0
## 4   Christina         0         0          0
## 5     Esteban         0         0          0
## 6        Anna         0         0          0
```

Using ustacked, wide output
Unstacked output gives essentially gives the same information as a stacked output, but with details for the first match and summary on all others (thus a one row per individual *and* search term). The wide output, which can only be used for non-stacked results, turns the non-stacked data into a wide format and gives each output variable name (which is not identical for each search term) a suffix (the alias). For certain applications, this is a shortcut.

```
tmwu <- grepict(
    pattern = searchString, ## what to search for
    x = c('what1', 'what2'), ## search variables in 'data'
    data = RECORDS, ## data set to search in
    id = 'identity', ## name of id variable in 'data'
    date = 'what.date', ## name of date variable in 'data'
    units = POP, ## data set, or vector, containing individuals
    units.id = 'id', ## name of id variable in 'units'
    begin = NULL, ## earliest date to search from
    end = 'enter', ## name of lates date to search
    include = c(TRUE, FALSE), ## include lower bound but not upper
    long = FALSE, ## use wide output
    stack = FALSE, ## do not stack output
    verbose = FALSE ## give calculation progression info?
)
## tmwu contains 33 variables, only some of which are of interest
tmwu[, names(tmwu)[grepl('(id|event)', names(tmwu))]]

##          id event.Foo events.Foo event.Bar events.Bar event.Quuz
## 1      Anna         0          0         0          0          0
## 2     Barry         1          1         0          0          0
## 3 Christina         0          0         0          0          0
## 4     David         1          2         1          1          0
## 5   Esteban         0          0         0          0          0
##   events.Quuz
## 1           0
## 2           0
## 3           0
## 4           0
## 5           0
```

4.2.3 Outcomes

Next, we'll look at outcomes. Since the end of study is variable, we'll have to create this variable, lets call it `endofstudy`.

```
POP$endofstudy <- POP$enter + 365
tm2 <- grepict(pattern = searchString, x = c('what1', 'what2'),
               data = RECORDS, id = 'identity',
               date = 'what.date', units = POP, units.id = 'id',
               begin = 'enter', ## earliest date to search from
               end = 'endofstudy', ## name of latest date
               include = c(FALSE, TRUE), ## include upper but not lower bound
               verbose = FALSE)
```

For the outcomes, we probably care about more things, especially time-to-event. The event and time variables now serve as right-censored data for each outcome.

```
tm2[, c('id', 'event', 'time', 'alias', 'match', 'match.in')]

##          id event time alias      match match.in
## 1     Barry     0  365   Foo       <NA>     <NA>
## 2 Christina     0  365   Foo       <NA>     <NA>
## 3     David     0  365   Foo       <NA>     <NA>
## 4   Esteban     0  365   Foo       <NA>     <NA>
## 5      Anna     0  365   Foo       <NA>     <NA>
## 6 Christina     1  144   Bar bar type I    what1
## 7  Christina     1  144    Bar  bar type II    what2
## 8      Barry     0  365    Bar          <NA>     <NA>
## 9      David     0  365    Bar          <NA>     <NA>
## 10   Esteban     0  365    Bar          <NA>     <NA>
## 11      Anna     0  365    Bar          <NA>     <NA>
## 12     David     1  292   Quuz          quuz    what1
## 13     Barry     0  365   Quuz          <NA>     <NA>
## 14 Christina     0  365   Quuz          <NA>     <NA>
## 15   Esteban     0  365   Quuz          <NA>     <NA>
## 16      Anna     0  365   Quuz          <NA>     <NA>
```

We'll assume that we only care about the first instance of each outcome.

```
tmp2 <- subset(tm2, first.id == 1,
               select = c('id', 'event', 'time', 'alias'))
(outcomes <- reshape(tmp2, idvar = 'id', timevar = c('alias'),
                     direction = 'wide'))

##          id event.Foo time.Foo event.Bar time.Bar event.Quuz time.Quuz
## 1     Barry         0      365         0      365          0       365
## 2 Christina         0      365         1      144          0       365
## 3     David         0      365         0      365          1       292
## 4   Esteban         0      365         0      365          0       365
## 5      Anna         0      365         0      365          0       365

names(outcomes) <- gsub("event", "ev", names(outcomes), fixed = TRUE)
names(outcomes) <- gsub("time", "t", names(outcomes), fixed = TRUE)
```

Now, we have a data frame containing the relevant outcomes.

```
outcomes

##          id ev.Foo t.Foo ev.Bar t.Bar ev.Quuz t.Quuz
## 1     Barry      0   365      0   365       0    365
## 2 Christina      0   365      1   144       0    365
## 3     David      0   365      0   365       1    292
## 4   Esteban      0   365      0   365       0    365
## 5      Anna      0   365      0   365       0    365
```

Other ways to get wide output As shown before, we can get wide output directly

```
POP$endofstudy <- POP$enter + 365
tm2wu <- grepict(pattern = searchString, x = c('what1', 'what2'),
               data = RECORDS, id = 'identity',
               date = 'what.date', units = POP, units.id = 'id',
               begin = 'enter', ## earliest date to search from
               end = 'endofstudy', ## name of latest date
               include = c(FALSE, TRUE), ## include upper but not lower bound
               long = FALSE, ## use wide output
               stack = FALSE, ## do not stack
               verbose = FALSE)
tm2wu[, names(tm2wu)[grepl('(id|event[^s]|time)', names(tmwu))]]

##            id event.Foo time.Foo event.Bar time.Bar event.Quuz time.Quuz
## 1        Anna         0      365         0      365          0       365
## 2       Barry         0      365         0      365          0       365
## 3 Christina         0      365         1      144          0       365
## 4       David         0      365         0      365          1       292
## 5     Esteban         0      365         0      365          0       365
```

4.3 Some details on wide and unstacked output

4.3.1 Wide and stacked

With a wide, stacked output, we get one row per individual and search. We get some information on the first match - all information from the long stacked format, except `first.id` and `first.id_date` - and some summary information on all matches:

- `events` which counts the matches,
- `matches` which concatenates the matches, and
- `matches.info` which stores a concatenation of *match*:*math.in*:*date* for all matches.

```
tm3 <- grepict(pattern = searchString, x = c('what1', 'what2'),
                  data = RECORDS, id = 'identity', date = 'what.date',
                  units = POP, units.id = 'id', begin = 'enter',
                  end = 'endofstudy',
                  long = FALSE, ## wide output format
                  stack = TRUE, ## stack
                  verbose = FALSE
)
str(tm3)

## 'data.frame': 15 obs. of  13 variables:
##  $ id          : Factor w/ 5 levels "Anna","Barry",..: 1 2 3 4 5 1 2 3 4 5 ...
##  $ begin       : Date, format: "2010-01-01" "2010-02-01" ...
##  $ end         : Date, format: "2011-01-01" "2011-02-01" ...
##  $ date        : Date, format: "2011-01-01" "2010-02-01" ...
##  $ event       : num  0 1 0 0 1 0 0 1 0 0 ...
##  $ time        : num  365 0 365 365 0 365 365 144 365 365 ...
##  $ match       : Factor w/ 18 levels "","bar type I",..: NA 4 NA NA 13 NA NA 2 NA NA ...
##  $ match.in    : Factor w/ 2 levels "what1","what2": NA 1 NA NA 2 NA NA 1 NA NA ...
##  $ pattern     : chr  "foo" "foo" "foo" "foo" ...
##  $ alias       : chr  "Foo" "Foo" "Foo" "Foo" ...
##  $ events      : num  0 1 0 0 1 0 0 2 0 0 ...
##  $ matches     : chr  NA "foo X" NA NA ...
##  $ matches.info: chr  NA "foo X:what1:2010-02-01" NA NA ...
```

Also, selected info tabulated below.

```
val <- c('id', 'alias', 'event', 'time', 'events', 'matches.info')
tm3[, val]

##             id alias event time events
## 1       Anna   Foo     0  365      0
## 2      Barry   Foo     1    0      1
## 3  Christina   Foo     0  365      0
## 4      David   Foo     0  365      0
## 5    Esteban   Foo     1    0      1
## 6       Anna   Bar     0  365      0
## 7      Barry   Bar     0  365      0
## 8  Christina   Bar     1  144      2
## 9      David   Bar     0  365      0
## 10   Esteban   Bar     0  365      0
## 11      Anna  Quuz     0  365      0
## 12     Barry  Quuz     0  365      0
## 13 Christina  Quuz     0  365      0
## 14     David  Quuz     1  292      1
## 15   Esteban  Quuz     0  365      0
##                                                          matches.info
## 1                                                                <NA>
## 2                                          foo X:what1:2010-02-01
## 3                                                                <NA>
## 4                                                                <NA>
## 5                                          foo Y:what2:2010-05-01
## 6                                                                <NA>
## 7                                                                <NA>
## 8  bar type I:what1:2010-07-23 bar type II:what2:2010-07-23
## 9                                                                <NA>
## 10                                                               <NA>
## 11                                                               <NA>
## 12                                                               <NA>
## 13                                                               <NA>
## 14                                          quuz:what1:2011-01-18
## 15                                                               <NA>
```

4.3.2 Wide and unstacked

With a wide and unstacked output, we get all variables (from the wide stacked output) *for each search term* - except `id`, `begin` and `end` which are the same for all rows - with the name of the search term as suffix (or a naming scheme if no names are supplied).

Below you can see the structure

```
tm4 <- grepict(pattern = searchString, x = c('what1', 'what2'),
               data = RECORDS, id = 'identity', date = 'what.date',
               units = POP, units.id = 'id', begin = 'enter',
               end = 'endofstudy',
               long = FALSE, ## wide output format
               stack = FALSE, ## don't stack
               verbose = FALSE
)
str(tm4)

## 'data.frame': 5 obs. of  33 variables:
##  $ id               : Factor w/ 5 levels "Anna","Barry",..: 1 2 3 4 5
##  $ begin            : Date, format: "2010-01-01" "2010-02-01" ...
##  $ end              : Date, format: "2011-01-01" "2011-02-01" ...
##  $ date.Foo         : Date, format: "2011-01-01" "2010-02-01" ...
##  $ event.Foo        : num  0 1 0 0 1
##  $ time.Foo         : num  365 0 365 365 0
##  $ match.Foo        : Factor w/ 18 levels "","bar type I",..: NA 4 NA NA 13
##  $ match.in.Foo     : Factor w/ 2 levels "what1","what2": NA 1 NA NA 2
##  $ pattern.Foo      : chr  "foo" "foo" "foo" "foo" ...
##  $ alias.Foo        : chr  "Foo" "Foo" "Foo" "Foo" ...
##  $ events.Foo       : num  0 1 0 0 1
##  $ matches.Foo      : chr  NA "foo X" NA NA ...
##  $ matches.info.Foo : chr  NA "foo X:what1:2010-02-01" NA NA ...
##  $ date.Bar         : Date, format: "2011-01-01" "2011-02-01" ...
##  $ event.Bar        : num  0 0 1 0 0
##  $ time.Bar         : num  365 365 144 365 365
##  $ match.Bar        : Factor w/ 18 levels "","bar type I",..: NA NA 2 NA NA
##  $ match.in.Bar     : Factor w/ 2 levels "what1","what2": NA NA 1 NA NA
##  $ pattern.Bar      : chr  "bar" "bar" "bar" "bar" ...
##  $ alias.Bar        : chr  "Bar" "Bar" "Bar" "Bar" ...
##  $ events.Bar       : num  0 0 2 0 0
##  $ matches.Bar      : chr  NA NA "bar type I bar type II" NA ...
##  $ matches.info.Bar : chr  NA NA "bar type I:what1:2010-07-23 bar type II:what2:2010-07-
##  $ date.Quuz        : Date, format: "2011-01-01" "2011-02-01" ...
##  $ event.Quuz       : num  0 0 0 1 0
##  $ time.Quuz        : num  365 365 365 292 365
##  $ match.Quuz       : Factor w/ 10 levels "","bar type I",..: NA NA NA 10 NA
##  $ match.in.Quuz    : Factor w/ 2 levels "what1","what2": NA NA NA 1 NA
##  $ pattern.Quuz     : chr  "quuz" "quuz" "quuz" "quuz" ...
##  $ alias.Quuz       : chr  "Quuz" "Quuz" "Quuz" "Quuz" ...
##  $ events.Quuz      : num  0 0 0 1 0
##  $ matches.Quuz     : chr  NA NA NA "quuz" ...
##  $ matches.info.Quuz: chr  NA NA NA "quuz:what1:2011-01-18" ...
```

Also, selected info on the Bar- and Quuz outcome tabulated.

```
val <- c('id', names(tm4)[grepl("event|time", names(tm4))])
tm4[, val[!grepl("Foo", val)]]
```

```
##          id event.Bar time.Bar events.Bar event.Quuz time.Quuz events.Quuz
## 1      Anna         0      365          0          0       365           0
## 2     Barry         0      365          0          0       365           0
## 3 Christina         1      144          2          0       365           0
## 4     David         0      365          0          1       292           1
## 5   Esteban         0      365          0          0       365           0
```

5 Other functions

5.1 `cdate`

`cdate` is a function to handle dates of the form "20010700" or "20010000", which can appear as dates of death when the precise date is unknown. If nothing else is known this function will replace an unknown

- day of the month with the midpoint of that month, and
- month (and day) with the midpoint of that year.

Sometimes, there is another date for an individual when the individual was known to be alive. In the applications of most interest to the author it is an admission, or discharge, date to a hospital, and thus we believe that the individual did *not* die at the hospital (else the date of death would be known), therefore the replacement date will be the midpoint of whatever remains of the unknown period, e.g if we encounter date of death as "20010100" with a known hospital discharge at 2001-01-21, we will interpret it as 2001-01-26.

Examples:

```
cdate(x = c("2001/01/01", "2001/01/00", "2001/00/00"), sep = "/")

## some x not interpretable as dates (at most 100 printed):

## [1] "2001/01/00" "2001/00/00"

##
## we'll try to fix them

## fixed!
## [1] "2001-01-01" "2001-01-16" "2001-07-02"

cdate(x = c("20010101", "20010100", "20010000"),
      low.bound = as.Date(c("1999-01-01", "2001-01-21",
                            "2001-06-20")))

## some x not interpretable as dates (at most 100 printed):

##   not_ok_dates  low.bound
## 1     20010100 2001-01-21
## 2     20010000 2001-06-20

##
## we'll try to fix them

## fixed!
## [1] "2001-01-01" "2001-01-26" "2001-09-25"
```