Introduction to R.HDSS

Alexander Ploner

2014-11-12

Background

Sankoh & Byass (2012) have described a standardized data format for storing and sharing data collected in health and demographic surveillance system (HDSS) sites collaborating in the International Network for the Demographic Evaluation of Populations and their Health (INDEPTH) Network. THe package `R.HDSS` offers tools for working with data in this format using the R statistical software language: currently, data files can be read into R, tested for their validity and checked for quality and consistence.

Reading in data

HDSS data is distributed in form of comma-separated text files (CSV). Below, we read in an artificially generated example data file that is included in the package:

```
require(R.HDSS)
file = system.file("extdata/testHDSS.csv", package="R.HDSS")
rawdata = readRawHDSS(file)
```

```
## Reading C:/Program Files/R/R-3.1.1/library/R.HDSS/extdata/testHDSS.csv...
```

This is the structure of the raw data read into R:

```
str(rawdata)
```

```
## 'data.frame':    24339 obs. of  14 variables:
##  $ RecNr          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ CountryId      : Factor w/ 1 level "testCountry": 1 1 1 1 1 1 1 1 1 1 ...
##  $ CentreId       : Factor w/ 1 level "testCentre": 1 1 1 1 1 1 1 1 1 1 ...
##  $ IndividualId   : int  1 1 2 2 3 3 4 4 4 5 ...
##  $ Sex            : Factor w/ 2 levels "f","m": 1 1 2 2 1 1 1 1 1 1 ...
##  $ DoB            : Factor w/ 6618 levels "1914-09-06","1914-09-30",..: 370 370 5528 5528 1865 1865 3
##  $ EventCount     : int  2 2 2 2 2 2 3 3 3 2 ...
##  $ EventNr        : int  1 2 1 2 1 2 1 2 3 1 ...
##  $ EventCode      : Factor w/ 10 levels "BTH","DLV","DTH",..: 5 8 5 8 7 8 5 2 8 5 ...
##  $ EventDate      : Factor w/ 1276 levels "1987-10-01","2002-11-27",..: 321 1224 205 1224 1132 1172 (
##  $ ObservationDate: Factor w/ 4175 levels "1792-10-11","1792-10-22",..: 3069 3730 2028 721 3052 3757
##  $ LocationId     : int  916 916 10766 10766 8945 1994 4366 4366 4366 3311 ...
##  $ MotherId       : int  NA NA NA NA NA NA NA 4 NA NA ...
##  $ DeliveryId     : int  NA NA NA NA NA NA NA 228 NA NA ...
```

Checking for validity

In the first step, we test whether the data we have just read in fulfills the formal criteria for an HDSS data file, as outlined in S & B (2012). We check whether the required columns are present comply with some minimal requirements:

```
coreVariableTests(rawdata)
```

```
##                      Test        Variable  Pass
## 1         Variable exists           RecNr  TRUE
## 2       No missing values           RecNr  TRUE
## 3    No duplicate values           RecNr  TRUE
## 4  Numeric and sequential           RecNr  TRUE
## 5         Variable exists       CountryId  TRUE
## 6       No missing values       CountryId  TRUE
## 7          Constant label       CountryId  TRUE
## 8         Variable exists        CentreId  TRUE
## 9       No missing values        CentreId  TRUE
## 10         Constant label        CentreId  TRUE
## 11        Variable exists    IndividualId  TRUE
## 12      No missing values    IndividualId  TRUE
## 13        Variable exists             Sex  TRUE
## 14      No missing values             Sex  TRUE
## 15        Variable exists      LocationId  TRUE
## 16      No missing values      LocationId  TRUE
## 17        Variable exists             DoB  TRUE
## 18      No missing values             DoB  TRUE
## 19        Variable exists       EventCode  TRUE
## 20      No missing values       EventCode  TRUE
## 21        Variable exists       EventDate  TRUE
## 22      No missing values       EventDate  TRUE
## 23        Variable exists ObservationDate  TRUE
## 24      No missing values ObservationDate FALSE
## 25        Variable exists      EventCount  TRUE
## 26      No missing values      EventCount  TRUE
## 27        Variable exists         EventNr  TRUE
## 28      No missing values         EventNr  TRUE
```

In this case, we see that

- all the required variables are part of the data set,
- the unique record identifier `RecNr` is a numerical variable with sequential entries and no missing values,
- country- and site identifiers are non-missing and constant across the data set.

The only test that the data failed to pass was for the variable `ObservationDate`, where some missing values were found. We can conclude that the data we have just read in is a valid HDSS file, though possibly with some complications (as can be expected from real or realistic data).

Preprocessing the raw data

Up to this point, we have not modified the data at all. For our next step, however, we need to be able to make some assumptions about the content of the different variables. This requires converting (or attempting to convert) all date variables into a standard R date format (note that S & B do not specifiy an explicit date format in their paper, so date notation can vary considerably between data files, depending on country settings of the operating system, software used and preferences).

```
testdata = preprocHDSS(rawdata)
str(testdata)
```

```
## Classes 'HDSSdata' and 'data.frame': 24339 obs. of  14 variables:
##  $ RecNr          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ CountryId      : Factor w/ 1 level "testCountry": 1 1 1 1 1 1 1 1 1 1 ...
##  $ CentreId       : Factor w/ 1 level "testCentre": 1 1 1 1 1 1 1 1 1 1 ...
##  $ IndividualId   : int  1 1 2 2 3 3 4 4 4 5 ...
##  $ Sex            : Factor w/ 2 levels "m","f": 2 2 1 1 2 2 2 2 2 2 ...
##  $ DoB            : Date, format: "1946-06-24" "1946-06-24" "2001-12-02" ...
##  $ EventCount     : int  2 2 2 2 2 2 3 3 3 2 ...
##  $ EventNr        : int  1 2 1 2 1 2 1 2 3 1 ...
##  $ EventCode      : Factor w/ 10 levels "BTH","DLV","DTH",..: 5 8 5 8 7 8 5 2 8 5 ...
##  $ EventDate      : Date, format: "2005-08-17" "2008-02-07" "2005-04-23" ...
##  $ ObservationDate: Date, format: "2007-11-24" "2009-09-20" "2005-01-16" ...
##  $ LocationId     : int  916 916 10766 10766 8945 1994 4366 4366 4366 3311 ...
##  $ MotherId       : int  NA NA NA NA NA NA NA 4 NA NA ...
##  $ DeliveryId     : int  NA NA NA NA NA NA NA 228 NA NA ...
```

Note that the new `testdata` looks very similar to the old `rawdata`, but some important changes have happened under the hood: for the original data, the birth date `DoB` was a categorical (factor) variable, whereas for the preprocessed data, it is now a valid date format, with which we can do comparisons and arithmetic, and similar for the other date variables.

Quality testing

We can now check the content of the preprocessed data. Below, we see the result of applying a number of tests to each record in the data set; for each test, the number of `TRUE` (test passed), `FALSE` (test failed) and `NA` (test not applicable) results is listed:

```
coreRecordTests(testdata)
```

```
##                                                  Description  TRUE FALSE    NA
## 1                                Correct codes in 'CountryId'     0 24339     0
## 2                                 Correct codes in 'CentreId'     0 24339     0
## 3                                      Correct codes in 'Sex' 24339     0     0
## 4                                Correct codes in 'EventCode' 24339     0     0
## 5                           Value not missing in 'LocationId' 24339     0     0
## 6                                  Value not missing in 'DoB' 24339     0     0
## 7                                       Valid range for 'DoB' 24339     0     0
## 8                            Value not missing in 'EventDate' 24339     0     0
## 9                                 Valid range for 'EventDate' 24337     2     0
## 10                      Value not missing in 'ObservationDate' 22972  1367     0
## 11                           Valid range for 'ObservationDate' 15561  7411  1367
## 12                          'DoB' less or equal to 'EventDate' 24335     4     0
## 13 No OBE only: 'EventDate' less or equal to 'ObservationDate' 10902  2690 10747
## 14                           Delivery for female subjects only   521     0 23818
## 15                         Birth event linked to valid delivery   521     0 23818
```

Let's go through the tests:

1. Correct codes in `CountryId/CentreId`: these are checked against a (currently incomplete) list of INDEPTH participating sites, see `?INDEPTH_Centres`. Our synthetic test data lists country as `testCountry` and site as `testCentre` and therefore fails these tests.

2. Correct codes for `Sex`: should be `m/f`for male/female, the test data passes.

3. Correct codes for `EventCodes`: checks the reported event codes against the pre-defined ones in `INDEPTH_eventCodes`.

4. Values not missing in `LocationId` and `DoB`: all events in the test data are reported with a correct identifier and recognizable date, respectively.

5. Valid range for `DoB`: we keep an eye out for unexpectedly early or late birth dates that either indicate excpetional old age or birth after the end of the study period. By default, we look for births fate the last closing event (coded `OBE`) or indicating an age of greater 105 years during any time of the stuyd period, though this can be modified, see `? coreRecordTests`. For our test data, all records pass the default range tests.

6. Missing values and valid range for `EventDate`: all events should be dated and fall into the study period. By default, the stuyd period is between the earliest non-missing event date of all entry events (enumeration, birth and immigration, coded `ENU`, `BTH` and `IMG`) and the last non-missing closing event date (`OBE`, see above). For our test data, we see that two records fail this test.

7. Missing values and valid range for `ObservationDate`: these are checked against an extended stuyd period, i.e. observation dates that fall not too far behind the study end are still accepted as valid; by default, this grace period is half a year, but it can be varied by the user, see `?coreRecoredTests`. For the test data, we see fairly large number of missing values, as well as an even larger number of out-of-range values, even with the grace period, indicating that the quality of the observation dates is lower than for the other date variables.

8. As a minimal test for consistency, we check whether recorded events fall after the birth date of the subject; here, we find four exceptions.

9. As a similar consistency test, we check whether the observation date falls after the date of the observed event. We limit this check to non-closing (i.e. non-`OBE`) events, as these fail the check notoriously and by construction: when closing a cohort, as a rule the last observation date prior to closing is pulled forward, leading to observation before the fact. As we can seen in the test data, even when we exclude the closing events, we find a reasonably large number of observation dates preceding the event date.

10. Finally, we test whether all delivery events (coded `DLV`) are linked to female subjects, and whether each birth event (coded `BTH`) is linked to an existing delivery event (the latter is required, as only births to female subjects already participating in the study should be registered). For our test data, all births and deliveries pass these tests (note that for out simple test data, the number of births and deliveries match exactly, as we have no still-births and no multiple births).

Investigating quality issues

We have seen a number of quality issues with the pre-processed data in the previous section. We can actually take the output of the testing routine `coreRecordTests` to investigate more closely the records that failed some or all of the tests.

```
rt = coreRecordTests(testdata)
str(rt)
```

```
## List of 2
##  $ Index: logi [1:24339, 1:15] FALSE FALSE FALSE FALSE FALSE FALSE ...
```

```
##  $ Desc : chr [1:15] "Correct codes in 'CountryId'" "Correct codes in 'CentreId'" "Correct codes in
##  - attr(*, "class")= chr "recTest"
```

As shown above, the test result is a list consisting of two items: `Index`, a logical matrix with as many rows as records in the data and as many columns as tests performed (15 to be precise), and `Description`, a vector of strings describing the nature of the test. Now if we want to look at the two records that event dates out of range, as seen above, we note that this is the 9-th test in the list. Then we can use `dplyr` to extract the corresponding records:

```
require(dplyr)
filter(testdata, !rt$Index[,9]) %>% select(RecNr:EventDate)
```

```
##   RecNr   CountryId   CentreId IndividualId Sex        DoB EventCount EventNr EventCode  EventDate
## 1   211 testCountry testCentre           87   m 1980-12-01          4       3       ENT 2002-11-27
## 2  7460 testCountry testCentre         3076   f 1981-01-08          6       5       ENT 1987-10-01
```

We find that there are two `ENT`-events with very early dates: given that the earliest date for an entry vent in the rest of the data is in October 2004, and given that `ENT` indicates a in-migration event internal to the site are and should always be paired with with corresponding out-migration event `EXT`. we want to have a closer look at the records of these two individuals:

```
## Start of study
filter(testdata, EventCode %in% c("ENU", "IMG", "BTH")) %>% summarise(min(EventDate))
```

```
##   min(EventDate)
## 1     2004-10-03
```

```
## All records for the individuals
filter(testdata, IndividualId %in% c(87, 3076)) %>% select(IndividualId:ObservationDate)
```

```
##    IndividualId Sex        DoB EventCount EventNr EventCode  EventDate ObservationDate
## 1            87   m 1980-12-01          4       1       ENU 2004-11-10      2004-08-26
## 2            87   m 1980-12-01          4       2       EXT 2007-11-02      2008-12-25
## 3            87   m 1980-12-01          4       3       ENT 2002-11-27      2007-01-11
## 4            87   m 1980-12-01          4       4       OBE 2008-02-15      2005-10-18
## 5          3076   f 1981-01-08          6       1       IMG 2006-01-16      2005-11-18
## 6          3076   f 1981-01-08          6       2       EXT 2006-05-16            <NA>
## 7          3076   f 1981-01-08          6       3       ENT 2006-07-08      2006-10-03
## 8          3076   f 1981-01-08          6       4       EXT 2007-11-20      2010-05-18
## 9          3076   f 1981-01-08          6       5       ENT 1987-10-01      1987-12-15
## 10         3076   f 1981-01-08          6       6       OBE 2008-01-12      2009-10-10
```

It appears that the two `ENT`-events out of range were coded incorrectly, possibly at data entry or -processing. We now might decide to either

- ignore the issue and to continue with the data as-is,
- impute plausible event dates for the out-of-range events, and to continue with the modifed data,
- remove the two subjects from the data set and further analysis.

All of these are in principle valid approaches, depending on availability of background information, goals of the analysis and other circumstances. Here, we choose to eliminate subjects from the data, not because we feel that this is an especially suitable default behaviour, but because we want to demonstrate how we can manipulate an HDSS data set:

```
testdata2 = filter(testdata, !(IndividualId %in% c(87, 3076)) )
```

Note that the filtering here is very straightforward, because neither of the individuals had a delivery-event: in that case, we would need to decide how to deal with the birth events(s) that would have been (figuratively) orphaned by eliminating the mother from the data set.

Events, event patterns and transitions

With a few exceptions, the diagnostics above are based on dates and comparison of dates. However, just looking at the frequency and sequence of events (as indicated by the event codes) can be highly informative. We'll demonstrate with the slighlty revised test data below.

Let's start with a simple frequency distribution of events:

```
table(testdata2$EventCode)
```

```
##
##  BTH  DLV  DTH  ENT  ENU  EXT  IMG  OBE  OBL  OMG
##  521  521  145  686 7474  718 2127 9998  134 2005
```

We find

1. $n = 9998$ closing events (`OBE`), indicating that each individual in the data set has a closing event, as required by the HDSS specification;
2. slightly more immigration into the study (`IMG`, $n = 2127$) than emigration from the study (`OMG`, $n = 2005$);
3. a strong overhand of births (`BTH`, $n = 521$) over deaths (`DTH`, $n = 145$);
4. as noted before, perfect agreement between the number of births (`BTH`) and deliveries (`DLV`), which is an artefact due to the synthetic nature of the data; however, this could be informative regarding unexpcted numbers of multiple births and/or still births;
5. non-matching numbers of internal out- and in-migration (`EXT`, $n = 718$, and `ENT`, $n = 686$); by specification, these are two aspects of the same event, and should always appear matched, so apparently, the (fictitious) data collectors in this situation were not able to trace all cahnges of residence successfully.

Let's continue with looking at the frequencies of first/last events for each individual:

```
table(firstEvents(testdata2))
```

```
##
##  BTH  DLV  ENU  IMG
##  521    7 7472 1998
```

```
table(lastEvents(testdata2))
```

```
##
##  OBE
## 9998
```

As suggested above, each individual has the required `OBE` event at the end of their event sequence, dating the end of follow-up for the subject. In the same manner, the birth-, enumeration- and immigration events are valid entry events as per specification, though the seven deliveries (`DLV`) are not.

Finally, we can look at the transitions between event states as follows:

```
trans = getTransitions(testdata2)
```

This both counts the number of times any possible event code is followed by any other possible event code, and sums the total person time that is spent between the two events (in days). Let's look at the counts first: here, the rows indicate the first (or source-) event, and the columns the second (or sink-) event.

```
trans$counts
```

```
##     BTH ENU IMG  OMG EXT ENT DTH DLV  OBE OBL OBS
## BTH   0   1   3   42  22   0  20   0  432   1   0
## ENU   0   0   0 1311 445   0 109 377 5171  61   0
## IMG   0   0   0  511 134   0   8  76 1372  26   0
## OMG   0   0 123    0   0   0   1   1 1843  37   0
## EXT   0   0   0    0   0 686   0   0   28   4   0
## ENT   0   0   0   96  95   0   5  27  459   4   0
## DTH   0   0   0    0   0   0   0   0  145   0   0
## DLV   0   1   3   45  22   0   2  33  414   1   0
## OBE   0   0   0    0   0   0   0   0    0   0   0
## OBL   0   0   0    0   0   0   0   0  134   0   0
## OBS   0   0   0    0   0   0   0   0    0   0   0
```

(interpretation)

These are the aggregated number of times elapsing between states:

```
trans$deltaT
```

```
##     BTH ENU   IMG    OMG    EXT   ENT   DTH    DLV     OBE   OBL OBS
## BTH   0 264   623  12635   5554     0   737      0  221412   377   0
## ENU   0   0     0 597798 190010     0 48858 167084 5322860 28512   0
## IMG   0   0     0 165299  38212     0  2625  20319  629665 10029   0
## OMG   0   0 45437      0      0     0    25    237  931907 10999   0
## EXT   0   0     0      0      0 53515     0      0   16321   647   0
## ENT   0   0     0  22110  20015     0  1433   6726  170398   712   0
## DTH   0   0     0      0      0     0     0      0   81771     0   0
## DLV   0 264   623  14657   5554     0   208  18245  203274   377   0
## OBE   0   0     0      0      0     0     0      0       0     0   0
## OBL   0   0     0      0      0     0     0      0   73468     0   0
## OBS   0   0     0      0      0     0     0      0       0     0   0
```

(interpretation)

We can use this to calculate average time between events, e.g. between birth into the study and death:

```
with(trans, deltaT["BTH", "DTH"]/counts["BTH", "DTH"])
```

```
## [1] 36.85
```

indicating that average age at death for the children that were born into the cohort and died was ca. 37 days.