Revisiting hospital transfers

Background

The process of dealing with hospital transfers to allow for transfers has been dealth with in SAS over a number of years, with the latest version of this code available at https://github.com/timothydobbins/HospitalTransfers. This code is based on the principles that a record is a transfer if:

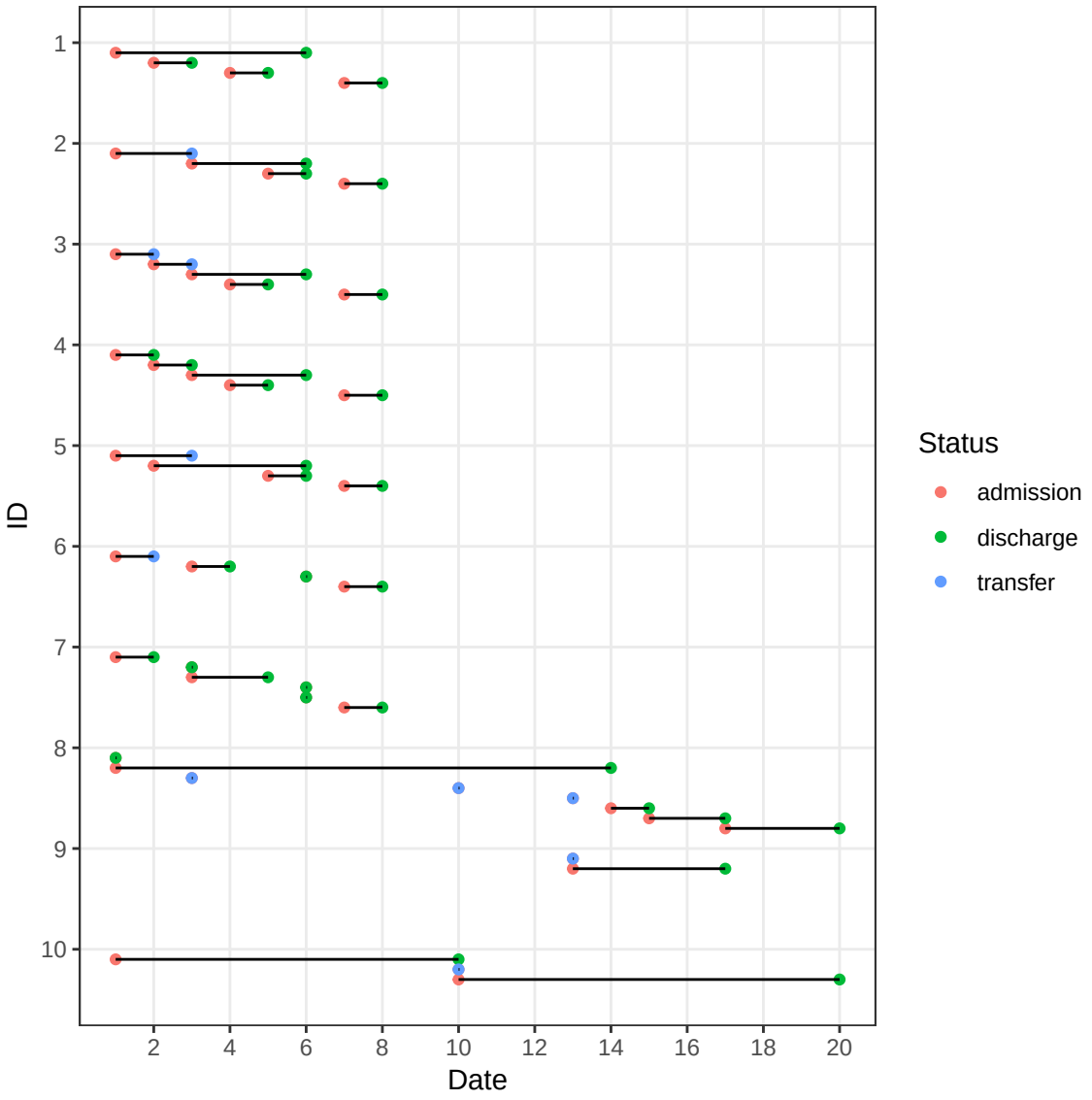
1. its start date is before the previous record's end date (i.e. a *nested* transfer); or
2. its start date is equal to the previous record's end date *and* the previous record's mode of separation indicated a (hospital or type-change) transfer.

The code has been adapted over many years to allow for a number of edge cases, by adding code to tweak the above two principles. However, this code (written and developed in SAS) makes use of a number of SAS-specific quirks. This limits the anlalyst to having access to SAS in order to process hospital data.

This document outlines the process of identifying and dealing with hospitalisation data in R, with a view to be exported to any other package.

Sample data

Consider the following, small set of sample data, with 10 individuals.



Some interesting observations:

- ID 1 has two nested transfers within the first episode;
- ID 3 has three consecutive transfers, with a nested transfer in the third episode;
- ID 4 has three consecutive (non-transferred) admissions, and a nested transfer in the third episode;
- ID 7 has two same-day episodes at day 6;
- ID 9 has a single-day episode resulting in a transfer (day 13);
- ID 10 has a separation, single-day transferred episode and admission all on the same day.

A proposed alternative

The crux of the following is derived from **Stata tip 114: Expand paired dates to pairs of dates**; The Stata Journal (2013), 13, Number 1, pp. 217–219 (https://www.stata-journal.com/sjpdf.html?articlenum=dm0068)

An alternative approach (not yet investigated) is the concept of Allen's Interval Algebra, discussed at stackoverflow (https://stackoverflow.com/questions/325933/determine-whether-two-date-ranges-overlap)

We can transpose our paired data comprising (ID, start_date, end_date) to (ID, data, inout) where: inout=1

for an admission and -1 for a separation. By calculating a cumulative sum for each ID, a person's hospital status can be inferred as being in hospital for sum>0 and out of hospital for sum=0.

For example, ID 6:

id	admdate	sepdate	transfer
6	1	2	1
6	3	4	0
6	6	6	0
6	7	8	0

Transposing data, creating inout and creating a cumulative sum gives:

ID	date	type	inout	cum.sum
6	1	adm	1	1
6	2	sep	-1	0
6	3	adm	1	1
6	4	sep	-1	0
6	6	adm	1	1
6	6	sep	-1	0
6	7	adm	1	1
6	8	sep	-1	0

Here each record with a pair dates has been converted into two dates. ID 6 was admitted on day 1, separated on day 2, admitted on day 3, separated on day 4 etc. ID 6 had four distinct periods of stay, with no transfers, and each time they left the hospital, their cum.sum was 0. Let's now consider ID 1, with nested transfers:

ID	admdate	sepdate	transfer
1	1	6	0
1	2	3	0
1	4	5	0
1	7	8	0

Transposing and creating inout:

ID	date	type	inout
1	1	adm	1
1	6	sep	-1
1	2	adm	1
1	3	sep	-1
1	4	adm	1
1	5	sep	-1
1	7	adm	1
1	8	sep	-1

Before creating the cumulative sum, we can **sort by date**:

ID	date	type	inout	cum.sum
1	1	adm	1	1
1	2	adm	1	2
1	3	sep	-1	1
1	4	adm	1	2
1	5	sep	-1	1
1	6	sep	-1	0
1	7	adm	1	1
1	8	sep	-1	0

We can see from this example that the inout variable indicates whether a participant is currently in or out of

hospital. Here ID 1 had only two periods of stay: they left hospital on day 6 and day 8. The nested transfers have been dealt with, and dates of separation out of hospital are indicated by a cum.sum of 0.

So far we have seen how this concept works for multiple admissions with no transfers (ID 6) and multiple admissions with nested transfers (ID 1). A transfer will often happen on the same day as a new admission (i.e. a hosptial or type-change transfer). ID 3 is a good example of this: they were transferred on day 2 and day 3, and had a nested transfer. Let's examine what happens using the same process as above.

id	admdate	sepdate	transfer
3	1	2	1
3	2	3	1
3	3	6	0
3	4	5	0
3	7	8	0

Transposed:

ID	date	type	transfer	inout	cum.sum
3	1	adm	1	1	1
3	2	sep	1	-1	0
3	2	adm	1	1	1
3	3	sep	1	-1	0
3	3	adm	0	1	1
3	4	adm	0	1	2
3	5	sep	0	-1	1
3	6	sep	0	-1	0
3	7	adm	0	1	1
3	8	sep	0	-1	0

We can see that the process outlined so far will not deal with serial transfers (day 2 and day 3 are incorrectly flagged as new admissions). We can use the *transfer* variable to increase the date of separation of a transfer by, say 0.1 days. This will ensure that, after sorting by date, a subsequent admission in a serial transfer occurs before the date of transfer. For example:

ID	date	type	transfer	inout	cum.sum
3	1.0	adm	1	1	1
3	2.0	adm	1	1	2
3	2.1	sep	1	-1	1
3	3.0	adm	0	1	2
3	3.1	sep	1	-1	1
3	4.0	adm	0	1	2
3	5.0	sep	0	-1	1
3	6.0	sep	0	-1	0
3	7.0	adm	0	1	1
3	8.0	sep	0	-1	0

This process then successfully indicates that ID 6 had two periods of stay: one from day 1 to 6 (comprising four episodes), and one from day 7 to 8.

Using this basic idea, we can construct transfers code in R to identify transfers, count periods of stay and populate a final date of separation for transferred records. After running this code, we get the following table for our example data:

id	stayseq	episode	admdate	sepdate	transfer	transseq	sepdate_fin	totlos
1	1	1	1	6	0	0	6	5
		2	2	3	0	1	6	4
		3	4	5	0	2	6	2
	2	4	7	8	0	0	8	1
2	1	1	1	3	1	0	6	5
		2	3	6	0	1	6	3
		3	5	6	0	2	6	1
	2	4	7	8	0	0	8	1
3	1	1	1	2	1	0	6	5
		2	2	3	1	1	6	4
		3	3	6	0	2	6	3
		4	4	5	0	3	6	2
	2	5	7	8	0	0	8	1
4	1	1	1	2	0	0	2	1
	2	2	2	3	0	0	3	1
	3	3	3	6	0	0	6	3
		4	4	5	0	1	6	2
	4	5	7	8	0	0	8	1
5	1	1	1	3	1	0	6	5
		2	2	6	0	1	6	4
		3	5	6	0	2	6	1
	2	4	7	8	0	0	8	1
6	1	1	1	2	1	0	2	1
	2	2	3	4	0	0	4	1
	3	3	6	6	0	0	6	0
	4	4	7	8	0	0	8	1
7	1	1	1	2	0	0	2	1
	2	2	3	3	0	0	3	0
	3	3	3	5	0	0	5	2
	4	4	6	6	0	0	6	0
	5	5	6	6	0	0	6	0
	6	6	7	8	0	0	8	1
8	1	1	1	1	0	0	1	0
	2	2	1	14	0	0	14	13
		3	3	3	1	1	14	11
		4	10	10	1	2	14	4
		5	13	13	1	3	14	1
	3	6	14	15	0	0	15	1
	4	7	15	17	0	0	17	2
	5	8	17	20	0	0	20	3
9	1	1	13	13	1	0	17	4
		2	13	17	0	1	17	4
10		1	1	10	0	0	10	9
	2	2	10	10	1	0	20	10
		3	10	20	0	1	20	10