Identifying episodes of care in hospital admissions data for measures of disease burden: A protocol for individual-level data analysis

February 3, 2025
https://github.com/jimb0w/HA

Jedidiah I Morton, Adam Livori, Lee Nedkoff, Dianna J Magliano, Derrick Lopez, Ingrid Stacey, Zanfina Ademi.

Correspondence to:

Jedidiah Morton
Jedidiah.Morton@monash.edu
Monash University, Melbourne, Australia
Baker Heart and Diabetes Institute, Melbourne, Australia

Adam Livori
adam.livori@gh.org.au
Grampians Health, Ballarat, Australia
Monash University, Melbourne, Australia

Contents

1 Introduction

After cleaning hospital data for the *nth* time on a new, completely different dataset from another country that required almost exactly the same data processing as Australian admissions data, we decided that it would be useful to have a standard protocol for processing hospital admissions data and defining episodes of care. This is supposed to be that protocol, although we welcome criticism and change – this won't be perfect and all the nuances may not be covered, but this should serve as a good starting point for people attempting to analyse hospital data for the first time.

This protocol will therefore outline most of the general principles for processing hospital admissions data to define an episode of care, such as:

- Defining your event
- Finding and removing "duplicate" admissions while keeping the information they contain
- Finding and removing "nested" admissions while keeping the information they contain
- Tagging and removing transfers

And will include a full worked example at the end. Some of this may be basic, but unless it's standardised, we (the research community) risk doing it incorrectly or different each time, or even just wasting time re-writing code every time (especially as a lot of this work is done by students new to this data).

Finally, because we want to show individual-level data, we are going to use completely synthetic datasets for the first part of this protocol, but with cases based on common aspects of the data we have come across. In the synthetic dataset, we will assume the following variables are present:

- Unique inidividual identifier (*id*)
- Admission date (*admdate*)
- Separation date (*sepdate*)
- Diagnosis codes (*diagx*)
- Admission source (*admmode*)
- Separation destination (*sepmode*)

Therefore, the first part of this protocol (2) will go through worked examples of how to process admissions data and define an episode of care. Then, in the second part of the protocol (3), we show a full example using hospital episode statistics from inpatient data (the HESIN dataset) derived via linkage to the UK Biobank study that we have used for research previously (e.g., Morton et al., Value in Health, 2024).

2 Defining an episode of care

2.1 Data harmonisation and checks

We have decided not to go through this in any detail, given that the specifics will depend on the dataset in question. However, three brief comments about data harmonisation and checks. First, check *every* variable you use and create. As in, look at it, tabulate or histogram it, make sure it makes sense, check how many missing values there are, and make sure it makes sense in the context of other variables (e.g., is separation date ever earlier than admission date?). Second, watch everything you do across multiple examples, it is labourious, but worth it – slow is smooth and smooth is fast (i.e., it's much quicker to get it right the first time and then move on than to have to keep coming back to code you wrote a while ago every time you notice a new mistake/problem further along in the analysis). Similarly, tabulate key variables as you go to make sure you haven't made unexpected errors, such as erroneously deleting or observations/data, recoded variables incorrectly, made typos and used the wrong function (e.g., *gen* instead of *egen*). Third, do not waste time perfecting code for 0.001% of the dataset. The processes and steps we outline below will work for most cases. We are also sure that these steps will miss things, but we are fairly confident those will be irrelevant – usually you will be working with samples that have at least 10,000 people in them, so having a handful of admissions coded incorrectly should not impact analyses and we do not recommend scouring through thousands of admissions to try and find them (especially if you are doing data checks, as recommended in our first comment, which will pick up any serious problems and you can then deal with them individually if they are likely to impact your analysis).

2.2 Defining admissions of interest

After preparaing the variables you intend to use, the first step in defining an episode of care is to define the event of interest. Defining an event or an admission you're interested in is primarily a clinical question and related to the study at hand. There are two common ways to define an event or admission, that we will now outline.

First, using only the primary diagnosis for an admission. In this example, let's assume you're looking at myocardial infarctions (ICD-10 code: I21), in which case, an event is probably only relevant when it's the primary diagnosis for an admission (as when it's a secondary diagnosis it probably relates to a prior myocardial infarction rather than an event). Usually, the first diagnosis code in a dataset is the primary reason for admission, so we will only use that variable to tag all admissions for myocardial infarction.

```
. list in 1/10, separator(0)
```

	diag1
1.	K833
2.	J749
3.	B895
4.	D209
5.	X408
6.	T700
7.	M308
8.	F421
9.	N609
10.	U688

```
. *Our dataset is just randomly generated ICD-10 codes
. gen MI = 1 if substr(diag,1,3)=="I21"
(99,960 missing values generated)
. sort MI
. list in 35/44, separator(0)
```

	diag1	MI
35.	I213	1
36.	I219	1
37.	I217	1
38.	I217	1
39.	I217	1
40.	I215	1
41.	O885	.
42.	I947	.
43.	W878	.
44.	B974	.

```
. *We successfully tag all admissions with a primary diagnosis for MI
```

The second way an event is often defined is when the position of the diagnosis (or procedure), is not relevant for the study. For this example, let's assume you're interested in diabetes status for an individual (ICD-10 codes: E10-E14), and that there are 10 diagnoses associated with each admission.

```
. list in 1/10, separator(0)
```

	diag1	diag2	diag3	diag4	diag5	diag6	diag7	diag8	diag9	diag10
1.	F609	O466	S516	V186	J121	G993	F631	L768	Y330	R266
2.	W790	O263	R353	F456	R119	F380	F081	K963	Z584	H879
3.	A433	S762	C249	O166	U852	N334	P040	R127	W641	V687
4.	H678	P884	I903	B003	H713	V166	L178	M872	H091	T575
5.	Y334	Z135	L002	K111	I956	U035	H706	W761	D399	U715
6.	N574	C216	M566	Q240	S958	W024	B033	A222	C830	S202
7.	J413	U880	B076	K921	E117	P650	V910	M413	D493	N804
8.	J280	H261	W341	E273	Z854	W896	F527	O826	B651	A676
9.	Z774	C835	P681	M020	U289	S419	Y873	R719	Y319	Q046
10.	R536	M761	K485	U037	Q858	D389	K805	Y912	A562	N392

```
. *Our dataset is again just randomly generated ICD-10 codes
. gen DM = .
(1,000 missing values generated)
. forval i = 1/10 {
  2. replace DM = 1 if inrange(diag`i´,"E10","E149")
  3. }
(0 real changes made)
(0 real changes made)
(3 real changes made)
(2 real changes made)
(2 real changes made)
(1 real change made)
(0 real changes made)
(1 real change made)
(1 real change made)
(3 real changes made)
. sort DM
. list in 9/18, separator(0)
```

	diag1	diag2	diag3	diag4	diag5	diag6	diag7	diag8	diag9	diag10	DM
9.	D819	V569	N497	O920	E146	M240	Z526	E620	S187	Z155	1

10.	X634	M545	F128	M473	D391	B736	L387	V638	C800	E106	1
11.	I130	U544	E110	Q410	H625	P330	Z654	M966	F785	L129	1
12.	W548	P770	K929	E103	Z747	G805	O239	P708	U992	W840	1
13.	J413	U880	B076	K921	E117	P650	V910	M413	D493	N804	1
14.	U158	M638	B708	Q323	Y234	G917	D668	I382	L748	A268	.
15.	N574	C216	M566	Q240	S958	W024	B033	A222	C830	S202	.
16.	K610	N105	X636	O871	Z730	G781	P741	C980	U376	Z937	.
17.	O169	F596	I339	K674	P798	Y920	G615	F341	J221	I137	.
18.	G323	C798	V560	N367	O920	K725	Y839	H253	H319	G366	.

```
. *We successfully tag all admissions with a diagnosis of diabetes at any position
```

The next step after defining the event is usually to capture all admissions for individuals with that event in your dataset (this just makes the dataset smaller and thus faster and easier to work with). You will see below why we need to keep all admissions for the individual, and not just those with the event of interest.

```
. sort id diag1
. list id DM if id == 5 | id == 6, separator(0)
```

	id	DM
39.	5	.
40.	5	.
41.	5	.
42.	5	.
43.	5	.
44.	5	.
45.	5	.
46.	5	.
47.	5	.
48.	5	.
49.	5	.
50.	5	.
51.	5	.
52.	5	.
53.	6	.
54.	6	.
55.	6	.
56.	6	.
57.	6	.
58.	6	.
59.	6	.
60.	6	.
61.	6	.
62.	6	1

```
. *Person 6 has an admission with diabetes.
. bysort id (diag1) : egen DMP = min(DM)
(858 missing values generated)
. *Stata treats missing values as infinity, so the minimum function works here
. list id DM DMP if id == 5 | id == 6, separator(0)
```

	id	DM	DMP
39.	5	.	.
40.	5	.	.
41.	5	.	.
42.	5	.	.
43.	5	.	.
44.	5	.	.
45.	5	.	.
46.	5	.	.
47.	5	.	.
48.	5	.	.

```
 49. |  5    .     .  |
 50. |  5    .     .  |
 51. |  5    .     .  |
 52. |  5    .     .  |
 53. |  6    .     1  |
 54. |  6    .     1  |
 55. |  6    .     1  |
 56. |  6    .     1  |
 57. |  6    .     1  |
 58. |  6    .     1  |
 59. |  6    .     1  |
 60. |  6    .     1  |
 61. |  6    .     1  |
 62. |  6    1     1  |
     +-----------------+
```

```
. keep if DMP == 1
(858 observations deleted)
. list id DM DMP if id == 5 | id == 6, separator(0)
```

```
     +-----------------+
     | id   DM   DMP   |
     |-----------------|
  1. |  6    .     1  |
  2. |  6    .     1  |
  3. |  6    .     1  |
  4. |  6    .     1  |
  5. |  6    .     1  |
  6. |  6    .     1  |
  7. |  6    .     1  |
  8. |  6    .     1  |
  9. |  6    .     1  |
 10. |  6    1     1  |
     +-----------------+
```

2.3 Same-day admissions

At this point, you will have a dataset with all admissions for any individual that ever had your event of interest. Next, we will drop "duplicate" admissions (in the sense they have the same admission and separation date as the prior admission) and keep information they contain. Deleting the admissions has a drawback: you lose information about the episode of care that you don't explicitly decide to keep, but you can always re-run the code and keep extra information if you find you need that information in the future.

```
. list, separator(0)
```

	id	admdate	sepdate	diag1
1.	1	01jan2020	01jan2020	I214
2.	1	01jan2020	01jan2020	I214
3.	1	01jan2020	01jan2020	E119

```
. gen MI = 1 if substr(diag1,1,3)=="I21"
(1 missing value generated)
. gen DM2= 1 if substr(diag1,1,3)=="E11"
(2 missing values generated)
. forval i = 1/4 {
  2. bysort id (admdate sepdate) : gen dup = 1 if admdate == admdate[_n-1] & sepdate == sepdate[_n-1
> ]
  3. bysort id (admdate sepdate) : replace dup =. if dup[_n-1]==1
  4. bysort id (admdate sepdate) : replace MI = 1 if MI[_n+1] == 1 & dup[_n+1] == 1
  5. bysort id (admdate sepdate) : replace DM2 = 1 if DM2[_n+1] == 1 & dup[_n+1] == 1
  6. list, separator(0)
  7. drop if dup == 1
  8. drop dup
  9. }
(1 missing value generated)
(1 real change made, 1 to missing)
(0 real changes made)
(0 real changes made)
```

	id	admdate	sepdate	diag1	MI	DM2	dup
1.	1	01jan2020	01jan2020	I214	1	.	.
2.	1	01jan2020	01jan2020	I214	1	.	1
3.	1	01jan2020	01jan2020	E119	.	1	.

```
(1 observation deleted)
(1 missing value generated)
(0 real changes made)
(0 real changes made)
(1 real change made)
```

	id	admdate	sepdate	diag1	MI	DM2	dup
1.	1	01jan2020	01jan2020	I214	1	1	.
2.	1	01jan2020	01jan2020	E119	.	1	1

```
(1 observation deleted)
(1 missing value generated)
(0 real changes made)
(0 real changes made)
(0 real changes made)
```

	id	admdate	sepdate	diag1	MI	DM2	dup
1.	1	01jan2020	01jan2020	I214	1	1	.

```
(0 observations deleted)
(1 missing value generated)
(0 real changes made)
(0 real changes made)
(0 real changes made)
```

	id	admdate	sepdate	diag1	MI	DM2	dup
1.	1	01jan2020	01jan2020	I214	1	1	.

```
(0 observations deleted)
```

We see that the code above iterates through each duplicate admission until finally we have a single admission.

Note we ran the loop for one more iteration than needed. This was to show that the code stops working when there are no more duplicate admissions. Usually, for large admissions datasets, you'll need to run 50-100 or more iterations to remove all the duplicate admissions (you'll know how many because every output will be "0 real changes made").

2.4 Nested admissions

The next step in data processing is to drop what we call "nested" admissions, which are admissions in the dataset that sit within a larger admission. For example:

```
. list, separator(0)
```

	id	admdate	sepdate	diag1	diag2
1.	1	01jan2020	08jan2020	I214	
2.	1	02jan2020	02jan2020	I214	E119
3.	1	03jan2020	03jan2020	I499	E119
4.	1	05jan2020	05jan2020	J152	E119
5.	1	06jan2020	12jan2020	I214	E119
6.	1	15jul2020	15jul2020	I509	E119

So we see here in row 1 that there is an admission lasting from 1 January to 8 January, but the next four "admissions" all occur before this admission is over. Additionally, these other admissions contain information that might be relevant for the index admission. So these admissions suggest the following:

- The individual presented to hospital with a mycardial infarction on 1 January.
- The individual has type 2 diabetes (E119; but we can't know if it is pre-existing or diagnosed in hospital).
- On 3 January, the individual was treated for cardiac arrhythmia (I499).
- On 5 January, the individual was treated for Pneumonia (J152).
- On 12 January, the individual was discharged.
- Later that year (15 July), the individual is admitted for heart failure (I509).

However, if we only used the admissions coded with I214, we would miss most of this information. So, we need to remove the nested admissions, but keep the information they contain. The steps are as follows:

- Tag nested admissions, defined as having an admission date before the separation date of the prior admission (you must therefore sort by admission and separation date, in that order).
- De-tag any admission that isn't the first nested admission in a set. It's easier to work on these one at a time.
- Collect the relevant information from the next admission and attach it to the first admission in the set.

```
. gen MI = 1 if substr(diag1,1,3)=="I21"
(3 missing values generated)
. gen DM2= 1 if substr(diag1,1,3)=="E11" | substr(diag2,1,3)=="E11"
(1 missing value generated)
. gen other_arr= 1 if substr(diag1,1,3)=="I49" | substr(diag2,1,3)=="I49"
(5 missing values generated)
. *Note in this example we are not interested in pneumonia, so we don´t collect that.
```

```
. forval i = 1/6 {
  2. bysort id (admdate sepdate) : gen nest = 1 if admdate < sepdate[_n-1] & sepdate[_n-1]!=.
  3. bysort id (admdate sepdate) : replace nest =. if nest[_n-1]==1
  4. bysort id (admdate sepdate) : replace MI = 1 if MI[_n+1] == 1 & nest[_n+1] == 1
  5. bysort id (admdate sepdate) : replace DM2 = 1 if DM2[_n+1] == 1 & nest[_n+1] == 1
  6. bysort id (admdate sepdate) : replace other_arr = 1 if other_arr[_n+1] == 1 & nest[_n+1] == 1
  7. bysort id (admdate sepdate) : replace sepdate = sepdate[_n+1] if sepdate[_n+1] > sepdate & nest
> [_n+1] == 1
  8. list, separator(0)
  9. drop if nest == 1
 10. drop nest
 11. }
(5 missing values generated)
(0 real changes made)
(0 real changes made)
(1 real change made)
(0 real changes made)
(0 real changes made)
```

	id	admdate	sepdate	diag1	diag2	MI	DM2	other_~r	nest
1.	1	01jan2020	08jan2020	I214		1	1	.	.
2.	1	02jan2020	02jan2020	I214	E119	1	1	.	1
3.	1	03jan2020	03jan2020	I499	E119	.	1	1	.
4.	1	05jan2020	05jan2020	J152	E119	.	1	.	.
5.	1	06jan2020	12jan2020	I214	E119	1	1	.	.
6.	1	15jul2020	15jul2020	I509	E119	.	1	.	.

```
(1 observation deleted)
(4 missing values generated)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(1 real change made)
(0 real changes made)
```

	id	admdate	sepdate	diag1	diag2	MI	DM2	other_~r	nest
1.	1	01jan2020	08jan2020	I214		1	1	1	.
2.	1	03jan2020	03jan2020	I499	E119	.	1	1	1
3.	1	05jan2020	05jan2020	J152	E119	.	1	.	.
4.	1	06jan2020	12jan2020	I214	E119	1	1	.	.
5.	1	15jul2020	15jul2020	I509	E119	.	1	.	.

```
(1 observation deleted)
(3 missing values generated)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(0 real changes made)
```

	id	admdate	sepdate	diag1	diag2	MI	DM2	other_~r	nest
1.	1	01jan2020	08jan2020	I214		1	1	1	.
2.	1	05jan2020	05jan2020	J152	E119	.	1	.	1
3.	1	06jan2020	12jan2020	I214	E119	1	1	.	.
4.	1	15jul2020	15jul2020	I509	E119	.	1	.	.

```
(1 observation deleted)
(2 missing values generated)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(1 real change made)
```

```
     | id     admdate     sepdate   diag1   diag2   MI   DM2   other_~r   nest |
     |-------------------------------------------------------------------------|
  1. |  1   01jan2020   12jan2020    I214            1     1          1      . |
  2. |  1   06jan2020   12jan2020    I214    E119    1     1          .      1 |
  3. |  1   15jul2020   15jul2020    I509    E119    .     1          .      . |
     +-------------------------------------------------------------------------+
(1 observation deleted)
(2 missing values generated)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(0 real changes made)

     | id     admdate     sepdate   diag1   diag2   MI   DM2   other_~r   nest |
     |-------------------------------------------------------------------------|
  1. |  1   01jan2020   12jan2020    I214            1     1          1      . |
  2. |  1   15jul2020   15jul2020    I509    E119    .     1          .      . |
     +-------------------------------------------------------------------------+
(0 observations deleted)
(2 missing values generated)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(0 real changes made)

     | id     admdate     sepdate   diag1   diag2   MI   DM2   other_~r   nest |
     |-------------------------------------------------------------------------|
  1. |  1   01jan2020   12jan2020    I214            1     1          1      . |
  2. |  1   15jul2020   15jul2020    I509    E119    .     1          .      . |
     +-------------------------------------------------------------------------+
(0 observations deleted)
```

We see, as with duplicate admissions, that the code above iterates through each nested admission until finally we have a single admission with the information we desire (note not all the information from the nested admissions is kept, only the information we defined).

Note again that we ran the loop for one more iteration than needed. This was to show that the code stops working when there are no more nested admissions. As for duplicates, with large admissions datasets, you'll need to run 50-100 or more iterations to remove all the nested admissions (and again, you'll know how many because every output will be "0 real changes made").

Finally, for both duplicate and nested admissions, you should make sure you keep *all* the information you need (separation and admission mode, procedures, all diagnoses you want, etc.)

2.5 Transfers

Finally, we tag transfer admissions, keep the information they contain, then drop them. Admission and separation mode become important here. The format and values for them will vary from dataset to dataset – we recommend reclassifying them into a single variable with three values: home (an actual separation from hospital); transfer (either within or between hospitals); and death. Note that these are often not coded perfectly, so clinical judgement often needs to be employed to determine the most likely outcome (for example, we once came across someone with more than 60 strokes in one month, all coded as having been admitted from home and separated to home, which is *unlikely*; in this case, we coded it as a single stroke).

```
. list, separator(0)
```

	id	admdate	sepdate	diag1	diag2	admmode	sepmode
1.	1	01jan2020	01jan2020	I214		0	1
2.	1	02jan2020	03jan2020	I214	E119	1	1
3.	1	03jan2020	05jan2020	I499	E119	1	0
4.	1	05jan2020	06jan2020	J152	E119	1	1
5.	1	06jan2020	10jan2020	I214	E119	1	0
6.	1	10jan2020	15jan2020	I214	E119	0	0
7.	1	15jul2020	15jul2020	I509	E119	1	2

```
. *Where: 0=home; 1=transfer; 2=death
```

We have deliberately made "mistakes" here to illustrate common aspects that are present in admissions data. Let's go through it line by line:

1. The individual is admitted to hospital with a myocardial infarction on 1 January 2020, then transferred on 1 January 2020.

2. This transfer is reflected in the next line, but the date is the next day. This is very likely a mistake, so this will be considered a transfer and not a new episode of care (because it is unlikely someone has a myocardial infarction and is discharged the same day; note also that it wouldn't really matter what the separation mode is – if this admission and the next one are both myocardial infarction, then we would consider them the same episode of care). The individual is then transferred again on 3 January 2020.

3. The diagnosis has changed, but this is still part of the same episode of care. The individual is then discharged home on 5 January.

4. Or are they? Given that the admission source for this line is a transfer (and there's also a final line (below) indicating probably the last period of care for this myocardial infarction) it's likely this is a transfer within the same episode of care. They then transfer on 6 January 2020

5. The individual appears to have discharged home again on 10 January 2020 and the next admission does not indicate a transfer either.

6. However, this next admission date is the same day as the previous one and for the same condition (myocardial infarction). This is where clinical judgement comes in – the question now is: is it more likely that this is a second event that happened on the same day as discharge, or is this a coding error and this is the same event? We cannot know from this data, but

on balance of probability, we have always treated readmissions for myocardial infarction on the same day as the same event (this will definitely vary for different clinical conditions, and is probably the most debateable step here). Assuming it's the same episode of care, the individual has transferred again on 10 January 2020 and then the individual discharges home on 15 January 2020.

7. The individual is readmitted on 15 July 2020, and dies the same day in hospital. Note that the admission source here is a transfer, but given there's no prior admission since January 2015, it's safe to assume this is a mistake (or there is missing data). This would work in the other direction too – i.e., if there was a separation coded as a transfer, but no admission for weeks or months, it's likely the person went home and there's a mistake (or again, missing data).

So, ultimately, we want to process this data into two lines for this individual, the first their full period of care for the MI, with any important information, and their second admissions for heart failure. The steps are as follows:

1. Tag potential transfers (either admission or separation mode indicating a transfer).
2. Define an actual transfer based on the distance between admissions. Here we allow a 1-day difference (you can increase or remove this based on your dataset error rate/if you are missing data, and your judgement).
3. Also define transfers if the admission is the same day as the previous separation (again, whether or not you include this step is up to your clinical judgement).
4. Move the important information onto the admission above the one coded as a transfer.
5. Drop transfers and repeat

Finally, note that sometimes getting rid of transfers can generate new nested admissions, so sometimes the processing of transfers needs to be combined with processing of nested admissions.

```
. gen MI = 1 if substr(diag1,1,3)=="I21"
(3 missing values generated)
. bysort id (admdate sepdate) : gen ptr = 1 if admmode==1 | sepmode[_n-1]==1
(2 missing values generated)
. bysort id (admdate sepdate) : gen transferdist = admdate-sepdate[_n-1]
(1 missing value generated)
. gen tr = 1 if ptr == 1 & inrange(transferdist,0,1)
(3 missing values generated)
. bysort id (admdate sepdate) : replace tr = 1 if transferdist==0 & (MI==1 | MI[_n-1]==1)
(1 real change made)
. list, separator(0)
```

	id	admdate	sepdate	diag1	diag2	admmode	sepmode	MI	ptr	transf~t	tr
1.	1	01jan2020	01jan2020	I214		0	1	1	.	.	.
2.	1	02jan2020	03jan2020	I214	E119	1	1	1	1	1	1
3.	1	03jan2020	05jan2020	I499	E119	1	0	.	1	0	1
4.	1	05jan2020	06jan2020	J152	E119	1	1	.	1	0	1
5.	1	06jan2020	10jan2020	I214	E119	1	0	1	1	0	1
6.	1	10jan2020	15jan2020	I214	E119	0	0	1	.	0	1
7.	1	15jul2020	15jul2020	I509	E119	1	2	.	1	182	.

```
. *All the transfers are coded as such.
. *Again, work on one at a time (in pairs)
. bysort id (admdate sepdate) : replace tr =. if tr[_n-1]==1
(2 real changes made, 2 to missing)
. bysort id (admdate sepdate) : replace MI = 1 if MI[_n+1]==1 & tr[_n+1]==1
(0 real changes made)
. bysort id (admdate sepdate) : replace sepdate = sepdate[_n+1] if tr[_n+1]==1
(3 real changes made)
. bysort id (admdate sepdate) : replace sepmode = sepmode[_n+1] if tr[_n+1]==1
(1 real change made)
. bysort id (admdate sepdate) : drop if tr == 1 & tr[_n-1]==.
(3 observations deleted)
. drop ptr tr transferdist
. list, separator(0)
```

	id	admdate	sepdate	diag1	diag2	admmode	sepmode	MI
1.	1	01jan2020	03jan2020	I214		0	1	1
2.	1	03jan2020	06jan2020	I499	E119	1	1	.
3.	1	06jan2020	15jan2020	I214	E119	1	0	1
4.	1	15jul2020	15jul2020	I509	E119	1	2	.

```
. *Repeat (with nested admissions cleared again, if necessary)
. forval i = 1/3 {
  2. bysort id (admdate sepdate) : gen ptr = 1 if admmode==1 | sepmode[_n-1]==1
  3. bysort id (admdate sepdate) : gen transferdist = admdate-sepdate[_n-1]
  4. gen tr = 1 if ptr == 1 & inrange(transferdist,0,1)
  5. bysort id (admdate sepdate) : replace tr = 1 if transferdist==0 & (MI==1 | MI[_n-1]==1)
  6. bysort id (admdate sepdate) : replace tr =. if tr[_n-1]==1
  7. bysort id (admdate sepdate) : replace MI = 1 if MI[_n+1]==1 & tr[_n+1]==1
  8. bysort id (admdate sepdate) : replace sepdate = sepdate[_n+1] if tr[_n+1]==1
  9. bysort id (admdate sepdate) : replace sepmode = sepmode[_n+1] if tr[_n+1]==1
 10. bysort id (admdate sepdate) : drop if tr == 1 & tr[_n-1]==.
 11. drop ptr tr transferdist
 12. list, separator(0)
 13. }
(1 missing value generated)
(1 missing value generated)
(2 missing values generated)
(0 real changes made)
(1 real change made, 1 to missing)
(0 real changes made)
(1 real change made)
(0 real changes made)
(1 observation deleted)
```

	id	admdate	sepdate	diag1	diag2	admmode	sepmode	MI
1.	1	01jan2020	06jan2020	I214		0	1	1
2.	1	06jan2020	15jan2020	I214	E119	1	0	1
3.	1	15jul2020	15jul2020	I509	E119	1	2	.

```
(1 missing value generated)
(1 missing value generated)
(2 missing values generated)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(1 real change made)
(1 real change made)
(1 observation deleted)
```

	id	admdate	sepdate	diag1	diag2	admmode	sepmode	MI

	id	admdate	sepdate	diag1	diag2	admmode	sepmode	MI
1.	1	01jan2020	15jan2020	I214		0	0	1
2.	1	15jul2020	15jul2020	I509	E119	1	2	.

```
(1 missing value generated)
(1 missing value generated)
(2 missing values generated)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(0 real changes made)
(0 observations deleted)
```

	id	admdate	sepdate	diag1	diag2	admmode	sepmode	MI
1.	1	01jan2020	15jan2020	I214		0	0	1
2.	1	15jul2020	15jul2020	I509	E119	1	2	.

And now we have two fully-processed episodes of care.

2.6 Summary

A summary of the key data processing steps is shown below.

1. Data harmonisation and checks
   ↓
2. Define an event of interest
   ↓
3. Identify and remove duplicate admissions
   ↓
4. Identify and remove nested admissions
   ↓
5. Identify and remove transfers

3 Full example

Below, we reproduce in full the code used to process the HESIN dataset associated with the UK Biobank study that we have used for research before (e.g., Morton et al., Value in Health, 2024). (The codes defining admission and discharge types are available at: https://biobank.ctsu.ox.ac.uk/crystal/coding.cgi?id=265 and https://biobank.ctsu.ox.ac.uk/crystal/coding.cgi?id=267 , respectively.)

The point of doing this was to define every MI event any individual had at any point during follow up. To do this, we needed to generate a dataset that had an accurate event date and discharge date for each MI. This is not that simple for some admissions (as seen in some examples above). For example, a person may initially admit to hospital, but then transfer between different wards or rehabilitation facilities. Each time they transfer, there's a new row in the dataset, but the diagnosis might change or remain the same. Thus, over their episode of care for the MI, it could appear as though this individual was discharged and readmitted with 3 or 4 MIs and/or other conditions, when in reality they had a single MI and received multiple forms of treatment.

3.1 Initial steps and data import

The first step is to import your dataset. Note you should *never modify the master file in any way.* Thus, we will import the master dataset, keep the variables we require, and save our own dataset which can be manipulated for the purposes of processing and subsequent analysis. The HESIN dataset is from the UK Biobank, and in this example we are using two different datasets, with a data dictionary available for download here:

- HESIN - Overal master table containing admission and discharge information
- HESIN_DIAG - Contains diagnosis codes relating to inpatient records

In most cases data provided to a research team will be in separate datasets, with a linkage key shared between all sets (in this case, the variable eid). For the case of HESIN and HESIN_DIAG, there is a one-to-many relationship, with single admissions in HESIN linked to multiple entries in HESIN_DIAG. The process below follows the same steps described above (see section 2):

1. Defining your event
2. Finding and removing same-day admissions while keeping the information they contain
3. Finding and removing "nested" admissions while keeping the information they contain
4. Finding and removing "transferred" admissions while keeping the information they contain

By defining and processing out cohort, we will create a dataset of discrete MI admissions across the cohort.

```
import delimited /home/jimb0w/Documents/UKB/hesin.txt, varnames(1) clear
keep eid ins_index epistart epiend admidate admisorc_uni disdate disdest_uni
save HESIN, replace
import delimited /home/jimb0w/Documents/UKB/hesin_diag.txt, varnames(1) clear
keep eid ins_index arr_index level diag_icd10
save HESIN_DIAG, replace
```

Now, while we can't show the actual data for privacy reasons, I will show a synthetic example to illustrate the structure of the data we are working with here.

```
set seed 23021917
clear
set obs 10
gen double eid = 12345678
replace eid = 23456789 if _n > 5
format eid %8.0f
bysort eid : gen ins_index = _n-1
gen epistart = td(1,1,2006) if _n == 1 | _n == 6
bysort eid (ins_index) : replace epistart = epistart[_n-1]+runiformint(1,1000) if _n!=1
gen epiend = epistart+rpoisson(10)
gen admidate=epistart
gen disdate=epien
gen admisorc_uni = 1000
gen disdest_uni = 1000
tostring epistart-disdate, replace format(%td) force
preserve
keep eid ins_index
save HesDeg, replace
restore

. list, separator(0)
```

	eid	ins_in~x	epistart	epiend	admidate	disdate	admiso~i	disdes~i
1.	12345678	0	01jan2006	16jan2006	01jan2006	16jan2006	1000	1000
2.	12345678	1	07mar2006	11mar2006	07mar2006	11mar2006	1000	1000
3.	12345678	2	09nov2006	17nov2006	09nov2006	17nov2006	1000	1000
4.	12345678	3	06nov2007	15nov2007	06nov2007	15nov2007	1000	1000
5.	12345678	4	08apr2010	13apr2010	08apr2010	13apr2010	1000	1000
6.	23456789	0	01jan2006	13jan2006	01jan2006	13jan2006	1000	1000
7.	23456789	1	31oct2006	08nov2006	31oct2006	08nov2006	1000	1000
8.	23456789	2	16feb2007	01mar2007	16feb2007	01mar2007	1000	1000
9.	23456789	3	21jun2008	30jun2008	21jun2008	30jun2008	1000	1000
10.	23456789	4	08oct2009	18oct2009	08oct2009	18oct2009	1000	1000

So we have the following variables for HESIN:

- *eid* – unique individual identifier.
- *ins_index* – instance index. Together with *eid* this uniquely identifies each admission in the dataset.
- *epistart* – episode start date.
- *epiend* – episode end date.
- *admidate* – admission start date.
- *disdate* – discharge date.
- *admisorc_uni* – admission source.
- *disdest_uni* – discharge destination.

We have excluded spell index or other variables as they are usually unavailable in other countries. I.e., the data is long and of the structure we have been dealing with above.

```
set seed 25101917
use HesDeg, clear
gen A = runiformint(1,3)
expand A
drop A
bysort eid ins_index : gen arr_index = _n-1
gen level = 1 if arr_index==0
recode level .=2
gen diag_icd10 = char(runiformint(65,90)) + string(runiformint(0,9)) + string(runiformint(0,9)) + st
> ring(runiformint(0,9))
```

```
. list, separator(0)
```

	eid	ins_in~x	arr_in~x	level	diag_~10
1.	12345678	0	0	1	M422
2.	12345678	0	1	2	K580
3.	12345678	0	2	2	K137
4.	12345678	1	0	1	M105
5.	12345678	1	1	2	K046
6.	12345678	2	0	1	I124
7.	12345678	2	1	2	M405
8.	12345678	2	2	2	M940
9.	12345678	3	0	1	W670
10.	12345678	4	0	1	L628
11.	23456789	0	0	1	G123
12.	23456789	1	0	1	E908
13.	23456789	2	0	1	X887
14.	23456789	3	0	1	J031
15.	23456789	3	1	2	I773
16.	23456789	3	2	2	V014
17.	23456789	4	0	1	T979

And for HESIN_DIAG:

- *eid* – unique individual identifier.
- *ins_index* – instance index.
- *arr_index* – array index. Together with *eid* and *ins_index* this uniquely identifies each observation in the dataset. I.e., each observation is a different level of ICD-10 code diagnosis for a given array index.
- *level* – classification of diagnosis (primary/secondary/external).
- *diag_icd10* – ICD-10 code for condition(s) diagnosed.

3.2 Defining your event

In this example, we want to create a cohort of people who were admitted for a myocardial infarction (MI). This means we want to use the diagnosis codes and only look at ICD-10 codes that were associated with this event, and importantly, that were the primary diagnosis for the admission. This allows us to capture MI admissions, rather than admissions where history of MI was coded as a secondary diagnosis. Some datasets may separate pre-admission and admission diagnosis codes which can also assist in understanding what the admission was for vs. what may have been present prior to the admission.

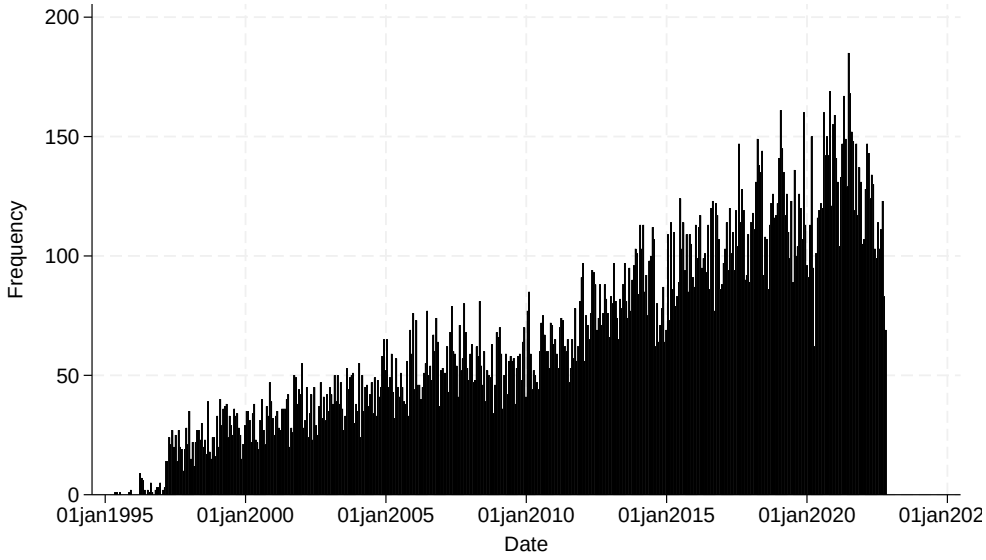
```
. use HESIN_DIAG, clear
. *Only keep primary diagnoses
. keep if level == 1
(12,953,647 observations deleted)
. ta arr_index
  arr_index |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |  4,149,189      100.00      100.00
------------+-----------------------------------
      Total |  4,149,189      100.00
. *Generate a variable that will tag diagnoses that define MI and then drop everything else
. gen MI = 1 if inrange(diag,"I21","I229")
(4,116,019 missing values generated)
. keep if MI == 1
(4,116,019 observations deleted)
. save hesinmi, replace
file hesinmi.dta saved
. use HESIN, clear
. merge 1:m eid ins_index using hesinmi
    Result                      Number of obs
    -----------------------------------------
    Not matched                     4,142,808
        from master                 4,142,808  (_merge==1)
        from using                          0  (_merge==2)
    Matched                            33,170  (_merge==3)
    -----------------------------------------
. *Of the more than 4,000,000 episodes, we have 33,170 where MI is coded as the primary diagnosis.
```

3.3 Process variables

We can now get episode start and end dates and transfer status into useable formats.

For transfer status, what each code represents is available via the links above. For our purposes, there are only two admission sources of interest – whether the admissions source is from the same or another hospital (i.e., a potential transfer) or not. Similarly, for discharge destination, there are three – home, a transfer, or death. We will categorise admission source and discharge distination as such.

Figure 3.1: MI episode start dates




```
. gen epist = date(epistart,"DMY")
(57,149 missing values generated)
. gen epien = date(epiend,"DMY")
(57,446 missing values generated)
. format epist epien %td
. *Check missing values and replace with admission and discharge dates if necesary
. count if epist==.
  57,149
. count if epist==. & admidate==""
  42
. replace epist = date(admidate,"DMY") if epist==.
(57,107 real changes made)
. drop if epist==.
(42 observations deleted)
. count if epien==.
  57,404
. count if epien==. & disdate==""
  314
. replace epien = date(disdate,"DMY") if epien==.
(57,090 real changes made)
. replace epien = epist if epien==.
(314 real changes made)
. *Check distributions
```

Figure 3.2: MI episode end dates

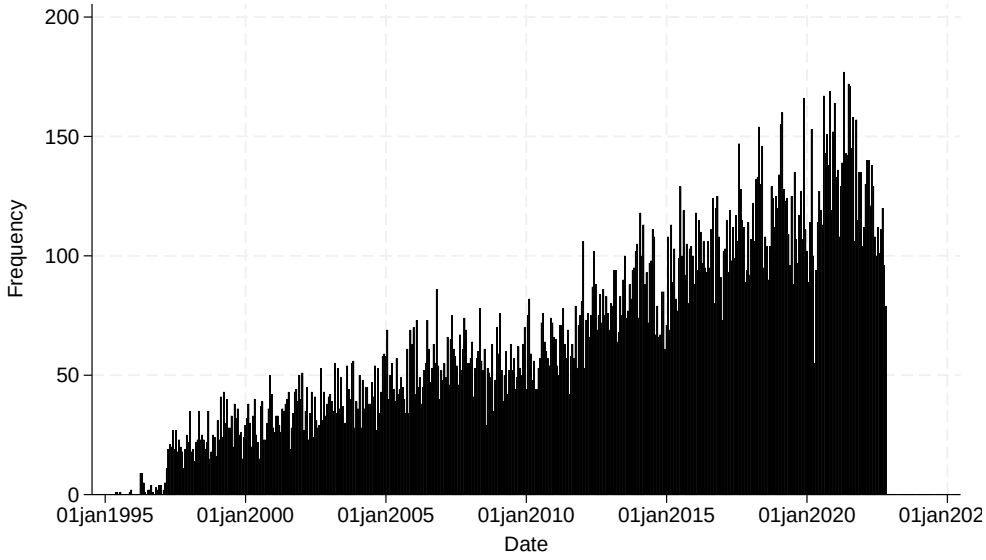
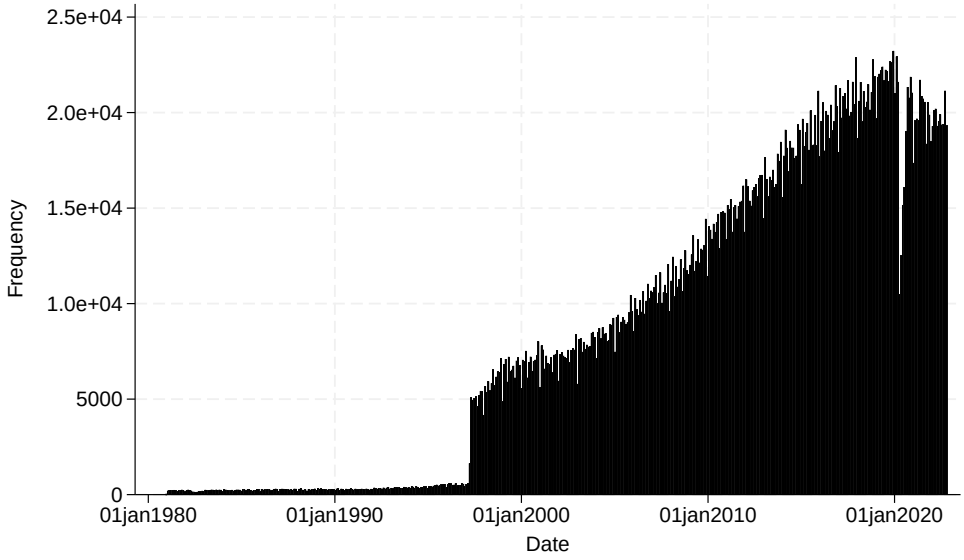
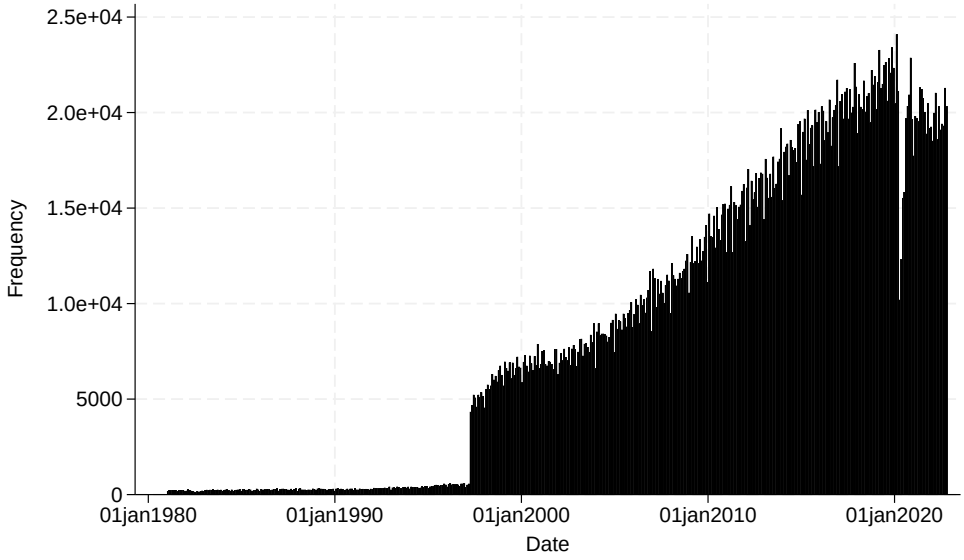


Figure 3.3: All episode start dates




```
. hist epist if MI == 1, color(black) graphregion(color(white)) ///
> frequency xtitle("Date") bin(500)
(bin=500, start=12906, width=20.086)
. hist epien if MI == 1, color(black) graphregion(color(white)) ///
> frequency xtitle("Date") bin(500)
(bin=500, start=12912, width=20.074)
. hist epist, color(black) graphregion(color(white)) ///
> frequency xtitle("Date") bin(500)
(bin=500, start=7641, width=30.616)
```

Figure 3.4: All episode end dates




```
. hist epien, color(black) graphregion(color(white)) ///
> frequency xtitle("Date") bin(500)
(bin=500, start=7671, width=30.556)

. gen admmode = 1

. replace admmode = 0 if inrange(admisorc,1000,2002) | inrange(admisorc,4000,4001) ///
>  | inrange(admisorc,7000,7003) | (admisorc >= 10000 & admisorc!=11000)
(4,046,768 real changes made)

. ta admmode
```

admmode	Freq.	Percent	Cum.
0	4,046,768	96.91	96.91
1	129,168	3.09	100.00
Total	4,175,936	100.00	

```
. *The vast majority are from home
. gen sepmode = 1

. replace sepmode = 0 if inrange(disdest,1000,2002) | inrange(disdest,4000,4001) ///
>  | inrange(disdest,7000,7003) | (disdest >= 10000 & disdest!=11000)
(3,765,066 real changes made)

. replace sepmode = 2 if disdest==11001
(19,565 real changes made)

. ta sepmode
```

sepmode	Freq.	Percent	Cum.
0	3,745,501	89.69	89.69
1	410,870	9.84	99.53
2	19,565	0.47	100.00
Total	4,175,936	100.00	

```
. *Again, the vast majority discharge home
```

Any dates you use should be checked. Visualisation is the simplest way to do this. What you're looking for is potential errors and to understand the shape of the data. For example, in Figure 3.1, we see a 'jump' in MI admissions around 1997 – likely inidicating one or more datasets begins

here (or earlier data was coded using ICD-9, or any other reason). We also see that the data falls off around 2022. We also note the seasonality of MI admissions and can see a dip in admissions around the time COVID-19 lockdowns started (becoming much more noticeable when including all admissions; Figure 3.3).

Also note there's a "problem" with the output of the admission source and discharge destination codes – within a given healthcare system, you would expect the number of episodes coded as having transferred on discharge should approximate the number of episodes coded as being admitted from a transfer (because they're transferring within a healthcare system). This is clearly not the case, meaning one of these fields is not "correctly" coded (correctly for our purposes, there may be a genuine reason these don't align). Just something to keep in mind when doing further processing.

3.4 Finding and removing "duplicate" and "nested" admissions

As above, we first remove admissions on the same day as the previous admissions, but keep the relevant information they contain. In doing so, we're making the assumption that two admissions on the same day for MI are related to a single event, and not two separate MI's. This seems like a reasonable assumption.

Then, the same for nested admissions. Note that both admission and separation date are updated in each iteration.

Both processes will be looped to capture individuals with multiple MI admissions. The number of loops is determined by running them until no further changes occur.

```
*Drop same day admissions, but keep the information they contain
forval i = 1/5 {
bysort eid (epist epien sepmode) : gen A = 1 if epist == epist[_n-1] & epien == epien[_n-1]
bysort eid (epist epien sepmode) : replace A =. if A[_n-1]==1
bysort eid (epist epien sepmode) : replace MI = 1 if MI[_n+1]==1 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]!=0 & A[_n+1]==1
drop if A == 1
drop A
}
*Drop nested admissions, but keep the information they contain.
forval i = 1/100 {
bysort eid (epist epien sepmode) : gen A = 1 if epist < epien[_n-1] & epien[_n-1]!=.
bysort eid (epist epien sepmode) : replace A =. if A[_n-1]==1
bysort eid (epist epien sepmode) : replace MI = 1 if MI[_n+1]==1 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epist = epist[_n+1] if epist[_n+1] < epist & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==1 & epien[_n+1]
>  > epien & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==2 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epien = epien[_n+1] if epien[_n+1] > epien & A[_n+1]==1
drop if A == 1
drop A
}
```

3.5 Identifying and removing transfers

First, it's worth looking at the time between admissions for potential transfers.

Figure 3.5: Time between admissions.

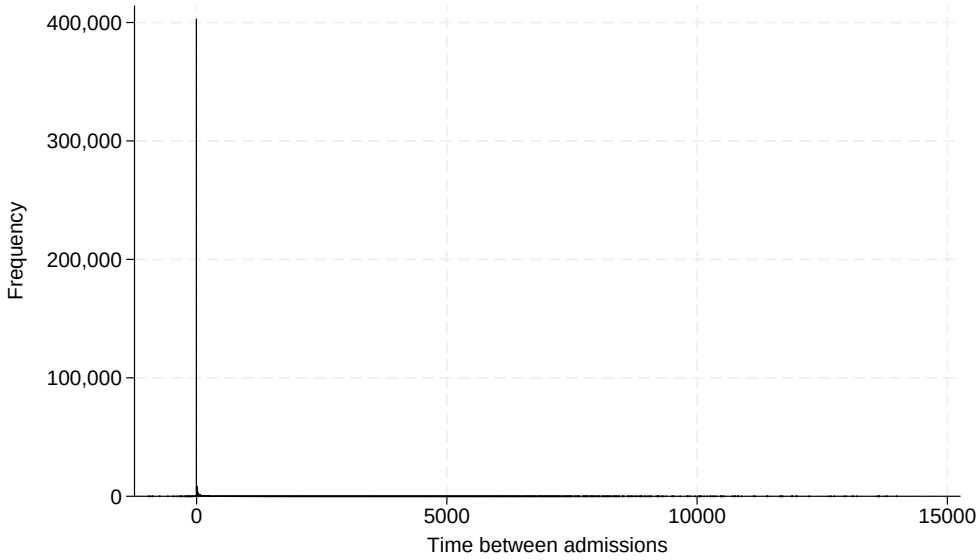
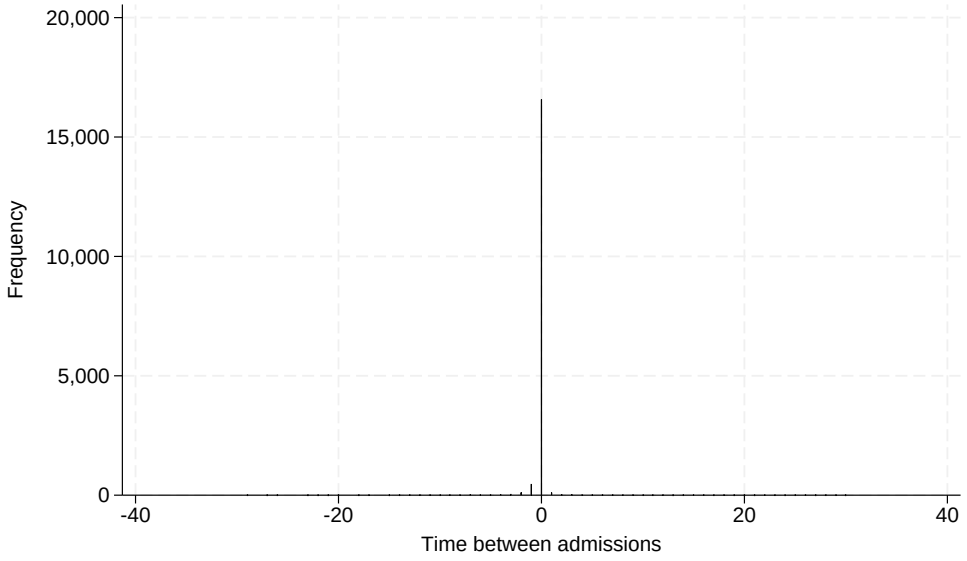
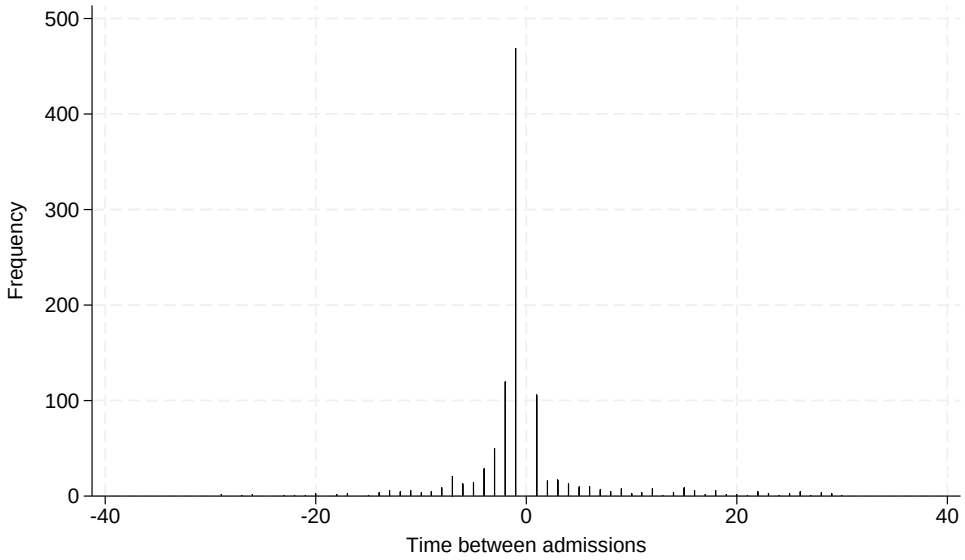


Figure 3.6: Time between admissions for an MI constrained to values between -30 and 30.




```
*Tag potential transfers
bysort eid (epist epien sepmode) : gen ptr = 1 if admmode==1 | sepmode[_n-1]==1
*Distance between transfers
bysort eid (epist epien sepmode) : gen transferdist = epist-epien[_n-1]
```

Figure 3.7: Time between admissions for an MI constrained to values between -30 and 30, excluding 0.




```
hist transferdist if ptr == 1, bin(1000) color(gs0) frequency graphregion(color(white)) ///
xtitle("Time between admissions") ylabel(,format(%9.0fc) angle(0))
hist transferdist if inrange(transferdist,-30,30) & ///
MI[_n-1]==1 & ptr == 1, ///
bin(1000) color(gs0) frequency graphregion(color(white)) ///
xtitle("Time between admissions") ylabel(,format(%9.0fc) angle(0))
hist transferdist if inrange(transferdist,-30,30) & transferdist!=0 & ///
MI[_n-1]==1 & ptr == 1, ///
bin(1000) color(gs0) frequency graphregion(color(white)) ///
xtitle("Time between admissions") ylabel(,format(%9.0fc) angle(0))
> luding 0.)
```

From these figures (Figures 3.5-**??**) we see that most transfers occur on the same day, or one day after, the seperation for the initial admission. At this point, we have to make a decision about how long we are going to allow between events. Given that it appears that 16,000 transfer admissions have a transfer date that matches the previous admission date, compared to just 100 with that distance being 1 day, and even fewer with 2, 3, etc., it doesn't really matter if we set a "buffer" (i.e., a period of time where we assume that a transfer did occur even if the dates don't match perfectly or that we are missing data). Thus, here, we will assume the data coding is not perfect, and if someone has transferred within 1 day, we will assume that it is indeed a transfer, and not a new admission.

Moreover, if the next episode of care occurs on the exact same day as the original admission, we are going to assume it is more likely that this is an admission where the transfer has been mis-coded (or gain, that we have missing data) than it is a new admission.

```
gen tr = 1 if ptr == 1 & inrange(transferdist,0,1)
bysort eid (epist epien sepmode) : replace tr = 1 if transferdist==0 & (MI==1 | MI[_n-1]==1)
*Only deal with one at a time
bysort eid (epist epien sepmode) : replace tr =. if tr[_n-1]==1
*Drop transfers, but keep the information they contain
bysort eid (epist epien sepmode) : replace MI = 1 if MI[_n+1]==1 & tr[_n+1]==1
bysort eid (epist epien sepmode) : replace epien = epien[_n+1] if tr[_n+1]==1
```

```
bysort eid (epist epien sepmode) : drop if tr == 1 & tr[_n-1]==.
drop ptr tr transferdist
*This introduces new nested admissions, so we need to cycle between dropping transfers and nested ad
> missions
bysort eid (epist epien sepmode) : gen A = 1 if epist < epien[_n-1] & epien[_n-1]!=.
bysort eid (epist epien sepmode) : replace A =. if A[_n-1]==1
bysort eid (epist epien sepmode) : replace MI = 1 if MI[_n+1]==1 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epist = epist[_n+1] if epist[_n+1] < epist & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==1 & epien[_n+1]
>  > epien & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==2 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epien = epien[_n+1] if epien[_n+1] > epien & A[_n+1]==1
drop if A == 1
drop A
forval i = 1/100 {
bysort eid (epist epien sepmode) : gen ptr = 1 if admmode==1 | sepmode[_n-1]==1
bysort eid (epist epien sepmode) : gen transferdist = epist-epien[_n-1]
gen tr = 1 if ptr == 1 & inrange(transferdist,0,1)
bysort eid (epist epien sepmode) : replace tr = 1 if transferdist==0 & (MI==1 | MI[_n-1]==1)
bysort eid (epist epien sepmode) : replace tr =. if tr[_n-1]==1
bysort eid (epist epien sepmode) : replace MI = 1 if MI[_n+1]==1 & tr[_n+1]==1
bysort eid (epist epien sepmode) : replace epien = epien[_n+1] if tr[_n+1]==1
bysort eid (epist epien sepmode) : drop if tr == 1 & tr[_n-1]==.
drop ptr tr transferdist
bysort eid (epist epien sepmode) : gen A = 1 if epist < epien[_n-1] & epien[_n-1]!=.
bysort eid (epist epien sepmode) : replace A =. if A[_n-1]==1
bysort eid (epist epien sepmode) : replace MI = 1 if MI[_n+1]==1 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epist = epist[_n+1] if epist[_n+1] < epist & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==1 & epien[_n+1]
>  > epien & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==2 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epien = epien[_n+1] if epien[_n+1] > epien & A[_n+1]==1
drop if A == 1
drop A
}
*And that´s it.
keep if MI == 1
keep eid epist epien MI
save AllMI, replace
```

3.6 Checking the processed dataset

As mentioned above, the whole time you are developing this code you should be carefully watching what it does and seeing if it makes sense. But now that it's "done", it's also worth doing some final checks to see if we can detect any final errors.

First, we check that no events have been dropped completely.

```
use AllMI, clear
collapse (sum) MI, by(eid)
rename MI nMI
save tempcheck, replace
use HESIN, clear
merge 1:m eid ins_index using hesinmi
drop _merge
collapse (sum) MI, by(eid)
merge 1:1 eid using tempcheck
drop _merge
recode nMI .=0

. count if MI < nMI
  0
. count if MI > nMI & nMI==0
  0
. erase tempcheck.dta
```

Next, we check there aren't any duplicate events and that the number of events per person seems reasonable.

```
. use AllMI, clear
. bysort eid epist : gen njm = _n
. ta njm
```

njm	Freq.	Percent	Cum.
1	18,289	100.00	100.00
Total	18,289	100.00	

```
. bysort eid (epist) : gen nj = _N
. ta nj
```

nj	Freq.	Percent	Cum.
1	14,216	77.73	77.73
2	2,794	15.28	93.01
3	783	4.28	97.29
4	304	1.66	98.95
5	105	0.57	99.52
6	54	0.30	99.82
7	7	0.04	99.86
8	8	0.04	99.90
18	18	0.10	100.00
Total	18,289	100.00	

It's possible that there is an error for the individual with 18 MIs, but it's also entirely possible that someone had 18 MIs during 10 years of follow-up. We can't show this, but at this point it's worth examining the complete admission history for the individuals with high numbers of admissions to see if you have missed anything obvious, or if the result is reasonable. In our case, it does indeed look like an individual with 18 MIs.

Finally, we check the overall dataset.

```
. use AllMI, clear
```

Figure 3.8: Histogram of date of admission for MI.

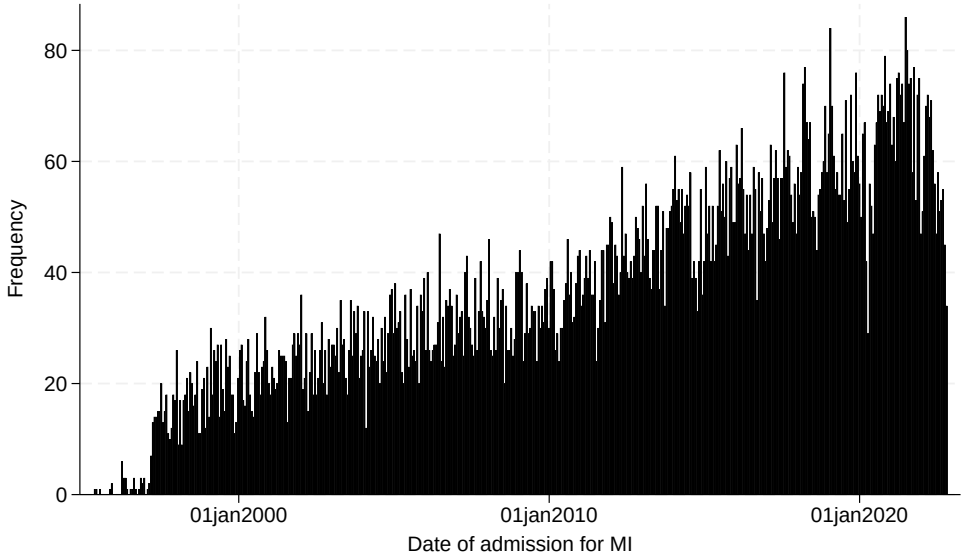
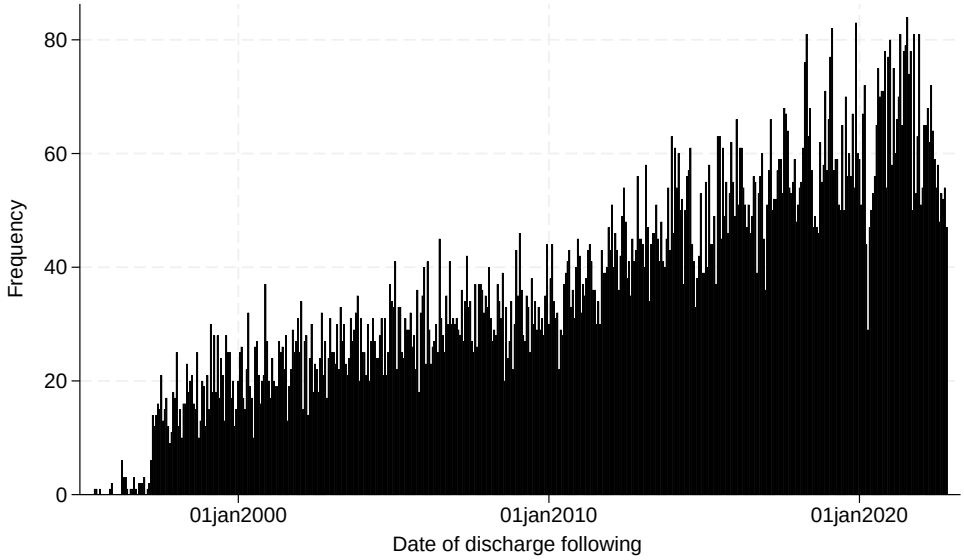
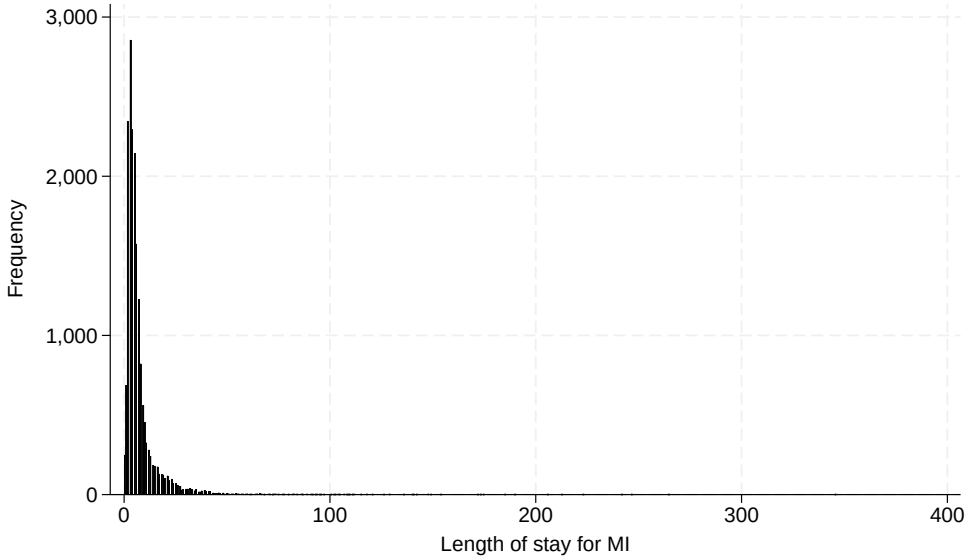


Figure 3.9: Histogram of date of discharge following MI.




```
. hist epist, bin(500) color(gs0) frequency graphregion(color(white)) ///
> xtitle("Date of admission for MI") ylabel(,format(%9.0fc) angle(0)) ///
> tlabel(01jan2000 01jan2010 01jan2020)
(bin=500, start=12906, width=20.084)
. hist epien, bin(500) color(gs0) frequency graphregion(color(white)) ///
> xtitle("Date of discharge following `i´") ylabel(,format(%9.0fc) angle(0)) ///
> tlabel(01jan2000 01jan2010 01jan2020)
(bin=500, start=12912, width=20.074)
```

Figure 3.10: Histogram of length of stay for an MI.




```
. gen LOS = epien-epist
. count if LOS < 0
  0
. hist LOS, bin(500) color(gs0) frequency graphregion(color(white)) ///
> xtitle("Length of stay for MI") ylabel(,format(%9.0fc) angle(0))
(bin=500, start=0, width=.692)
. su LOS, detail
```

LOS

Percentiles		Smallest		
1%	0	0		
5%	1	0		
10%	2	0	Obs	18,289
25%	3	0	Sum of wgt.	18,289
50%	5		Mean	7.640932
		Largest	Std. dev.	11.13823
75%	8	242		
90%	16	247	Variance	124.0602
95%	23	265	Skewness	8.785268
99%	48	346	Kurtosis	147.2561

These 0-day MIs seem like an error, but after manually checking a few of them in the unprocessed dataset, they appear to be "real".

3.7 Effects of processing data

So, we just spent a long time doing a lot of data processing. Many people just analyse admissions as an outcome in themselves, so it's worth emphasising why you would perform data processing like this. One situtation where I can think of why you wouldn't do this would be analysing time to first event, where the only date of relevance is the episode start date. In most other situations when analysing events or episodes of care, including studying anything to do with what happens after an event, most of the data processing outlined above is necessary.

Let's take a look at how necessary it really is.

```
. use hesinmi, clear
. count if MI == 1
  33,170
. local A = r(N)
. use AllMI, clear
. count if MI == 1
  18,289
. di round(100-100*r(N)/`A´,1)
45
```

Clearly necessary in our case – the difference between the unprocessed and processed data is almost double (i.e., any analysis that didn't involve data processing would overestimate the rate of MI almost two-fold). This effect of overcounting has been previously shown in Australian linked data Lopez et al., BMJ Open, 2017).

Now it's worth checking this for a few other outcomes to emphasise how variable the effects of data processing are. We will repeat the data processing for the following primary admission diagnoses (ICD-10 codes):

- Lung cancer (C34)
- Heart failure (I50)
- Stroke (I60-I64)
- Pneumonia (J44)
- Acute Kidney Failure (N17)
- Head injury (S00-S09)

Notice here we skip the data checking steps and just include the same rules as for MI – this is for brevity in this document only, don't do this for real studies.

```
foreach oc in LC HF ST PN AK HI {
use HESIN_DIAG, clear
keep if level == 1
if "`oc´" == "LC" {
gen OC = 1 if substr(diag,1,3)=="C34"
}
if "`oc´" == "HF" {
gen OC = 1 if substr(diag,1,3)=="I50"
}
if "`oc´" == "ST" {
gen OC = 1 if inrange(diag,"I60","I649")
}
if "`oc´" == "PN" {
gen OC = 1 if substr(diag,1,3)=="J44"
}
if "`oc´" == "AK" {
gen OC = 1 if substr(diag,1,3)=="N17"
}
if "`oc´" == "HI" {
gen OC = 1 if inrange(diag,"S00","S099")
}
keep if OC == 1
save hesin`oc´, replace
use HESIN, clear
```

```
merge 1:m eid ins_index using hesin`oc´
gen epist = date(epistart,"DMY")
gen epien = date(epiend,"DMY")
format epist epien %td
count if epist==.
count if epist==. & admidate==""
replace epist = date(admidate,"DMY") if epist==.
drop if epist==.
count if epien==.
count if epien==. & disdate==""
replace epien = date(disdate,"DMY") if epien==.
replace epien = epist if epien==.
gen admmode = 1
replace admmode = 0 if inrange(admisorc,1000,2002) | inrange(admisorc,4000,4001) ///
 | inrange(admisorc,7000,7003) | (admisorc >= 10000 & admisorc!=11000)
gen sepmode = 1
replace sepmode = 0 if inrange(disdest,1000,2002) | inrange(disdest,4000,4001) ///
 | inrange(disdest,7000,7003) | (disdest >= 10000 & disdest!=11000)
replace sepmode = 2 if disdest==11001
forval i = 1/5 {
bysort eid (epist epien sepmode) : gen A = 1 if epist == epist[_n-1] & epien == epien[_n-1]
bysort eid (epist epien sepmode) : replace A =. if A[_n-1]==1
bysort eid (epist epien sepmode) : replace OC = 1 if OC[_n+1]==1 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]!=0 & A[_n+1]==1
drop if A == 1
drop A
}
forval i = 1/100 {
bysort eid (epist epien sepmode) : gen A = 1 if epist < epien[_n-1] & epien[_n-1]!=.
bysort eid (epist epien sepmode) : replace A =. if A[_n-1]==1
bysort eid (epist epien sepmode) : replace OC = 1 if OC[_n+1]==1 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epist = epist[_n+1] if epist[_n+1] < epist & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==1 & epien[_n+1]
>  > epien & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==2 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epien = epien[_n+1] if epien[_n+1] > epien & A[_n+1]==1
drop if A == 1
drop A
}
bysort eid (epist epien sepmode) : gen ptr = 1 if admmode==1 | sepmode[_n-1]==1
bysort eid (epist epien sepmode) : gen transferdist = epist-epien[_n-1]
gen tr = 1 if ptr == 1 & inrange(transferdist,0,1)
bysort eid (epist epien sepmode) : replace tr = 1 if transferdist==0 & (OC==1 | OC[_n-1]==1)
bysort eid (epist epien sepmode) : replace tr =. if tr[_n-1]==1
bysort eid (epist epien sepmode) : replace OC = 1 if OC[_n+1]==1 & tr[_n+1]==1
bysort eid (epist epien sepmode) : replace epien = epien[_n+1] if tr[_n+1]==1
bysort eid (epist epien sepmode) : drop if tr == 1 & tr[_n-1]==.
drop ptr tr transferdist
bysort eid (epist epien sepmode) : gen A = 1 if epist < epien[_n-1] & epien[_n-1]!=.
bysort eid (epist epien sepmode) : replace A =. if A[_n-1]==1
bysort eid (epist epien sepmode) : replace OC = 1 if OC[_n+1]==1 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epist = epist[_n+1] if epist[_n+1] < epist & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==1 & epien[_n+1]
>  > epien & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==2 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epien = epien[_n+1] if epien[_n+1] > epien & A[_n+1]==1
drop if A == 1
drop A
forval i = 1/100 {
bysort eid (epist epien sepmode) : gen ptr = 1 if admmode==1 | sepmode[_n-1]==1
bysort eid (epist epien sepmode) : gen transferdist = epist-epien[_n-1]
gen tr = 1 if ptr == 1 & inrange(transferdist,0,1)
bysort eid (epist epien sepmode) : replace tr = 1 if transferdist==0 & (OC==1 | OC[_n-1]==1)
bysort eid (epist epien sepmode) : replace tr =. if tr[_n-1]==1
bysort eid (epist epien sepmode) : replace OC = 1 if OC[_n+1]==1 & tr[_n+1]==1
bysort eid (epist epien sepmode) : replace epien = epien[_n+1] if tr[_n+1]==1
bysort eid (epist epien sepmode) : drop if tr == 1 & tr[_n-1]==.
drop ptr tr transferdist
```

```
bysort eid (epist epien sepmode) : gen A = 1 if epist < epien[_n-1] & epien[_n-1]!=.
bysort eid (epist epien sepmode) : replace A =. if A[_n-1]==1
bysort eid (epist epien sepmode) : replace OC = 1 if OC[_n+1]==1 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epist = epist[_n+1] if epist[_n+1] < epist & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==1 & epien[_n+1]
>  > epien & A[_n+1]==1
bysort eid (epist epien sepmode) : replace sepmode = sepmode[_n+1] if sepmode[_n+1]==2 & A[_n+1]==1
bysort eid (epist epien sepmode) : replace epien = epien[_n+1] if epien[_n+1] > epien & A[_n+1]==1
drop if A == 1
drop A
}
keep if OC == 1
keep eid epist epien OC
save All`oc´, replace
}
```

```
. foreach oc in LC HF ST PN AK HI {
  2. di "`oc´"
  3. use hesin`oc´, clear
  4. count if OC == 1
  5. local A = r(N)
  6. use All`oc´, clear
  7. count if OC == 1
  8. di round(100-100*r(N)/`A´,1)
  9. }
LC
  29,274
  26,389
10
HF
  16,486
  9,509
42
ST
  30,569
  16,233
47
PN
  21,029
  12,334
41
AK
  9,866
  5,773
41
HI
  21,957
  17,736
19
```

In summary, process your data.