A Small Example

David Wu

2024-10-23

Here, we will clean and process a small database into a temporal network of patient transfers.

All the functionality is provided in the `hospinet` library.

```
import hospinet

import polars as pl
import networkx as nx

from matplotlib import pyplot as plt # for later
```

We load the database (in the form of a csv), and clean it using the `hospinet.cleaner` submodule. We specify the column names, since they are different from the (standardised) default.

```
source_db = hospinet.cleaner.ingest_csv("./data/admissions.csv", convert_dates=True)
source_db.head(5)
```

patient i64	hospital str	admission datetime[s]	discharge datetime[s]
3	"D"	2020-04-02 02:56:19.917759	2021-06-04 14:58:15.715780
2	"E"	2020-04-10 13:56:21.043894	2021-06-07 14:35:45.036138
3	"B"	2020-04-15 03:22:06.938845	2020-05-27 10:14:20.476332
2	"A"	2020-04-20 20:33:17.858555	2020-05-23 10:31:03.186415
1	"E"	2020-04-26 03:36:11.245264	2021-01-08 13:36:08.182081

```
cleaned_db = hospinet.cleaner.clean_database(
    source_db,
    delete_missing="record",
    delete_errors="record",
```

```python
    subject_id = 'patient',
    facility_id = 'hospital',
    admission_date = 'admission',
    discharge_date = 'discharge',
    retain_auxiliary_data = True,
)
# encode dates to numerics
first_date = cleaned_db.select(pl.col('Adate').min()).item()
numeric_db = hospinet.cleaner.normalise_dates(
    cleaned_db,
    cols = ['Adate', 'Ddate'],
    ref_date = first_date
)
```

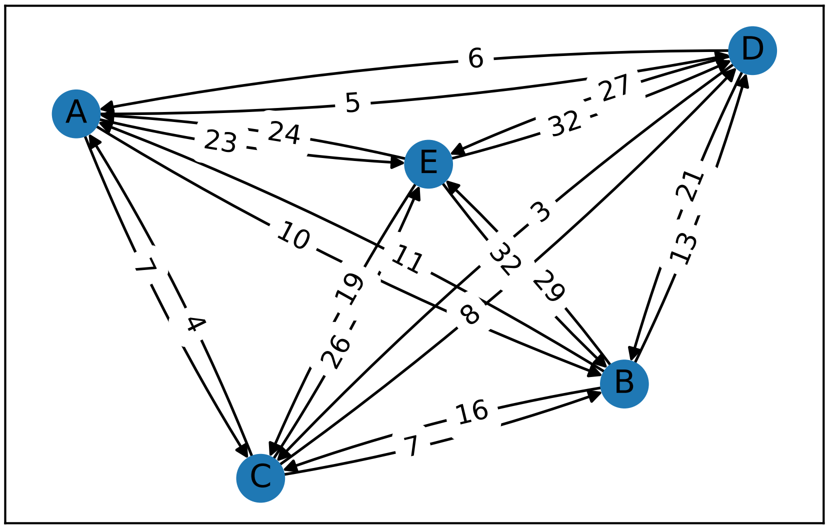
```
Checking existence of columns...
Column existence OK.
Standardising column names...
Coercing types...
Type coercion done.
Checking for missing values...
Checking for erroneous records...
Removing duplicate records...
Finding and fixing overlapping records...
Iteration 0 : 144 entries; 92 overlaps; 0.00986635300796479 s
Iteration 1 : 171 entries; 62 overlaps; 0.007867809967137873 s
Iteration 2 : 194 entries; 50 overlaps; 0.008401825092732906 s
Iteration 3 : 215 entries; 44 overlaps; 0.008451727917417884 s
Iteration 4 : 234 entries; 38 overlaps; 0.008249283069744706 s
Iteration 5 : 250 entries; 34 overlaps; 0.00763934594579041 s
Iteration 6 : 265 entries; 32 overlaps; 0.008349796058610082 s
Iteration 7 : 276 entries; 28 overlaps; 0.007633648929186165 s
Iteration 8 : 286 entries; 21 overlaps; 0.007548433030024171 s
Iteration 9 : 294 entries; 18 overlaps; 0.008352378034032881 s
Iteration 10 : 302 entries; 16 overlaps; 0.008632156997919083 s
Iteration 11 : 310 entries; 16 overlaps; 0.008329477976076305 s
Iteration 12 : 317 entries; 15 overlaps; 0.006984411971643567 s
Iteration 13 : 324 entries; 14 overlaps; 0.007457658066414297 s
Iteration 14 : 330 entries; 13 overlaps; 0.007909717969596386 s
Iteration 15 : 335 entries; 12 overlaps; 0.008741413010284305 s
Iteration 16 : 339 entries; 9 overlaps; 0.008681689971126616 s
Iteration 17 : 243 entries; 7 overlaps; 0.008594542043283582 s
Iteration 18 : 188 entries; 5 overlaps; 0.008171394933015108 s
Iteration 19 : 120 entries; 4 overlaps; 0.007406029035337269 s
Iteration 20 : 122 entries; 4 overlaps; 0.006484979996457696 s
Iteration 21 : 123 entries; 3 overlaps; 0.008550931001082063 s
Iteration 22 : 124 entries; 2 overlaps; 0.006678613950498402 s
Iteration 23 : 125 entries; 2 overlaps; 0.008923328015953302 s
Iteration 24 : 126 entries; 2 overlaps; 0.0072985399747267365 s
Iteration 25 : 127 entries; 2 overlaps; 0.007936462992802262 s
Iteration 26 : 128 entries; 2 overlaps; 0.007584752049297094 s
Iteration 27 : 129 entries; 2 overlaps; 0.007430013967677951 s
Iteration 28 : 130 entries; 2 overlaps; 0.007733201025985181 s
Iteration 29 : 131 entries; 2 overlaps; 0.007016850053332746 s
Iteration 30 : 132 entries; 2 overlaps; 0.0073259149212390184 s
Iteration 31 : 133 entries; 2 overlaps; 0.006115732016041875 s
Iteration 32 : 134 entries; 2 overlaps; 0.006184889003634453 s
Iteration 33 : 135 entries; 2 overlaps; 0.006999772973358631 s
Iteration 34 : 135 entries; 1 overlaps; 0.007312481990084052 s
Iteration 35 : 0 entries; 0 overlaps; 0.008117643068544567 s
History of non-overlapping patient records:
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 99, 57, 70, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0 overlaps remaining after iterations...
```

We can then process this into a TemporalNetwork object directly using the `from_presence` class method

```python
network = hospinet.TemporalNetwork.from_presence(
    numeric_db,
    discretisation=1,
)
```

This object is also a `networkx.DiGraph` object, so we can use the native plotting functionality. We project this down to static nodes first, so that we don't get the full temporal graph.

```python
static_network = network.to_static()
pos = nx.spring_layout(static_network, seed=1451)
edge_labels = {
        tuple(edge): f"{attr}"
        for *edge, attr in static_network.edges(data='weight')
    }
nx.draw_networkx(static_network, pos=pos, connectionstyle='arc3,rad=0.05')
nx.draw_networkx_edge_labels(
    static_network,
    pos=pos,
    edge_labels=edge_labels,
    label_pos=0.4,
    connectionstyle='arc3,rad=0.05'
);
```



We provide some basic indexing support via methods:

```
print("Nodes with presence at time 15: ", network.nodes_at_time(15))
print("When hospital D is occupied: ", network.when_present('D'))
```

```
Nodes with presence at time 15:  [('E', 15), ('B', 15)]
When hospital D is occupied:  [('D', 0), ('D', 1), ('D', 2), ('D', 3), ('D', 4), ('D', 5), (
```