1.1b_create_time_based_cohort

May 26, 2021

1 1.1b_create_time_based_cohort

- postgres = 9.4 (likely that this does not make much difference)

1.0.1 Create a table in Postgres of admissions (and corresponding notes) related to heparin time periods

1.1.0 Import hadm_ids for which time key exists 1.1.1 Keep only adults from Carevue 1.1.2 Pull admissions that match the hadm_ids with time key 1.1.3 Pull admit times/ discharge times from admissions table 1.1.4 Save to a new table in posgres 1.1.5 Get notes for those admissions 1.1.6 Calculate which admissions have any leap days 1.1.7 Calculate and apply time shift to all dates to get true dates 1.1.8 Double check the time shifts are correct (compare deltas) 1.1.9 Make tables for notes in this study

[ ]:
```
# only run this cell if you need to reset the connection to postgres database␣
↪after an error
conn.commit();
cur.close();
conn.close();
```

1.1 import libraries, connect to mimic database

[ ]:
```
import sys
import os
from datetime import datetime, timedelta
from tqdm import tnrange, tqdm_notebook
from time import sleep
import time
import re

import pandas as pd

import psycopg2
from sqlalchemy import create_engine, update, event

from importlib_metadata import version
```

[ ]:
```
PASSWORD = os.environ.get("PASSWORD")
USERNAME = os.environ.get("USERNAME")
POSTGRES_CONNECT = os.environ.get("POSTGRES_CONNECT")
POSTGRES_ENGINE = os.environ.get("POSTGRES_ENGINE")

conn = psycopg2.connect(POSTGRES_CONNECT)
engine = create_engine(POSTGRES_ENGINE)

cur = conn.cursor();
cur.execute("""SET search_path = mimiciii;""")

libraries = ['pandas','sqlalchemy','psycopg2','tqdm']
print('last ran: ',datetime.now() )
print("Python Version:", sys.version[0:7])
print( "operating system:", sys.platform)

for lib in libraries:
    print(lib + ' version: ' + version(lib))
```

1.1.1 1.1.0 import the hadmids where a time key exists

- year_key.csv is the file that contained the dates and hadm_id

[ ]:
```
key = pd.read_csv(path1 + 'year_key.csv')

key['hadm_id'] = key['hadm_id'].astype(str)
len(key) # number of keys
```

1.1.2 1.1.1 create a dictionary of keys and ids for our cohort

- get adult hadm_ids from the inputevents_cv_adult table
- keep only the hadm_ids that have a key in the year_key.csv file

[ ]:
```
# load hadm_ids from adult carevue inputs
adult_ids = pd.read_sql(""" SELECT DISTINCT(hadm_id)
FROM mimiciii.inputevents_cv_adult;""", engine)

# drop nans
adult_ids.dropna(inplace=True)

adult_ids['hadm_id'] = adult_ids['hadm_id'].astype(int)
adult_ids['hadm_id'] = adult_ids['hadm_id'].astype(str)
print('number of adult carevue ids ' + str(len(adult_ids)))

# get ids that are in both the key and the adult carevue inputs via an inner␣
 ↪join
adult_keys = key.merge(adult_ids, how='inner',on='hadm_id')
```

```
print('number of adult carevue ids with a key ' + str(len(adult_keys)))

key_ids = tuple(adult_keys.hadm_id.values)

# make key dict
key_dict = dict(zip(list(adult_keys['hadm_id'].astype(int)),␣
 ↪list(adult_keys['year'])))
```

1.1.3 1.1.2 load the data from the admissions table for ids in the key_dict

[ ]:
```
sql_q = """
SELECT *
FROM mimiciii.admissions
WHERE hadm_id IN {0}"""

sql = sql_q.format(key_ids)
df=pd.read_sql(sql, engine)
```

1.1.4 1.1.3 Get admittimes/dischtimes for each hadm_id

[ ]:
```
# make keys for admittime and dischtime
hadm_admit_key = dict(list(zip(list(df['hadm_id']), list(df['admittime']))))
hadm_disch_key = dict(list(zip(list(df['hadm_id']), list(df['dischtime']))))

# add a true year column
df['true_year'] = df['hadm_id'].map(key_dict)
#create a new table of ids and year
df_ids_times = df.loc[:,['hadm_id','true_year']]
```

1.1.5 1.1.4 make a new table in postgres of hadm_ids and keys

[ ]:
```
cur.execute("""
DROP TABLE IF EXISTS mimiciii.cv_real_dates;

CREATE TABLE mimiciii.cv_real_dates
(hadm_id int,
true_year int);""")

conn.commit()
```

save the data from our new table in posgres

[ ]:
```
df_ids_times.to_sql('cv_real_dates', con=engine, if_exists='append',␣
 ↪chunksize=1, index=False, schema='mimiciii')
```

1.1.6 1.1.5 get notes

- put notes and note times for adult cv admissions into a table

[ ]:
```
cur.execute("""
DROP TABLE IF EXISTS mimiciii.adult_cv_notes;

SELECT  B.*
INTO mimiciii.adult_cv_notes
    FROM mimiciii.noteevents B
    WHERE B.hadm_id IN (
            SELECT  x.hadm_id
            FROM mimiciii.cv_real_dates x)

    AND B.iserror IS NULL
;""")

conn.commit()
```

[ ]:
```
cur.execute("""SELECT COUNT(*), COUNT(DISTINCT hadm_id) FROM  mimiciii.
 ↪adult_cv_notes;""")
ncount=cur.fetchall()
print( pd.DataFrame(ncount, columns=[ 'total notes count','admissions']).
 ↪to_string(index=False))
```

[ ]:
```

```

1.1.7 1.1.6 Calculate which admissions were on leap days

[ ]:
```
note_df = pd.read_sql('SELECT * from mimiciii.adult_cv_notes', con=conn)

# map values from key/admissions
note_df['true_year'] = note_df['hadm_id'].map(key_dict)
note_df['admittime'] = note_df['hadm_id'].map(hadm_admit_key)
note_df['dischtime'] = note_df['hadm_id'].map(hadm_disch_key)

# check if a leap day
def leap_year_bool(row):
    if row.month == 2:
        if row.day == 29:
            return 1
        else:
            return 0
    else:
        return 0

# make columns to indicate if dates need leap year shift
note_df['admit_shift'] = note_df['admittime'].apply(lambda x: leap_year_bool(x))
```

```
note_df['disch_shift'] = note_df['dischtime'].apply(lambda x: leap_year_bool(x))
note_df['chart_shift'] = note_df['chartdate'].apply(lambda x: leap_year_bool(x))

# column to indicate any leap years in the note record
note_df['any_leap'] = note_df['admit_shift'] + note_df['disch_shift'] +␣
 ↪note_df['chart_shift']

# create dataframe for unique admissions (with all notes) aggregated, to check␣
 ↪if any leap day in record
time_shift = note_df.groupby('hadm_id', as_index=False).agg({"any_leap": "sum"})

# mark admission as having any leap days in record
time_shift['any_leap'] = time_shift['any_leap'].astype(bool).astype(int)

# making column for leapyear time change
timechange = {0: 0,
              1: 364} # 364 days is divisible by 7, will preserve day of week

time_shift['leap_shift'] = time_shift['any_leap'].map(timechange)

time_shift['leap_shift'] = [timedelta(days = i) for i in␣
 ↪time_shift['leap_shift']]

# map the leapyear time change to df
leap_dict = dict(zip(list(time_shift['hadm_id']),␣
 ↪list(time_shift['leap_shift'])))

note_df['leap_shift'] = note_df['hadm_id'].map(leap_dict)

# applying the leap year shift to just admission date (apply this shift to all␣
 ↪admission dates later)
note_df['new_admittime'] = note_df['admittime'] - note_df['leap_shift']
```

1.1.8 1.1.7 Apply the true times and time shifts to all dates

[ ]:
```
# make column for true year based on original years key
note_df['true_year'] = note_df['hadm_id'].map(key_dict)

# applying the true years to the (leap year corrected) admission dates
admittime_list = list(note_df['new_admittime'])
true_years_list =list(note_df['true_year'])
admittime_true_list = []

for i, k in zip(admittime_list, true_years_list):
    try:
        j = i.replace(year = k)
```

```
    except:
        j ='Error'
    admittime_true_list.append(j)

# making true admittime
note_df['true_admittime'] = admittime_true_list

# calculating change between leap shifted admittime and true admittime
note_df['master_shift'] = note_df['new_admittime'] - note_df['true_admittime']

# apply the leap year shift and master shift to all other times
note_df['true_dischtime'] = note_df['dischtime'] - note_df['leap_shift'] -␣
 ↪note_df['master_shift']
note_df['true_chartdate'] = note_df['chartdate'] - note_df['leap_shift'] -␣
 ↪note_df['master_shift']
note_df['true_charttime'] = note_df['charttime'] - note_df['leap_shift'] -␣
 ↪note_df['master_shift']
note_df['true_storetime'] = note_df['storetime'] - note_df['leap_shift'] -␣
 ↪note_df['master_shift']

# drop unnecessary columns
note_df.drop(columns = ['admit_shift', 'disch_shift', 'chart_shift', 'any_leap',
                  'new_admittime'], inplace = True)
```

1.1.9 1.1.8 Double Checking the Time Shifts

[ ]:
```
# confirming the time deltas match
def check_time(row):
    fake_admit_to_chart = note_df.loc[row, 'chartdate'] - note_df.loc[row,␣
 ↪'admittime']
    fake_chart_to_disch = note_df.loc[row, 'dischtime'] - note_df.loc[row,␣
 ↪'chartdate']
    fake_admit_to_disch = note_df.loc[row, 'dischtime'] - note_df.loc[row,␣
 ↪'admittime']

    real_admit_to_chart = note_df.loc[row, 'true_chartdate'] - note_df.loc[row,␣
 ↪'true_admittime']
    real_chart_to_disch = note_df.loc[row, 'true_dischtime'] - note_df.loc[row,␣
 ↪'true_chartdate']
    real_admit_to_disch = note_df.loc[row, 'true_dischtime'] - note_df.loc[row,␣
 ↪'true_admittime']

    if fake_admit_to_chart != real_admit_to_chart:
        result = 'Error'
    elif fake_chart_to_disch != real_chart_to_disch:
        result = 'Error'
```

```
    elif fake_admit_to_disch != real_admit_to_disch:
        result = 'Error'
    else:
        result = 'Match'

    return result


# apply function to dataframe
note_df['check_time'] = [check_time(x) for x in note_df.index]

# print the results - should show all 'Match'
note_df['check_time'].value_counts()
```

1.1.10 1.1.9 Adding a table for real dates

[ ]:
```
# adding time shifts to table
df_ids_times = note_df.loc[:,['hadm_id','true_year', 'true_admittime',
                        'true_dischtime']]
```

[ ]:
```
cur.execute("""
DROP TABLE IF EXISTS mimiciii.time_study_id_date;

CREATE TABLE mimiciii.time_study_id_date
(hadm_id int,
true_year int,
true_admittime timestamp,
true_dischtime timestamp);""")

conn.commit()
```

[ ]:
```
df_ids_times.to_sql('time_study_id_date', con=engine, if_exists='append',␣
↪chunksize=1, index=False, schema='mimiciii')
```

[ ]:
```
cur.execute("""
DROP TABLE IF EXISTS mimiciii.time_study_notes;

CREATE TABLE mimiciii.time_study_notes
(hadm_id int,
chartdate timestamp,
charttime timestamp,
storetime timestamp,
text varchar,
true_year int,
true_admittime timestamp,
true_dischtime timestamp,
true_chartdate timestamp,
```

```
true_charttime timestamp,
true_storetime timestamp);""")

conn.commit()
```

[ ]:
```
note_df_save = note_df.loc[:,['hadm_id', 'chartdate', 'charttime', 'storetime',␣
↪'text', 'true_year',
                    'true_admittime', 'true_dischtime', 'true_chartdate',␣
↪'true_charttime', 'true_storetime']]
```

[ ]:
```
note_df_save.to_sql('time_study_notes', con=engine, if_exists='append',␣
↪chunksize=1, index=False, schema='mimiciii')
```

1.1.11 Clean Up, Commit, and Close

[ ]:
```
conn.commit();
cur.close();
conn.close();
```