# 1.2.2_create_groups_concat_notes

March 11, 2021

## 1 1.2.x_create_groups_concat_notes

- python 2.7.x
- from the mimic iii PostgreSQL database
- label admissions as **transfused** or **control** based on the ICD9 codes and inputs (labeled with our custom dictionary).
- all the notes for each admission (hadm_id) get ordered by time and concatenated into one note per admission.
-

### 1.1 create tables for transfused `transfused_notes_sink` and control `ctrl_notes_sink`

- corrected issues with infants (3/19/19). by inner join the icd9 tables with patients_adult
- added ability to add more data to the top of each note (charttime, providerID, note type) (5/16/19)

### 1.2 1. import libraries, connect to mimic database

[15]:
```
conn.commit();
cur.close();
conn.close();
```

[1]:
```
import sys

import time
from datetime import datetime
import datetime

import pandas as pd
import random

from tqdm import tnrange, tqdm_notebook
from time import sleep

from importlib_metadata import version

# things to connect to the posgres database
import psycopg2
from sqlalchemy import create_engine, update, event
POSTGRES_CONNECT = os.environ.get("POSTGRES_CONNECT")
POSTGRES_ENGINE = os.environ.get("POSTGRES_ENGINE")
conn = psycopg2.connect(POSTGRES_CONNECT)
cur = conn.cursor();
cur.execute("""SET search_path = mimiciii;""")
engine = create_engine(POSTGRES_ENGINE)

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))
```

```
last ran:  2019-12-24 23:53:36.145380
Python Version: 3.7.3 (
operating system: darwin
pandas version: 0.24.2
sqlalchemy version: 1.3.3
psycopg2 version: 2.7.6.1
tqdm version: 4.32.1
```

## 1.3 2. Create ICD-9 groups of admissions

- use the identified ICD-9 codes from RB (7/20/18)
- create transfue group
- create grey group
- create control group (everything that's not transfuse or grey)

### 1.3.1 2.1 create tranfusion group from table that lists procedures by admission

- pull out rows from `procedures_icd` that + have one of these icd9 codes [9901, 9903,9904, 9905, 9907] + exist in the `patients_adult` table
- admissions (hadm_id) = 7514

[4]:
```
cur.execute("""
DROP TABLE IF EXISTS mimiciii.transfusion_icd9;

SELECT p.*, i.icd9_code, i.hadm_id
    INTO transfusion_icd9
FROM mimiciii.procedures_icd i
INNER JOIN mimiciii.patients_adult p
```

```
    ON i.subject_id=p.subject_id
    WHERE i.icd9_code IN ('9901','9903','9904','9905','9907');""")

# get counts of transfusion admissions using only icd9 selection criteria.
cur.execute("""
SELECT COUNT(DISTINCT hadm_id) AS transf_admissions_count,
COUNT(distinct icd9_code) AS code_count
FROM mimiciii.transfusion_icd9
;""")

print(pd.DataFrame(cur.fetchall(), columns=[␣
↪'transf_admissions_count','code_count']).to_string(index=False))
```

```
transf_admissions_count  code_count
                   7514           5
```

### 1.3.2 2.2 create grey group from table that lists procedures (icd-9 codes) by admission

- pull out rows from `procedures_icd` that + have one of these icd9 codes [9900, 9902] + exist in the `patients_adult` table
- unique admissions (hadm_id) = 64

[5]:
```
cur.execute("""
DROP TABLE IF EXISTS mimiciii.grey_icd9;

SELECT p.*, i.icd9_code, i.hadm_id
    INTO mimiciii.grey_icd9
FROM mimiciii.procedures_icd i
INNER JOIN mimiciii.patients_adult p
    ON i.subject_id=p.subject_id
    WHERE i.icd9_code IN ('9900','9902');""")

# get counts of grey admissions using only icd9 selection criteria.
cur.execute("""
SELECT COUNT(DISTINCT hadm_id) AS grey_admissions_count,
    COUNT(DISTINCT icd9_code) AS code_count
FROM mimiciii.grey_icd9;""")

print(pd.DataFrame(cur.fetchall(), columns=[␣
↪'grey_admissions_count','code_count']).to_string(index=False))
```

```
grey_admissions_count  code_count
                   64           2
```

### 1.3.3 2.3 create control icd9 group `ctrl_icd9` from table that lists procedures (icd-9 codes) by admission

- keep all admissions that are not in the `transfusion_icd9` or the `grey_icd9` tables

- are in the `patients_adult` table
- this way we end up with only admissions that have never been assigned one of our transfusion or grey icd9 procedure codes
- the 'IS NOT TRUE' is there because of Null values, otherwise we would use 'NOT IN'
- unique admissions = 34269

[6]:
```
cur.execute("""
DROP TABLE IF EXISTS mimiciii.ctrl_icd9;

SELECT p.*, c.icd9_code, c.hadm_id
    INTO mimiciii.ctrl_icd9
FROM mimiciii.procedures_icd c

INNER JOIN mimiciii.patients_adult p
    ON c.subject_id=p.subject_id

    WHERE (c.subject_id IN (
            SELECT  x.subject_id
            FROM mimiciii.grey_icd9 x))
        IS NOT TRUE

    AND (c.subject_id IN (
            SELECT  t.subject_id
            FROM mimiciii.transfusion_icd9 t))
        IS NOT TRUE
    ;""")

# get counts of ctrl admissions using only icd9 selection criteria.
cur.execute("""
SELECT COUNT(DISTINCT hadm_id),
COUNT(DISTINCT icd9_code) AS code_count
FROM mimiciii.ctrl_icd9;""")

print(pd.DataFrame(cur.fetchall(), columns=[␣
 ↪'ctrl_admissions_count','code_count']).to_string(index=False))
```

```
ctrl_admissions_count  code_count
                34269        1871
```

## 1.4 3. Label each input event as transfuse, grey, or control

- load in the D_items identified as transufe group and Grey group from xlsx sheet (RB 7/19/18)
- use all adult input events to find

1. transfuse inputs (T) = inputs ever been assigned a transfuse label
2. grey inputs (G) = inputs that have been assigned a grey label
3. control inputs (N) = inputs that have only been assigned labels that are NOT transfue or grey

### 1.4.1 3.1 Load in the labeled D_items from xlsx sheet

[21]:
```
# create new empty table in mimiciii schema

cur.execute("""
DROP TABLE IF EXISTS mimiciii.transfusion_items_dict;

CREATE TABLE mimiciii.transfusion_items_dict
(Notes varchar,
 GRP char(1),
 ROW_ID int,
 ITEMID int,
 LABEL varchar,
 ABBREVIATION varchar,
 DBSOURCE varchar,
 LINKSTO varchar,
 CATEGORY varchar,
 UNITNAME varchar,
 PARAM_TYPE varchar,
 CONCEPTID varchar,
 ref varchar);""")
conn.commit()
```

[ ]:
```
#Run in postgres command line
COPY mimiciii.transfusion_items_dict
FROM 'D:\\20180717D_ITEMS_related_to_blood_full.csv'
DELIMITER ',' CSV HEADER;
```

Verify that the table has been created correctly. + should have 132 rows total + T=54 + G=40 + N=38

[7]:
```
cur.execute("""
SELECT grp, count(*)
    FROM mimiciii.transfusion_items_dict
GROUP BY grp;""")

colnames = [desc[0] for desc in cur.description]
print(pd.DataFrame(cur.fetchall(), columns=colnames).to_string(index=False))
```

```
grp  count
  N     38
  T     54
  G     40
```

### 1.4.2 3.2 join labeled items with d_items

- everything that isn't in the new dict, gets a grp label of 'N'

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

SELECT i.*, d.notes
    ,CASE WHEN grp IS NULL THEN 'N' ELSE grp END

    INTO mimiciii.D_items_labeled
FROM mimiciii.transfusion_items_dict d

RIGHT JOIN mimiciii.D_items i
    ON i.itemid=d.itemid
;""")

conn.commit()
```

### 1.4.3 3.3 Join new D_items with inputs_all to give each input a grp label

```
[9]: cur.execute("""DROP TABLE IF EXISTS mimiciii.inputs_all_labeled;""")

cur.execute("""
SELECT d.label,d.grp, i.*
    INTO mimiciii.inputs_all_labeled
FROM mimiciii.D_items_labeled d
    RIGHT JOIN mimiciii.inputs_all i
    ON i.itemid=d.itemid
;""")
conn.commit()
```

**Print the number of inputs (non-lab charted items) for each of the groups.**

- N = 289,352,348
- T = 153,154
- G = 3872

```
[10]: cur.execute("""
SELECT grp, count(*)
    FROM mimiciii.inputs_all_labeled
GROUP BY grp;""")

colnames = [desc[0] for desc in cur.description]
print(pd.DataFrame(cur.fetchall(), columns=colnames).to_string(index=False))
```

```
grp      count
  G       3872
  N  289352348
  T     153154
```

## 1.5 4. Create the full list of admissions in transfused group `transfused_hadm_id`

- Create a table (`transfused_hadm_id`) of transfuse group admission ids (**hadm_id**) from the icd9 (`transfusion_icd9`) and the non-lab chart events (`inputs_all_labeled` **grp=T**) criteria
- Transfusion admissions_count = 21541

[11]:
```
cur.execute(""" DROP TABLE IF EXISTS MIMICIII.transfused_hadm_id;

SELECT DISTINCT hadm_id
    INTO mimiciii.transfused_hadm_id

FROM mimiciii.inputs_all_labeled c

    WHERE grp='T'
    AND c.hadm_id IS NOT NULL
    UNION

SELECT DISTINCT hadm_id
FROM mimiciii.transfusion_icd9
    WHERE hadm_id IS NOT NULL
;""")
```

[12]:
```
cur.execute("""
SELECT count(DISTINCT hadm_id)
FROM mimiciii.transfused_hadm_id
;""")
print(pd.DataFrame(cur.fetchall(), columns=[ 'transfusion admissions_count']).
 ↪to_string(index=False))
```

```
transfusion admissions_count
                       21541
```

## 1.6 5. Create the full list of admissions in grey group `grey_hadm_id`

- Create a table (`grey_hadm_id`) of grey group admission ids (**hadm_id**) from the icd9 (`grey_icd9`) and the non-lab chart events (`inputs_all_labeled` **grp = G**) criteria
- grey admissions_count = 2373

[13]:
```
cur.execute(""" DROP TABLE IF EXISTS MIMICIII.grey_hadm_id;

SELECT DISTINCT hadm_id
    INTO mimiciii.grey_hadm_id
FROM mimiciii.inputs_all_labeled c

    WHERE grp='G'
    AND hadm_id IS NOT NULL
    UNION
SELECT DISTINCT hadm_id
FROM mimiciii.grey_icd9
    WHERE hadm_id IS NOT NULL
;""")
```

[14]:
```
cur.execute("""
SELECT count(DISTINCT hadm_id)
FROM mimiciii.grey_hadm_id
;""")
print(pd.DataFrame(cur.fetchall(), columns=[ 'grey admissions_count']).
 ↪to_string(index=False))
```

```
grey admissions_count
                 2373
```

## 1.7 6. Create list of ctrl admissions `ctrl_ids`

- make a list of hadm_ids admissions not in the transfused or grey groups.

### 1.7.1 6.1 join all admissions from `ctrl_icd9` and `inputs_all_labeled` = N

- this basically pulls every admission **50,328**

[15]:
```
cur.execute(""" DROP TABLE IF EXISTS mimiciii.ctrl_idsa;

SELECT DISTINCT i.hadm_id
    INTO mimiciii.ctrl_idsa
FROM mimiciii.inputs_all_labeled i

    WHERE i.grp='N'
    AND i.hadm_id IS NOT NULL
    UNION

SELECT DISTINCT y.hadm_id
FROM mimiciii.ctrl_icd9 y
    WHERE y.hadm_id IS NOT NULL
;""")
```

[16]:
```
cur.execute(""" SELECT COUNT(DISTINCT hadm_id)
FROM  mimiciii.ctrl_idsa;""")

ncount=cur.fetchall()
print( pd.DataFrame(ncount, columns=[ 'admissions']).to_string(index=False))
```

```
admissions
     50328
```

### 1.7.2 6.2 remove admissions that belong to the `transfused_hadm_id` table or the `grey_hadm_id`

- admissions = 28,128

[17]:
```
cur.execute(""" DROP TABLE IF EXISTS mimiciii.ctrl_ids;

SELECT DISTINCT i.hadm_id
    INTO mimiciii.ctrl_ids
FROM mimiciii.ctrl_idsa i

    WHERE  i.hadm_id NOT IN (
           SELECT  x.hadm_id
           FROM mimiciii.transfused_hadm_id x)

    AND i.hadm_id NOT IN (
            SELECT g.hadm_id
            FROM mimiciii.grey_hadm_id g)
;""")
```

[18]:
```
cur.execute(""" SELECT COUNT(DISTINCT hadm_id)
FROM  mimiciii.ctrl_ids;""")

ncount=cur.fetchall()
print( pd.DataFrame(ncount, columns=[ 'admissions']).to_string(index=False))
```

```
admissions
     28128
```

## 1.8 7. Get Transfused Notes `xf_notes`

### 1.8.1 7.1 Get all notes for admissions (hadm_id) that have been identified as transfused group `transfused_hadm_id`

- Print the total number of notes, and unique admissions **21,443**
- keep all types of timestamps **chartdate** is only a date but is present in every note

```
**charttime** and **storetime** are time and date, but are not always present (discharge
```

- also do not use any notes where the provider has indicated that the note is an error (**iserror**=1)
- note that there are 98 less admissions than in `transfused_hadm_id` (step 4), meaning that 98 admissions did not have any data in the `noteevents` table

[19]:
```
cur.execute("""
DROP TABLE IF EXISTS mimiciii.transfused_notes;

SELECT  B.*
INTO mimiciii.transfused_notes
```

```
    FROM mimiciii.noteevents B
    WHERE B.hadm_id IN (
            SELECT  x.hadm_id
            FROM mimiciii.transfused_hadm_id x)

    AND B.iserror IS NULL
;""")
```

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

```
 total notes count  admissions
            874711       21443
```

## 1.9 7.2 One Document per Admission

For each admission, concatenate all the notes for that admission into one note (thus, each admission has one **document**). Create a table of these admission notes using the hospital admission id (hadm_id) as the identifier rather than the note id (row_id)

### 1.9.1 Transfusion Notes by Admission `transfused_notes_sink` with or without metadata

- group by admission ID
- order by note date (‘note_dt’)
- concatenate all notes for that admission ID into one string
- metadata==True: concatenate all notes and other data (date(s), provider=cgid, note, type=category,description) for that admission ID into one string
- save as transfused_notes_sink or transfused_notes_sink_metadata

### 1.9.2 7.2.1 Create new table for results `transfused_notes_sink` or `transfused_notes_sink_metadata`

- hadm_id
- text (concatenate notes and/or other data)

```
[3]: # set whether you want to include metadata at the top of each note (we don't␣
      ↪use this for the NLP, but is' useful for the viewing by SMEs)
     metadata = False

     if metadata==False:

         cur.execute("""DROP TABLE IF EXISTS mimiciii.transfused_notes_sink;

         CREATE TABLE mimiciii.transfused_notes_sink
```

```
    (hadm_id int,
     text varchar);""")

else:
    cur.execute("""DROP TABLE IF EXISTS mimiciii.transfused_notes_sink_metadata;

    CREATE TABLE mimiciii.transfused_notes_sink_metadata
    (hadm_id int,
     text varchar);""")

conn.commit();
```

### 1.9.3 7.2.2 create list of unique hadm_ids

[4]:
```
xf = pd.read_sql("""
SELECT hadm_id
FROM mimiciii.transfused_notes """, engine)

# get list of ids
xf_ids = xf.hadm_id.unique()
len(xf_ids)
```

[4]:
```
21443
```

### 1.9.4 7.2.3 function that lets us make multiple requests to the postgres using pandas read_sql

[5]:
```
@event.listens_for(engine, 'before_cursor_execute')
def receive_before_cursor_execute(conn, cursor, statement, params, context,␣
 ↪executemany):
    #print("FUNC call")
    if executemany:
        cursor.fast_executemany = True
```

### 1.9.5 7.2.4 function to pull notes, concatenate and save

- this will take a few hours(2.7) to run
- iterate through for each unique admission (hadm_id)
- pull all notes for an admission
- order notes by charttime , then storetime
- concatenate
- save as one big note to new table

[6]:
```
s = time.time()

for j in tqdm_notebook(xf_ids):
    if metadata == False:

        table_name = 'transfused_notes_sink'

        sql = """

        SELECT  hadm_id, chartdate, charttime, storetime, text
        FROM mimiciii.transfused_notes
            WHERE hadm_id in ({0})
        GROUP BY hadm_id, chartdate, charttime, storetime, text
        ORDER BY chartdate, charttime, storetime"""

        # run sql query above to pull all notes for one admission (in order by␣
↪date)
        sql = sql.format(j)
        xnotes=pd.read_sql(sql, engine)

        xnotes = xnotes.loc[:,'text']

    else:

        table_name = 'transfused_notes_sink_metadata'

        sql = """

        SELECT subject_id, hadm_id, chartdate, charttime, storetime, category,␣
↪cgid, description, text
        FROM mimiciii.transfused_notes
            WHERE hadm_id in ({0})
        GROUP BY subject_id, hadm_id, chartdate, charttime, storetime,␣
↪category, cgid, description, text
        ORDER BY chartdate, charttime, storetime"""

        # run sql query above to pull all notes for one admission (in order by␣
↪date)
        # concatenate notes and all other cols (metadata)
        # all the metadata gets put into one token for duplicate removal␣
↪purposes

        sql = sql.format(j)
        xnotes=pd.read_sql(sql, engine)

        xnotes.loc[:,'text2'] = xnotes.loc[:,'text']
        xnotes.iloc[:,-2] = '. '
    # put a a period + whitespace to designate the end start and end of a note ␣
 ↪
    xnotes['separator'] = '. '

    xtext = xnotes.to_csv(None, header=False, index=False)

    # save as a new dataframe
    xtext2 = [(j, xtext)]
    xfulltext=pd.DataFrame(xtext2, columns=['hadm_id', 'text'])

    # append user and single note to the new table in database

    xfulltext.to_sql(table_name, con=engine, if_exists='append', chunksize=1,␣
 ↪index=False, schema='mimiciii')

print(time.time() - s)

conn.commit()
```

```
HBox(children=(IntProgress(value=0, max=21443), HTML(value='')))


7052.120042562485
```

[ ]:
```
if metadata==False:
    cur.execute(""" SELECT COUNT(DISTINCT hadm_id) FROM transfused_notes_sink;
 ↪""")
else:
    cur.execute(""" SELECT COUNT(DISTINCT hadm_id) FROM␣
 ↪transfused_notes_sink_metadata;""")

print( pd.DataFrame(cur.fetchall()).to_string(index=False))
```

[ ]:
```
if metadata==False:
    cur.execute(""" SELECT COUNT(*) FROM transfused_notes_sink;""")
else:
    cur.execute(""" SELECT COUNT(DISTINCT hadm_id) FROM␣
 ↪transfused_notes_sink_metadata;""")

print( pd.DataFrame(cur.fetchall()).to_string(index=False))
```

## 1.10 8. Get Control Notes `ctrl_notes`

### 1.10.1 8.1 Get all notes for admissions (hadm_id) that have been identified as control group `ctrl_ids`

- Print the total number of notes, and unique admissions

- note that there are **27,888** admissions w/ notes (240 control group admissions did not have data in the `noteevents` table.

[7]:
```
cur.execute(""" DROP TABLE IF EXISTS mimiciii.ctrl_notes;
SELECT n.*
    INTO ctrl_notes
FROM noteevents n
    WHERE n.hadm_ID IN (
        SELECT DISTINCT c.hadm_id
        FROM ctrl_ids c)

    AND n.iserror IS NULL

    ;""")

conn.commit()
```

[8]:
```
cur.execute(""" SELECT COUNT(*), COUNT(DISTINCT hadm_id) FROM ctrl_notes;""")
print( pd.DataFrame(cur.fetchall(), columns=[ 'total notes count', 'ctrl␣
 ↪admissions with notes']).to_string(index=False))
```

```
 total notes count  ctrl admissions with notes
            535639                       27888
```

### 1.10.2 8.2 Control Notes by Admission `ctrl_notes_sink` or `ctrl_notes_sink_metadata`

- group by admission ID
- order by note date ('note_dt')
- concatenate all notes for that admission ID into one string
- save as ctrl_notes_sink

### 1.10.3 8.2.1 Create a new table for the results `ctrl_notes_sink` or `ctrl_notes_sink_metadata`

- hadm_id
- subject_id
- text (concatenated notes + metadata (if metadata==True)

[9]:
```
if metadata==False:
    cur.execute("""DROP TABLE IF EXISTS ctrl_notes_sink;

    CREATE TABLE mimiciii.ctrl_notes_sink
    (hadm_id int,
     text varchar);""")

else:
    cur.execute("""DROP TABLE IF EXISTS ctrl_notes_sink_metadata;

    CREATE TABLE mimiciii.ctrl_notes_sink_metadata
    (hadm_id int,
     text varchar);""")

conn.commit();
```

### 1.10.4 8.2.2 pull the unique hadm_ids (identifies each admission) and make a list of them

[10]:
```
ctrl_ids = pd.read_sql("""
SELECT hadm_id
FROM mimiciii.ctrl_notes""", engine)

cids= ctrl_ids.hadm_id.unique()
```

### 1.10.5 8.3.4 Function to pull and concatenate and save

- this takes a few hours (2.3 hrs) to run
- iterates through each hadm_id
- pulls all notes (and other data if chosen)
- orders notes in order of charttime, then storetime
- concatenate and save in new table

[11]:
```
s = time.time()

for i in tqdm_notebook(cids):

    if metadata==False:

        table_name = 'ctrl_notes_sink'

        sql = """

        SELECT  hadm_id, chartdate, charttime, storetime,text
        FROM mimiciii.ctrl_notes
            WHERE hadm_id IN ({0})
        GROUP BY  hadm_id, chartdate, charttime, storetime, text
        ORDER BY chartdate, charttime, storetime"""

        sql = sql.format(i)
        cnotes = pd.read_sql(sql, engine)
        cnotes = cnotes.loc[:,'text']

    else:

        table_name = 'ctrl_notes_sink_metadata'

        sql = """
        SELECT subject_id, hadm_id, chartdate, charttime, storetime, category,␣
↪cgid, description, text
        FROM mimiciii.ctrl_notes
            WHERE hadm_id IN ({0})
        GROUP BY subject_id, hadm_id, chartdate, charttime, storetime,␣
↪category, cgid, description, text
        ORDER BY chartdate, charttime, storetime"""

        sql = sql.format(i)
        cnotes = pd.read_sql(sql, engine)

        cnotes.loc[:,'text2'] = cnotes.loc[:,'text']
        cnotes.iloc[:,-2] = '. '


    cnotes['separator'] = '. '

    #CONCAT NOTES
    ctext = cnotes.to_csv(None, header=False, index=False)

    #put into a data frame with hadm_id
    ctext2 = [(i, ctext)]
    cfulltext = pd.DataFrame(ctext2, columns=['hadm_id', 'text'])


    # append user and single note to the new table in database

    cfulltext.to_sql(table_name, con=engine, if_exists='append', chunksize=1,␣
↪index=False, schema='mimiciii')

print('total time=',((time.time() - s)/60),'min')

conn.commit()
```

```
HBox(children=(IntProgress(value=0, max=27888), HTML(value='')))


total time= 87.5706007361412 min
```

[17]:
```
if metadata==False:
    cur.execute(""" SELECT COUNT(*), COUNT(DISTINCT hadm_id) FROM␣
↪ctrl_notes_sink;""")
else:
    cur.execute(""" SELECT COUNT(*), COUNT(DISTINCT hadm_id) FROM␣
↪ctrl_notes_sink_metadata;""")
```

```
print( pd.DataFrame(cur.fetchall(), columns=[ 'total notes count', 'ctrl␣
↪admissions with notes']).to_string(index=False))
```

```
total notes count  ctrl admissions with notes
            27888                       27888
```

## 1.11 9. Clean Up, Commit, and Close

[12]:
```
conn.commit()
cur.close()
conn.close()
```