Author: Nicholas Bradford

Problem

ChenMed

Membership Analysis

In order to take care of patients at ChenMed, we must be able to track the patients we take care of and the associated Health Plan coverage. Recently, we started to notice that there were duplicates our key tables.

- T_PATIENT: List of patients who have been seen by a ChenMed provider

- T_ELIGIBLITY: For patients in the T_PATIENT table, this table documents the Health Plan the patient is associated with for healthcare coverage. If a patient does not have an associated record in this table, this means the patient does not have current Health Plan coverage, meaning the patient should not be seen by ChenMed.

This should not happen. Here are some examples of duplicates that were found:

T_PATIENT

FIRST_NAME	LAST_NAME	ACTIVE	DATE_OF_BIRTH	ADDRESS	CITY	STATE	ZIP	GENDER
[DYLHU	PFHOOLRWW	1	03/05/99	56:4#UXWODQG#VW	RSD#ORFND	IO	66387	M
[DYLHU	PFHOOLRWW	1	03/05/99	56:4#UXWODQG#VW	RSD#ORFND	IO	66387	M

T_ELIGIBILITY

PATIENT_ID	ELIGIBILITY_ID	INSURANCE_CARRIER	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	POLICY_NUMBER	GROUP_ID
34662	6071020	4	VDPXHO	OHZLV	07/06/33	536<35;-34	333
34662	6163743	4	VDPXHO	OHZLV	07/06/33	536<35;-34	UR-DEG97

We suspect two processes that may create these duplicates:

- T_PATIENT: When a patient shows up in the clinic, if we are not able to find them in the T_PATIENT table, the front office adds a new record into the T_PATIENT table.

- T_ELIGILIBITY: When a patient is added to the T_PATIENT table, coverage is confirmed, then a record is added to the T_ELGIIBILITY table using the new PATIENT_ID. This is done through an automated process. Additionally, each month, we get a membership file from each health plan. These files will include patients who we have seen and patients who have ***never*** seen. When we get these files monthly, we have a process to check the patients listed in the membership file against T_PATIENT and T_ELIGILIBITY to avoid creating duplicate records. Below are the criteria to determine a match:
  1. Policy Number
  2. First Name + Last Name +DOB
  3. DOB + Phone +Gender
  4. Last Name(first 3 letter) + First Name(first 2 letter) + DOB
  5. Last Name(first 3 letter) + First Name(first 2 letter) + Phone + Gender

As a result, we want you to review the data in the two tables above to clean up the duplicates and develop a process to ensure that new duplicates are not created.

For the interview, we would like you analyze the data to do the following:

- Provide a count of the duplicates and show some specific example of the duplicates that were found
  - Provide the criteria you used to determine which records were considered duplicates
- Provide a set of clean tables (T_PATIENT & T_ELIGIBILITY) where the duplicates have been removed
- Provide a recommendation for how we can avoid duplicates in the future
  - What criteria should we use to identify duplicates?

NOTE:

For the purposes of this exercise, many of the fields have been encrypted. The data in the fields still represent values that can be queried for duplicates. The same values in a column across 2 records can be considered a duplicate.

Also, social security number cannot be used as a unique identifier because the Health Plans do not provide it to us. So we do not store this value and have to rely on other demographic information to determine if a record is unique.

In [7]:
```python
import pandas as pd
import numpy as np
import pandas_dedupe #dedup propsal
```

In [8]:
```python
t_patient = pd.read_csv('/Users/nicholasbradford/Desktop/InterviewPrep/ChenMed/T_PATIENT.csv')
t_elig = pd.read_csv('/Users/nicholasbradford/Desktop/InterviewPrep/ChenMed/T_ELIGIBILITY.csv')
```

In [9]:
```python
t_elig.sort_values('PATIENT_ID').head()
```

Out[9]:

	ELIGIBILITY_ID	PATIENT_ID	INSURANCE_CARRIER	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	ADDRESS	CITY	STATE	
10806	6168218	20614	1	IUDQN#	PRVV	13-Aug-26	4:98#QZ#488WK#VW	RSD#ORFND	IO	66387
10081	6167492	20677	1	XGHOO#	PLOOV	16-Jun-41	4<<35#QZ#;9WK#DYH	KLDOHDK	IO	66348
12993	6170417	20700	1	DOYDOLD#	HOOLV	02-Dec-26	7876#VZ#4<8WK#ZD\	PLUDPDU	IO	6635<
12064	6169483	20706	1	GHERUD#	ZHVW	04-Feb-60	48544#QZ#65QG#SO	PLDPL#JDUGHQV	IO	66387
10574	6167986	20728	1	GHOUR\#	ILVKHU	29-Dec-26	4:444#QZ#<WK#SO	PLDPL	IO	6649<

In [10]:
```python
t_patient.sort_values('PATIENT_ID').head()
```

Out[10]:

	PATIENT_ID	OFFICE_ID	ACTIVE	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	ADDRESS	CITY	STATE	ZIP	GENDER
0	20592	20755	1	Fkulvwlqh	P|huv	01-Aug-43	939#VZ#:9wk#Dyh	Qruwk#Odxghugdoh	IO	6639;	F
1	20598	20754	1	Fduo	Vhqvderxjk	03-Jan-28	;;:4#Q1z1#9wk#Vw1	Shpeurnh#Slqhv	IO	66357	M
2	20601	20754	1	Gdylg	Vwdoozruwk	14-Apr-26	Pdloer{#884;:8	Rsd#Orfnd/	IO	663880379<	M
3	20614	20754	1	IUDQN#fdoo#diwhu#43	PRVV	13-Aug-26	4:98#QZ#488wk#Vw	Pldpl#Jdughqv	IO	66387	M
4	20615	20916	1	Mlpp|	Nrr	18-Apr-40	96:8#Vz#73vw	Pldpl	IO	66488	M

In [19]:
```python
print('t_elig unique patient ID # {}'.format(t_elig.PATIENT_ID.nunique())) #Check if patient ID is a unique identifier
print('t_patient unique patient ID # {}'.format(t_patient.PATIENT_ID.nunique()))
```

```
t_elig unique patient ID # 65273
t_patient unique patient ID # 65534
```

In [21]:
```python
len(t_patient[t_patient['PATIENT_ID'].isin(t_elig['PATIENT_ID'])].sort_values('PATIENT_ID'))
```

Out[21]: 6821

Note: We can see there are only 6489 matches between the 2 tables thus we should focus on patients that can be seen by ChenMed.

In [22]:
```python
t_elig[t_elig['PATIENT_ID'].isin(t_patient['PATIENT_ID'])].sort_values('PATIENT_ID')
```

Out[22]:

	ELIGIBILITY_ID	PATIENT_ID	INSURANCE_CARRIER	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	ADDRESS	CITY	STATE
10806	6168218	20614	1	IUDQN#	PRVV	13-Aug-26	4:98#QZ#488WK#VW	RSD#ORFND	IO
10081	6167492	20677	1	XGHOO#	PLOOV	16-Jun-41	4<<35#QZ#;9WK#DYH	KLDOHDK	IO
12993	6170417	20700	1	DOYDOLD#	HOOLV	02-Dec-26	7876#VZ#4<8WK#ZD\	PLUDPDU	IO
12064	6169483	20706	1	GHERUD#	ZHVW	04-Feb-60	48544#QZ#65QG#SO	PLDPL#JDUGHQV	IO
10574	6167986	20728	1	GHOUR\#	ILVKHU	29-Dec-26	4:444#QZ#<WK#SO	PLDPL	IO
...	...	...	...	...	...	...	...	...	...
28673	6196973	151077	1	UHJLQDOG	ULFH	20-Oct-46	4577#V#7WK#VW#DSW#66<	ORXLVYLOOH	N\
32829	6201760	151079	1	MLP#	FRYHUVRQ	26-Aug-29	7;3<#KXQWHU#WUFH	SRZGHU#VSULQJ	JD
33137	6202071	151082	1	MDPHV#	WKUHHW	03-Mar-44	5:44#KLJK#WLGH#GU	DWODQWD	JD
32877	6201810	151084	1	VKHULVH#	DGDPV	14-Oct-57	DSW#63D	MRQHVERUR	JD
32538	6201465	151089	1	MHVWLQH	SULFH	07-Feb-30	58;;#FRKDVVHWW#FW	GHFDWXU	JD

6839 rows × 15 columns

Find suplicates in t_elig:

I chose this table as its the only one that has the five criteria to spot duplicates as it was recommended on the statement:

```
1.  Policy Number
2.  First Name + Last Name +DOB
3.  DOB + Phone +Gender
4.  Last Name(first 3 letter) + First Name(first 2 letter) + DOB
5.  Last Name(first 3 letter) + First Name(first 2 letter) + Phone + Gender
```

In [23]:
```python
def find_duplicates_t_elig(t_elig):
    #build dummy columns with the matching criterias
    t_elig['ctwo'] = t_elig['FIRST_NAME'] + t_elig['LAST_NAME']+ t_elig['DATE_OF_BIRTH'] #Dummy column with criteria 2
    t_elig['cthree'] = t_elig['DATE_OF_BIRTH'] + t_elig['PHONE'] + t_elig['GENDER'] #Dummy column with criteria 3
    t_elig['cfour'] = t_elig['FIRST_NAME'].astype(str).str[:2] + t_elig['LAST_NAME'].astype(str).str[:1]+ t_elig['DATE
_OF_BIRTH']#Dummy column with criteria 4
    t_elig['cfive'] = t_elig['FIRST_NAME'].astype(str).str[:2] + t_elig['LAST_NAME'].astype(str).str[:1]+ t_elig['PHON
E']+ t_elig['GENDER']#Dummy column with criteria 5


    clean_t_elig = t_elig.sort_values('POLICY_NUMBER').loc[t_elig['POLICY_NUMBER'].duplicated(keep= False)]

    #find duplicates reccords matching each criteria
    criteriaone = t_elig.sort_values('POLICY_NUMBER').loc[t_elig['POLICY_NUMBER'].duplicated(keep= False)]
    criteriatwo = t_elig.sort_values('ctwo').loc[t_elig['ctwo'].duplicated(keep= False)]
    criteriathree = t_elig.sort_values('cthree').loc[t_elig['cthree'].duplicated(keep= False)]
    criteriafour = t_elig.sort_values('cfour').loc[t_elig['cfour'].duplicated(keep= False)]
    criteriafive = t_elig.sort_values('cfive').loc[t_elig['cfive'].duplicated(keep= False)]

    clean_table = pd.concat([t_elig, criteriaone,criteriatwo, criteriathree, criteriafour, criteriafive]).drop_duplica
tes(keep=False)

    print('Total reccords before cleanup--> {}'.format(len(t_elig)))
    print('Criteria one # dupliactes--> {}'.format(len(criteriaone)))
    print('Criteria two # dupliactes--> {}'.format(len(criteriatwo)))
    print('Criteria three # dupliactes--> {}'.format(len(criteriathree)))
    print('Criteria four # dupliactes--> {}'.format(len(criteriafour)))
    print('Criteria five # dupliactes--> {}'.format(len(criteriafive)))
    print('Total reccords clean table --> {}'.format(len(clean_table)))


    return clean_table

t_elig_clean = find_duplicates_t_elig(t_elig)
```

```
Total reccords before cleanup--> 65534
Criteria one # dupliactes--> 482
Criteria two # dupliactes--> 94
Criteria three # dupliactes--> 9895
Criteria four # dupliactes--> 562
Criteria five # dupliactes--> 9632
Total reccords clean table --> 55054
```

As we can see about **10k** reccords meet the duplicate criteria for this table.

In [25]:
```python
clean_tpatient = t_patient[t_patient['PATIENT_ID'].isin(t_elig_clean['PATIENT_ID'])].sort_values('PATIENT_ID')
len(clean_tpatient)
```

Out[25]: 6489

In [ ]:
```python
#use clean t_elig to find elegible patients on the t-patient table
t_elig_clean.to_csv('T_ELIGIBILITY_CLEAN.csv')
clean_tpatient.to_csv('T_PATIENT_CLEAN.csv')
```

Summary

As the only interest is to find eligible patients the clean t_patients table has only eligible patients that don't have duplicates. The way I identified the duplicates was through a concatenation of the columns that met the criteria for the duplicate and as the code identified new duplicates it will remove them from the data frame.

Recommendations

As we only care about patients that are eligible so we can use the T-eligibility table to serve as the master for t-patient. Thus finding and maintaining the data integrity of the t-eligibility will be easier as it contains all of the fields that define the duplicate criteria.

As we only care about patients that are eligible so we can use the T-eligibility table to serve as the master for t-patient. Thus finding and maintaining the data integrity of the t-eligibility will be easier as it contains all of the fields that define the duplicate criteria.

On the other hand, if it is not possible to turn the process upside down we might be able to identify duplicates to be able to use record linkage (through clustering) that might provide a certain confidence interval duplicate values within each of the data frames. These techniques offer the following:

```
- Ability to define the types of matches for each column based on the column data types
- Use “blocks” to limit the pool of potential matches
- Provides ranking of the matches using a scoring algorithm
- Multiple algorithms for measuring string similarity
- Supervised and unsupervised learning approaches
- Multiple data cleaning methods
```

Python Record Linkage Toolkit - Sample

This package works with clustering algorithms and gives a confidence interval on whether it thinks its a duplicate or not. If a record has multiple clusters ID's it will flag it as a duplicate. The technique used here is called fuzzy deduplication which looks for string similarity.

In [26]:
```python
t_eligdedup = pandas_dedupe.dedupe_dataframe(t_elig,['POLICY_NUMBER',
                                                      'FIRST_NAME',
                                                      'LAST_NAME',
                                                      'DATE_OF_BIRTH',
                                                      'GENDER',
                                                      'PHONE'])
```

```
Importing data ...
Reading from dedupe_dataframe_learned_settings
Clustering...
# duplicate sets 65142
```

In [27]:
```python
len(t_eligdedup)
```

Out[27]: 65534

In [ ]:
```python
#t_eligdedup.to_csv('t_eligdedup_test.csv')
```

In [29]:
```python
duplitelig_df = t_eligdedup.sort_values(['confidence','cluster id']).loc[t_eligdedup['cluster id'].duplicated(keep= Fa
lse)]

duplitelig_df
```

Out[29]:

	ELIGIBILITY_ID	PATIENT_ID	INSURANCE_CARRIER	FIRST_NAME	LAST_NAME	DATE_OF_BIRTH	ADDRESS	CITY	STATE	ZIP	...
21311	6186282	141904	1	ehuqhww	jodvshu	11-apr-47	:93pruhovw	qhzruohdqv	od	:345665:	...
21312	6186283	141905	1	hxjhqh	jodvshu	15-mar-44	:93pruhovw	qhzruohdqv	od	:345665:	...
8892	6164904	63866	4	hxjhqh	krzdug	11-aug-42	6434qz4:8wkvw	pldpl	io	66389	...
8893	6164905	63870	4	oloolh	krzdug	17-mar-42	6434qz4:8wkvw	pldpl	io	66389	...
8679	6164690	1017334	4	dodq	zhlqvwrfn	17-aug-49	:53vz53wkvw	pldpl	io	66488	...
...	...	...	...	...	...	...	...	...	...	...	...
34686	6203635	1063486	1	ohrqdug	edughq	27-jan-51	4634rdnylhzugdswi5	ghfdwxu	jd	63363757	...
18932	6183871	961794	1	ehww\	urjhuv	30-mar-33	None	None	None	None	...
26909	6192403	961794	1	ehww\	urjhuv	30-mar-33	49734riruggu	kduyh\	lo	93757:8	...
18936	6183877	941682	1	gdylg	urjhuv	19-jun-49	None	None	None	None	...
22101	6187079	941682	1	gdylg	urjhuv	19-jun-49	4:7:qplurvw	qhzruohdqv	od	:3444978	...

783 rows × 21 columns

In [ ]:
```python
#t_eligdedup.to_csv('t_eligdedup_test.csv')
```