**Design Document for merging the data and renaming the columns**

**-----------Red color represents (code)
-----------white and blue are headings**

**Setup datasets and load data into an SQL server**

To minimize processing costs, we will use SQL to for initial data dimensionality reduction. This is more cost-effective since the processing will be done in storage (cheap) instead of memory (expensive!). This will also minimize process interuptions from kernel shutdowns due to insufficient resources.

**Target variables standardized data set format¶**

We need an ability to query persons that have each of the predictors in the coeffiecients file.

The dataset should have the following columns:

.person\_id - unique person\_id

.concept\_code - the code in the coefficient file

.standard\_vocabulary - the vocabulary concept code for the predictor inside of All of Us (if any) or code type of the created feature

.standard\_concept\_code - the concept code for the predictor inside of All of Us (if any) or code of the created feature

.standard\_concept\_name - the name for the predictor inside of All of Us (if any) or name of the created feature The dataset will be formed from the following dataframes:

.target\_dx\_condition\_df

.target\_dx\_observation\_df

.target\_dx\_procedure\_df

.target\_dx\_measurement\_df

.target\_px\_df

.target\_rx\_df

.demographics\_df

.ad\_history\_df

The resulting table will be called targets\_df and will be stored in a csv file for further processing

**Initialize targets\_df**

In [1]:

**import** pandas **as** pd

targets\_df **=** pd**.**DataFrame(columns**=**['person\_id',

 'concept\_code',

 'standard\_vocabulary',

 'standard\_concept\_code',

 'standard\_concept\_name'])

**target diagnoses**

**Read data**

In [2]:

demographics\_df **=** pd**.**read\_csv("./data/demographics\_df.csv")

target\_dx\_condition\_df **=** pd**.**read\_csv('./data/target\_dx\_condition\_df.csv', low\_memory**=False**)

target\_dx\_observation\_df **=** pd**.**read\_csv('./data/target\_dx\_observation\_df.csv', low\_memory**=False**)

target\_dx\_procedure\_df **=** pd**.**read\_csv('./data/target\_dx\_procedure\_df.csv', low\_memory**=False**)

target\_dx\_measurement\_df **=** pd**.**read\_csv('./data/target\_dx\_measurement\_df.csv', low\_memory**=False**)

target\_px\_df **=** pd**.**read\_csv("./data/target\_px\_df.csv", low\_memory**=False**)

all\_medications\_df **=** pd**.**read\_csv("./data/all\_medications\_grouped\_df.csv")

**import** pandas **as** pd

**Define function to structure diagnoses table**

Name: structure\_diagnoses\_table

Input parameters:

* df - dataframe to be structured

Return:

* structured dataframe

Processing:

* retain only rows where source\_vocabulary is 'ICD9CM'
* remove duplicates based on person\_id, and standard\_concept\_code
* retain only the following columns in this order: 'person\_id', 'source\_concept\_code', 'standard\_vocabulary', 'standard\_concept\_code', 'standard\_concept\_name'
* rename column source\_concept\_code to concept\_code
* convert values in concept\_code column to reflect the strata codes in the coefficient file by removing the period and adding a prefix: "dx:ICD9:"
* return the structured dataframe
* **def** structure\_diagnoses\_table(df):
* filtered\_df **=** df**.**copy()
* filtered\_df **=** filtered\_df[filtered\_df['source\_vocabulary'] **==** 'ICD9CM']
* filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'standard\_concept\_code'])
* filtered\_df **=** filtered\_df[['person\_id',
* 'source\_concept\_code',
* 'standard\_vocabulary',
* 'standard\_concept\_code',
* 'standard\_concept\_name']]
* filtered\_df **=** filtered\_df**.**rename(columns**=**{'source\_concept\_code': 'concept\_code'})
* filtered\_df['concept\_code'] **=** (
* filtered\_df['concept\_code']
* **.**astype(str) *# Convert to string*
* **.**str**.**replace('.', '') *# Remove periods*
* **.**apply(**lambda** x: f"dx\_ICD9\_{x}") *# Add prefix*
* )
*
* **return** filtered\_df

**process data**

In [5]:

dx\_datasets **=** [

 target\_dx\_condition\_df,

 target\_dx\_observation\_df,

 target\_dx\_procedure\_df,

 target\_dx\_measurement\_df,

]

*# for each diagnoses dataset, structure the table and append to targets\_df*

**from** tqdm **import** tqdm *# for the status bar (there are some impatient analysts out there)*

*# initialize the status bar*

**with** tqdm(total**=**len(dx\_datasets), desc**=**'Processing Target Diagnoses') **as** pbar:

 **for** dx\_dataset **in** dx\_datasets:

 filtered\_df **=** structure\_diagnoses\_table(dx\_dataset)

 targets\_df **=** pd**.**concat([targets\_df, filtered\_df], ignore\_index**=True**)

 pbar**.**update(1)

targets\_df

 **(procedures)**

In [13]:

**import** pandas **as** pd

target\_px\_df **=** pd**.**read\_csv("./data/target\_px\_df.csv", low\_memory**=False**)

**Define function to structure procedures table**

Name: structure\_procedures\_table

Input parameters:

* df - dataframe to be structured

Return:

* structured dataframe

Processing:

* retain only rows where source\_vocabulary is 'CPT4'
* remove rows where standard\_vocabulary is 'Visit'
* remove duplicates based on person\_id, and standard\_concept\_code
* retain only the following columns in this order: 'person\_id', 'source\_concept\_code', 'standard\_vocabulary', 'standard\_concept\_code', 'standard\_concept\_name'
* rename column source\_concept\_code to concept\_code
* convert values in concept\_code column to reflect the strata codes in the coefficient file by adding a prefix: "px:CPT4:"
* return the structured dataframe

**def** structure\_procedures\_table(df):

 filtered\_df **=** df**.**copy()

 filtered\_df **=** filtered\_df[filtered\_df['source\_vocabulary'] **==** 'CPT4']

 filtered\_df **=** filtered\_df[filtered\_df['standard\_vocabulary'] **!=** 'Visit']

 filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'standard\_concept\_code'])

 filtered\_df **=** filtered\_df[['person\_id',

 'source\_concept\_code',

 'standard\_vocabulary',

 'standard\_concept\_code',

 'standard\_concept\_name']]

 filtered\_df **=** filtered\_df**.**rename(columns**=**{'source\_concept\_code': 'concept\_code'})

 filtered\_df['concept\_code'] **=** (

 filtered\_df['concept\_code']

 **.**astype(str) *# Convert to string*

 **.**apply(**lambda** x: f"px\_CPT4\_{x}") *# Add prefix*

 )

 **return** filtered\_df

filtered\_df **=** structure\_procedures\_table(target\_px\_df)

targets\_df **=** pd**.**concat([targets\_df, filtered\_df], ignore\_index**=True**)

targets\_df

**target\_rx (medications)**

In [18]:

**import** pandas **as** pd

all\_medications\_grouped\_df **=** pd**.**read\_csv('./data/all\_medications\_grouped\_df.csv', low\_memory**=False**)

**Define function to structure all medications table**

Name: structure\_all\_medications\_table

Input parameters:

df - dataframe to be structured

Return:

structured dataframe

Processing:

remove duplicates based on person\_id, and ingredient\_list

retain only the following columns in this order: 'person\_id', 'standard\_vocabulary', 'ingredient\_list', 'ingredient\_name\_list'

rename column ingredient\_list to standard\_concept\_code

rename column ingredient\_name\_list to standard\_concept\_name

create column concept\_code with value equal to rx\_standard\_vocabulary\_standard\_concept\_code

return the structured dataframe

**def** structure\_medications\_table(df):

 filtered\_df **=** df**.**copy()

 filtered\_df **=** filtered\_df[filtered\_df['gen\_code']**.**notna()]

 filtered\_df **=** filtered\_df[['person\_id',

 'gen\_code',

 'standard\_vocabulary',

 'ingredient\_list',

 'ingredient\_name\_list']]

 filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'gen\_code'])

 filtered\_df **=** filtered\_df**.**rename(columns**=**{

 'gen\_code' : 'concept\_code',

 'ingredient\_list' : 'standard\_concept\_code',

 'ingredient\_name\_list' : 'standard\_concept\_name'

 })

 filtered\_df['concept\_code'] **=** (

 filtered\_df['concept\_code']

 **.**astype(str) *# Convert to string*

 **.**apply(**lambda** x: f"rx\_GEN\_{x}") *# Add prefix*

 )

 **return** filtered\_df

**process data**

In [20]:

filtered\_df **=** structure\_medications\_table(all\_medications\_grouped\_df)

targets\_df **=** pd**.**concat([targets\_df, filtered\_df], ignore\_index**=True**)

targets\_df

**demographics\_df(female at birth)**

In [23]:

*#read data*

**import** pandas **as** pd

demographics\_df **=** pd**.**read\_csv("./data/demographics\_df.csv")

**Define function to structure females table**

Name: structure\_females\_table

Input parameters:

* df - dataframe to be structured

Return:

* structured dataframe

Processing:

* retain only rows where sex\_at\_birth is 'Female'
* remove duplicates based on person\_id, and sex\_at\_birth
* retain only the following columns in this order: 'person\_id', 'sex\_at\_birth'
* rename column sex\_at\_birth to concept\_code
* initialize standard\_vocabulary column to 'dem' to reflect coefficient file
* initialize standard\_concept\_code and standard\_concept\_name columns to the value of the concept\_code column
* Convert concept\_code to strata codes from the coefficient file by changing all values to 'aa:dem:gender\_F'
* return the structured dataframe
* **def** structure\_females\_table(df):
* filtered\_df **=** df**.**copy()
* filtered\_df **=** filtered\_df[filtered\_df['sex\_at\_birth'] **==** 'Female']
* filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'sex\_at\_birth'])
* filtered\_df **=** filtered\_df[['person\_id',
* 'sex\_at\_birth']]
* filtered\_df **=** filtered\_df**.**rename(columns**=**{'sex\_at\_birth': 'concept\_code'})
* filtered\_df['standard\_vocabulary'] **=** 'dem'
* filtered\_df['standard\_concept\_code'] **=** filtered\_df['concept\_code']
* filtered\_df['standard\_concept\_name'] **=** filtered\_df['concept\_code']
* filtered\_df['concept\_code'] **=** 'aa\_dem\_gender\_F'
*
* **return** filtered\_df

**process data**

In [25]:

filtered\_df **=** structure\_females\_table(demographics\_df)

In [26]:

targets\_df **=** pd**.**concat([targets\_df, filtered\_df], ignore\_index**=True**)

In [27]:

targets\_df

**ad\_history (antidepressant history)**

In [28]:

**import** pandas **as** pd

ad\_history\_df **=** pd**.**read\_csv("./data/ad\_history\_df.csv")

**Read code text definitions file**

This file maps the concept\_codes for antidepressant history and their corresponding text definitions. Download the file through this URL:

[https://gmuedu-my.sharepoint.com/:x:/r/personal/klybarge\_gmu\_edu/Documents/Research/PCORI%20AD/Documentation%20Regression%20Models/Files%20for%20upload%20to%20notebook/data/antidepressant\_range\_definitions.csv?d=wd14359db516b4f4b9e557960b6ebbe85&csf=1&web=1&e=xrhI7Y](https://gmuedu-my.sharepoint.com/%3Ax%3A/r/personal/klybarge_gmu_edu/Documents/Research/PCORI%20AD/Documentation%20Regression%20Models/Files%20for%20upload%20to%20notebook/data/antidepressant_range_definitions.csv?d=wd14359db516b4f4b9e557960b6ebbe85&csf=1&web=1&e=xrhI7Y)

Upload the file in the 'data' folder with file name: antidepressant\_range\_definitions.csv

**import** pandas **as** pd

ad\_code\_names\_df **=** pd**.**read\_csv("./data/antidepressant\_range\_definitions.csv")

**Define function to structure ranges in the antidepressant history**

Name: structure\_ad\_ranges\_table

Input parameters:

* df - dataframe to be structured
* range\_column - column name for the range to be structured
* range\_to\_retain - list of ranges to retain
* ctype - ctype in the coefficients file

Return:

* structured dataframe

Processing:

* retain only rows where the values of range\_column in the range\_list
* retain only the following columns in this order: 'person\_id', range\_column
* remove duplicates based on person\_id, and range\_column
* rename column range\_column to concept\_code
* create standard\_vocabulary column with value as ctype
* create standard\_concept\_code column with value equal to the concept\_code column
* create standard\_concept\_name column by left joining with ad\_code\_names\_df on concept\_code
* convert values in concept\_code column to reflect the strata codes in the coefficient file by adding a prefix: "aa:ctype:"
* return the structured dataframe

**def** structure\_ad\_ranges\_table(df, range\_column, range\_to\_retain, ctype):

 filtered\_df **=** df**.**copy()

 filtered\_df **=** filtered\_df[filtered\_df[range\_column]**.**isin(range\_to\_retain)]

 filtered\_df **=** filtered\_df[['person\_id', range\_column]]

 filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', range\_column])

 filtered\_df **=** filtered\_df**.**rename(columns**=**{range\_column: 'concept\_code'})

 filtered\_df['standard\_vocabulary'] **=** ctype

 filtered\_df['standard\_concept\_code'] **=** filtered\_df['concept\_code']

 filtered\_df **=** pd**.**merge(filtered\_df, ad\_code\_names\_df, on**=**'concept\_code', how**=**'left')

 filtered\_df['concept\_code'] **=** (

 filtered\_df['concept\_code']

 **.**astype(str) *# Convert to string*

 **.**apply(**lambda** x: f"aa\_{ctype}\_{x}") *# Add prefix*

 )

 **return** filtered\_df

**Define function to structure response history in the antidepressant history**

Name: structure\_ad\_hist\_table

Input parameters:

* df - dataframe to be structured
* hist\_type - history type being processed (SN, SR, DN, DR)
* ctype - ctype in the coefficients file

Return:

* structured dataframe

Processing:

* retain only rows where the values of hist\_type column is 1
* create concept\_code column by appending the lowercase of the value of the ad\_grouping column with :hist\_type for example amitrityline:SN
* retain only the following columns in this order: 'person\_id', 'concept\_code'
* remove duplicates based on person\_id, and concept\_code
* create standard\_vocabulary column with value as ctype
* create standard\_concept\_code column with value equal to the concept\_code column
* create standard\_concept\_name column by left joining with ad\_code\_names\_df on concept\_code
* convert values in concept\_code column to reflect the strata codes in the coefficient file by adding a prefix: "aa:ctype:"
* return the structured dataframe

**def** structure\_ad\_hist\_table(df, hist\_type, ctype):

 filtered\_df **=** df**.**copy()

 filtered\_df **=** filtered\_df[filtered\_df[hist\_type] **==** 1]

 filtered\_df['concept\_code'] **=** filtered\_df['ad\_grouping']**.**str**.**lower() **+** f'\_{hist\_type}'

 filtered\_df **=** filtered\_df[['person\_id', 'concept\_code']]

 filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'concept\_code'])

 filtered\_df['standard\_vocabulary'] **=** ctype

 filtered\_df['standard\_concept\_code'] **=** filtered\_df['concept\_code']

 filtered\_df **=** pd**.**merge(filtered\_df, ad\_code\_names\_df, on**=**'concept\_code', how**=**'left')

 filtered\_df['concept\_code'] **=** (

 filtered\_df['concept\_code']

 **.**astype(str) *# Convert to string*

 **.**apply(**lambda** x: f"aa\_{ctype}\_{x}") *# Add prefix*

 )

 **return** filtered\_df

**process data**

.Define function to structure remission antidepressant history

Name: structure\_remission\_table

Input parameters:

df - dataframe to be structured

ctype - ctype in the coefficients file

Return:

structured dataframe Processing:

retain only rows where the values of Remission column is 1

create concept\_code column by appending the lowercase of the value of the ad\_grouping column with \_rem for example amitriptyline\_rem

retain only the following columns in this order: 'person\_id', 'concept\_code'

remove duplicates based on person\_id, and concept\_code

create standard\_vocabulary column with value as ctype

create standard\_concept\_code column with value equal to the concept\_code column

create standard\_concept\_name column by left joining with ad\_code\_names\_df on concept\_code

convert values in concept\_code column to reflect the strata codes in the coefficient file by adding a prefix: "aa\_ctype:" return the structured dataframe

**def** structure\_remission\_table(df, ctype):

 filtered\_df **=** df**.**copy()

 filtered\_df **=** filtered\_df[filtered\_df['Remission'] **==** 1]

 filtered\_df['concept\_code'] **=** filtered\_df['ad\_grouping']**.**str**.**lower() **+** '\_rem'

 filtered\_df **=** filtered\_df[['person\_id', 'concept\_code']]

 filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'concept\_code'])

 filtered\_df['standard\_vocabulary'] **=** ctype

 filtered\_df['standard\_concept\_code'] **=** filtered\_df['concept\_code']

 filtered\_df **=** pd**.**merge(filtered\_df, ad\_code\_names\_df, on**=**'concept\_code', how**=**'left')

 filtered\_df['concept\_code'] **=** (

 filtered\_df['concept\_code']

 **.**astype(str) *# Convert to string*

 **.**apply(**lambda** x: f"aa\_{ctype}\_{x}") *# Add prefix*

 )

 **return** filtered\_df

**Process data**

age range

Target age ranges are:

ager\_1319

ager\_2040

ager\_6579

ager\_8089

range\_column **=** 'age\_range'

range\_to\_retain **=** [

 'ager\_1319',

 'ager\_2040',

 'ager\_6579',

 'ager\_8089'

]

ctype **=** 'dem'

filtered\_df **=** structure\_ad\_ranges\_table(ad\_history\_df, range\_column, range\_to\_retain, ctype)

targets\_df **=** pd**.**concat([targets\_df, filtered\_df], ignore\_index**=True**)

targets\_df

**episodes range**

Target episode ranges are:

* epi\_2p
* range\_column **=** 'episodes\_range'
* range\_to\_retain **=** [
* 'epi\_2p'
* ]
* ctype **=** 'meas'
* filtered\_df **=** structure\_ad\_ranges\_table(ad\_history\_df, range\_column, range\_to\_retain, ctype)

targets\_df **=** pd**.**concat([targets\_df, filtered\_df], ignore\_index**=True**)

**remissions range**

Target remission ranges are:

* rem\_2p
* range\_column **=** 'remissions\_range'
* range\_to\_retain **=** [
* 'rem\_2p'
* ]
* ctype **=** 'meas'
* filtered\_df **=** structure\_ad\_ranges\_table(ad\_history\_df, range\_column, range\_to\_retain, ctype)
* targets\_df **=** pd**.**concat([targets\_df, filtered\_df], ignore\_index**=True**)

targets\_df

**antidepressants range**

Target antidepressants number range:

* adep\_23
* adep\_4p
* range\_column **=** 'antidepressants\_range'
* range\_to\_retain **=** [
* 'adep\_23',
* 'adep\_4p'
* ]
* ctype **=** 'meas'
* filtered\_df **=** structure\_ad\_ranges\_table(ad\_history\_df, range\_column, range\_to\_retain, ctype)

targets\_df **=** pd**.**concat([targets\_df, filtered\_df], ignore\_index**=True**)

targets\_df

**antidepressant response history (SN, SR, DN, DR)**

Need to be verbose about SR, SN, DR, and DN. It should be antidepressant:SR, antidepressant:SN, antidepressant:DR, antidepressant:DN (ex. amitriptyline:SR

**from** tqdm **import** tqdm *# for the status bar (there are some impatient analysts out there)*

ctype **=** 'meas'

hist\_types **=** [

 'SR',

 'SN',

 'DR',

 'DN'

]

*# for each history type, structure the table and append to targets\_df*

*# initialize the status bar*

**with** tqdm(total**=**len(hist\_types), desc**=**'Processing Antidepressant Response History') **as** pbar:

 **for** hist\_type **in** hist\_types:

 filtered\_df **=** structure\_ad\_hist\_table(ad\_history\_df, hist\_type, ctype)

 targets\_df **=** pd**.**concat([targets\_df, filtered\_df], ignore\_index**=True**)

 pbar**.**update(1)

targets\_df

**antidepressant remission**

In [47]:

ctype **=** 'meas'

filtered\_df **=** structure\_remission\_table(ad\_history\_df, ctype)

targets\_df **=** pd**.**concat([targets\_df, filtered\_df], ignore\_index**=True**)

targets\_df

**Save results in a csv file**

This step ensures that succeeding steps can proceed without re-running data processing of cohort containing target variables.

In [50]:

targets\_df**.**to\_csv('./data/targets\_df.csv', index**=False**)

**Independent variables standardize data set format**

We need an ability to query all medical events of a given person. This will be needed for likehood ratio calculations for feature selection of the predictors of predictors.

The dataset should have the following columns:

.person\_id - unique person\_id

.concept\_code - the code that will be included in the coefficient file

.standard\_vocabulary - the vocabulary concept code for the predictor inside of All of Us (if any) or code type of the created feature

.standard\_concept\_code - the concept code for the medical event inside of All of Us (if any) or code of the created feature

.standard\_concept\_name - the name for the medical event inside of All of Us (if any) or name of the created feature

The dataset will be formed from the following dataframes:

.demographics\_df

.ad\_history\_df

.all\_diseases\_df

.all\_procedures\_df

.all\_medications\_df

The resulting table will be called medical\_events\_df and will be stored in a csv file for further processing

In [ ]:

 **Initialize medical\_events\_df**

In [1]:

**import** pandas **as** pd

medical\_events\_df **=** pd**.**DataFrame(columns**=**['person\_id',

 'concept\_code',

 'standard\_vocabulary',

 'standard\_concept\_code',

 'standard\_concept\_name'])

**demographic\_df (female at birth)**

In [2]:

**import** pandas **as** pd

demographics\_df **=** pd**.**read\_csv("./data/demographics\_df.csv")

**Define function to structure females table**

Name: structure\_females\_table

Input parameters:

* df - dataframe to be structured

Return:

* structured dataframe

Processing:

* retain only rows where sex\_at\_birth is 'Female'
* remove duplicates based on person\_id, and sex\_at\_birth
* retain only the following columns in this order: 'person\_id', 'sex\_at\_birth'
* rename column sex\_at\_birth to concept\_code
* initialize standard\_vocabulary column to 'dem' to reflect coefficient file
* initialize standard\_concept\_code and standard\_concept\_name columns to the value of the concept\_code column
* Convert concept\_code to strata codes from the coefficient file by changing all values to 'aa:dem:gender\_F'
* return the structured dataframe
* **def** structure\_females\_table(df):
* filtered\_df **=** df**.**copy()
* filtered\_df **=** filtered\_df[filtered\_df['sex\_at\_birth'] **==** 'Female']
* filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'sex\_at\_birth'])
* filtered\_df **=** filtered\_df[['person\_id',
* 'sex\_at\_birth']]
* filtered\_df **=** filtered\_df**.**rename(columns**=**{'sex\_at\_birth': 'concept\_code'})
* filtered\_df['standard\_vocabulary'] **=** 'dem'
* filtered\_df['standard\_concept\_code'] **=** filtered\_df['concept\_code']
* filtered\_df['standard\_concept\_name'] **=** filtered\_df['concept\_code']
* filtered\_df['concept\_code'] **=** 'aa\_dem\_gender\_F'
*
* **return** filtered\_df

**Process data**

In [4]:

filtered\_df **=** structure\_females\_table(demographics\_df)

In [5]:

medical\_events\_df **=** pd**.**concat([medical\_events\_df, filtered\_df], ignore\_index**=True**)

In [6]:

medical\_events\_df

**all\_diseases\_df (conditions / diseases)**

In [7]:

**import** pandas **as** pd

all\_conditions\_df **=** pd**.**read\_csv('./data/all\_conditions\_df.csv', low\_memory**=False**)

**Define function to structure all diagnoses table**

Name: structure\_all\_diagnoses\_table

Input parameters:

* df - dataframe to be structured

Return:

* structured dataframe

Processing:

* remove duplicates based on person\_id, and standard\_concept\_code
* retain only the following columns in this order: 'person\_id', 'standard\_vocabulary', 'standard\_concept\_code', 'standard\_concept\_name'
* create column concept\_code with value equal to dx:standard\_vocabulary:standard\_concept\_code
* return the structured dataframe
* **def** structure\_all\_diagnoses\_table(df):
* filtered\_df **=** df**.**copy()
* filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'standard\_concept\_code'])
* filtered\_df **=** filtered\_df[['person\_id',
* 'standard\_vocabulary',
* 'standard\_concept\_code',
* 'standard\_concept\_name']]
* filtered\_df['concept\_code'] **=** (
* 'dx\_' **+**
* filtered\_df['standard\_vocabulary']**.**astype(str) **+**
* '\_' **+**
* filtered\_df['standard\_concept\_code']**.**astype(str)
* )
*
* **return** filtered\_df

**Process data**

In [9]:

filtered\_df **=** structure\_all\_diagnoses\_table(all\_conditions\_df)

In [10]:

medical\_events\_df **=** pd**.**concat([medical\_events\_df, filtered\_df], ignore\_index**=True**)

In [11]:

medical\_events\_df

**all\_procedures (procedures)**

In [12]:

**import** pandas **as** pd

all\_procedures\_df **=** pd**.**read\_csv('./data/all\_procedures\_df.csv', low\_memory**=False**)

**Define function to structure all procedures table**

Name: structure\_all\_procedures\_table

Input parameters:

* df - dataframe to be structured

Return:

* structured dataframe

Processing:

* retain only rows where standard\_vocabulary is 'CPT4' and 'SNOMED'
* remove duplicates based on person\_id, standard\_vocabulary, and standard\_concept\_code
* retain only the following columns in this order: 'person\_id', 'standard\_vocabulary', 'standard\_concept\_code', 'standard\_concept\_name'
* create column concept\_code with value equal to px:standard\_vocabulary:standard\_concept\_code
* return the structured dataframe

**def** structure\_all\_procedures\_table(df):

 filtered\_df **=** df**.**copy()

 filtered\_df **=** filtered\_df[

 (filtered\_df['standard\_vocabulary'] **==** 'CPT4') **|**

 (filtered\_df['standard\_vocabulary'] **==** 'SNOMED')

 ]

 filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'standard\_vocabulary', 'standard\_concept\_code'])

 filtered\_df **=** filtered\_df[['person\_id',

 'standard\_vocabulary',

 'standard\_concept\_code',

 'standard\_concept\_name']]

 filtered\_df['concept\_code'] **=** (

 'px\_' **+**

 filtered\_df['standard\_vocabulary']**.**astype(str) **+**

 '\_' **+**

 filtered\_df['standard\_concept\_code']**.**astype(str)

 )

 **return** filtered\_df

**Process data**

In [14]:

filtered\_df **=** structure\_all\_procedures\_table(all\_procedures\_df)

In [15]:

medical\_events\_df **=** pd**.**concat([medical\_events\_df, filtered\_df], ignore\_index**=True**)

In [16]:

medical\_events\_df

**all\_medications (medications)**

In [17]:

**import** pandas **as** pd

all\_medications\_grouped\_df **=** pd**.**read\_csv('./data/all\_medications\_grouped\_df.csv', low\_memory**=False**)

**Define function to structure all medications table**

Name: structure\_all\_medications\_table

Input parameters:

df - dataframe to be structured Return:

structured dataframe Processing:

remove duplicates based on person\_id, and ingredient\_list retain only the following columns in this order: 'person\_id', 'standard\_vocabulary', 'ingredient\_list', 'ingredient\_name\_list' rename column ingredient\_list to standard\_concept\_code rename column ingredient\_name\_list to standard\_concept\_name create column concept\_code with value equal to rx\_standard\_vocabulary\_standard\_concept\_code return the structured dataframe

In [18]:

**def** structure\_all\_medications\_table(df):

 filtered\_df **=** df**.**copy()

 filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'ingredient\_list'])

 filtered\_df **=** filtered\_df[['person\_id',

 'standard\_vocabulary',

 'ingredient\_list',

 'ingredient\_name\_list']]

 filtered\_df **=** filtered\_df**.**rename(columns**=**{

 'ingredient\_list' : 'standard\_concept\_code',

 'ingredient\_name\_list' : 'standard\_concept\_name'

 })

 filtered\_df['concept\_code'] **=** (

 'rx\_' **+**

 filtered\_df['standard\_vocabulary']**.**astype(str) **+**

 '\_' **+**

 filtered\_df['standard\_concept\_code']**.**astype(str)

 )

 **return** filtered\_df

In [19]:

filtered\_df **=** structure\_all\_medications\_table(all\_medications\_grouped\_df)

In [20]:

medical\_events\_df **=** pd**.**concat([medical\_events\_df, filtered\_df], ignore\_index**=True**)

In [21]:

medical\_events\_df

**ad\_history\_df (antidepressant history)**

In [22]:

**import** pandas **as** pd

ad\_history\_df **=** pd**.**read\_csv("./data/ad\_history\_df.csv")

In [23]:

ad\_history\_df**.**shape[0]

**Read code text definitions file**

This file maps the concept\_codes for antidepressant history and their corresponding text definitions. Download the file through this URL: [https://gmuedu-my.sharepoint.com/:x:/r/personal/klybarge\_gmu\_edu/Documents/Research/PCORI%20AD/Documentation%20Regression%20Models/Files%20for%20upload%20to%20notebook/data/antidepressant\_range\_definitions.csv?d=wd14359db516b4f4b9e557960b6ebbe85&csf=1&web=1&e=xrhI7Y](https://gmuedu-my.sharepoint.com/%3Ax%3A/r/personal/klybarge_gmu_edu/Documents/Research/PCORI%20AD/Documentation%20Regression%20Models/Files%20for%20upload%20to%20notebook/data/antidepressant_range_definitions.csv?d=wd14359db516b4f4b9e557960b6ebbe85&csf=1&web=1&e=xrhI7Y)

Upload the file in the 'data' folder with file name: antidepressant\_range\_definitions.csv

**import** pandas **as** pd

ad\_code\_names\_df **=** pd**.**read\_csv("./data/antidepressant\_range\_definitions.csv")

**Define function to structure ranges in the antidepressant history**

Name: structure\_all\_ad\_ranges\_table

Input parameters:

* df - dataframe to be structured
* range\_column - column name for the range to be structured
* range\_to\_retain - list of ranges to retain
* ctype - ctype in the coefficients file

Return:

* structured dataframe

Processing:

* retain only rows where the values of range\_column in the range\_list
* retain only the following columns in this order: 'person\_id', range\_column
* remove duplicates based on person\_id, and range\_column
* rename column range\_column to concept\_code
* create standard\_vocabulary column with value as ctype
* create standard\_concept\_code column with value equal to the concept\_code column
* create standard\_concept\_name column by left joining with ad\_code\_names\_df on concept\_code
* convert values in concept\_code column to reflect the strata codes in the coefficient file by adding a prefix: "aa:ctype:"
* return the structured dataframe
* **def** structure\_all\_ad\_ranges\_table(df, range\_column, range\_to\_retain, ctype):
* filtered\_df **=** df**.**copy()
* filtered\_df **=** filtered\_df[filtered\_df[range\_column]**.**isin(range\_to\_retain)]
* filtered\_df **=** filtered\_df[['person\_id', range\_column]]
* filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', range\_column])
* filtered\_df **=** filtered\_df**.**rename(columns**=**{range\_column: 'concept\_code'})
* filtered\_df['standard\_vocabulary'] **=** ctype
* filtered\_df['standard\_concept\_code'] **=** filtered\_df['concept\_code']
* filtered\_df **=** pd**.**merge(filtered\_df, ad\_code\_names\_df, on**=**'concept\_code', how**=**'left')
* filtered\_df['concept\_code'] **=** (
* filtered\_df['concept\_code']
* **.**astype(str) *# Convert to string*
* **.**apply(**lambda** x: f"aa\_{ctype}\_{x}") *# Add prefix*
* )
*
* **return** filtered\_df

**Define function to structure response history in the antidepressant history**

Name: structure\_ad\_hist\_table

Input parameters:

* df - dataframe to be structured
* hist\_type - history type being processed (SN, SR, DN, DR)
* ctype - ctype in the coefficients file

Return:

* structured dataframe

Processing:

* retain only rows where the values of hist\_type column is 1
* create concept\_code column by appending the lowercase of the value of the ad\_grouping column with :hist\_type for example amitrityline:SN
* retain only the following columns in this order: 'person\_id', 'concept\_code'
* remove duplicates based on person\_id, and concept\_code
* create standard\_vocabulary column with value as ctype
* create standard\_concept\_code column with value equal to the concept\_code column
* create standard\_concept\_name column by left joining with ad\_code\_names\_df on concept\_code
* convert values in concept\_code column to reflect the strata codes in the coefficient file by adding a prefix: "aa:ctype:"
* return the structured dataframe

**def** structure\_all\_ad\_hist\_table(df, hist\_type, ctype):

 filtered\_df **=** df**.**copy()

 filtered\_df **=** filtered\_df[filtered\_df[hist\_type] **==** 1]

 filtered\_df['concept\_code'] **=** filtered\_df['ad\_grouping']**.**str**.**lower() **+** f'\_{hist\_type}'

 filtered\_df **=** filtered\_df[['person\_id', 'concept\_code']]

 filtered\_df **=** filtered\_df**.**drop\_duplicates(subset**=**['person\_id', 'concept\_code'])

 filtered\_df['standard\_vocabulary'] **=** ctype

 filtered\_df['standard\_concept\_code'] **=** filtered\_df['concept\_code']

 filtered\_df **=** pd**.**merge(filtered\_df, ad\_code\_names\_df, on**=**'concept\_code', how**=**'left')

 filtered\_df['concept\_code'] **=** (

 filtered\_df['concept\_code']

 **.**astype(str) *# Convert to string*

 **.**apply(**lambda** x: f"aa\_{ctype}\_{x}") *# Add prefix*

 )

 **return** filtered\_df

**process data**

**Age range**

Age ranges to retain will be as follows:

* 13 to 19 (ager\_1319)
* 20 to 40 (ager\_2040)
* 41 to 64 (ager\_4164)
* 65 to 79 (ager\_6579)
* 80 to 89 (ager\_8089)

Remove the following age range:

* 0 to 12 (ager\_0012) - the study does not include children and All of Us does not include them anyway
* range\_column **=** 'age\_range'
* range\_to\_retain **=** [
* 'ager\_1319',
* 'ager\_2040',
* 'ager\_4164',
* 'ager\_6579',
* 'ager\_8089'
* ]
* ctype **=** 'dem'
* filtered\_df **=** structure\_all\_ad\_ranges\_table(ad\_history\_df, range\_column, range\_to\_retain, ctype)
* medical\_events\_df **=** pd**.**concat([medical\_events\_df, filtered\_df], ignore\_index**=True**)
* medical\_events\_df

**episodes range**

Target episode ranges are:

* epi\_01
* epi\_2p

range\_column **=** 'episodes\_range'

range\_to\_retain **=** [

 'epi\_01',

 'epi\_2p'

]

ctype **=** 'meas'

filtered\_df **=** structure\_all\_ad\_ranges\_table(ad\_history\_df, range\_column, range\_to\_retain, ctype)

In [31]:

medical\_events\_df **=** pd**.**concat([medical\_events\_df, filtered\_df], ignore\_index**=True**)

In [32]:

medical\_events\_df

**remissions range**

Target remission ranges are:

* rem\_01
* rem\_2p

range\_column **=** 'remissions\_range'

range\_to\_retain **=** [

 'rem\_01',

 'rem\_2p'

]

ctype **=** 'meas'

filtered\_df **=** structure\_all\_ad\_ranges\_table(ad\_history\_df, range\_column, range\_to\_retain, ctype)

medical\_events\_df **=** pd**.**concat([medical\_events\_df, filtered\_df], ignore\_index**=True**)

medical\_events\_df

**antidepressants range**

Target antidepressants number range:

* adep\_01
* adep\_23
* adep\_4p

range\_column **=** 'antidepressants\_range'

range\_to\_retain **=** [

 'adep\_01',

 'adep\_23',

 'adep\_4p'

]

ctype **=** 'meas'

filtered\_df **=** structure\_all\_ad\_ranges\_table(ad\_history\_df, range\_column, range\_to\_retain, ctype)

medical\_events\_df **=** pd**.**concat([medical\_events\_df, filtered\_df], ignore\_index**=True**)

medical\_events\_df

**antidepressant response history (SN, SR, DN, DR)**

Need to be verbose about SR, SN, DR, and DN. It should be antidepressant:SR, antidepressant:SN, antidepressant:DR, antidepressant:DN (ex. amitriptyline:SR)antidepressant response history (SN, SR, DN, DR)

**from** tqdm **import** tqdm *# for the status bar (there are some impatient analysts out there)*

ctype **=** 'meas'

hist\_types **=** [

 'SR',

 'SN',

 'DR',

 'DN'

]

*# for each history type, structure the table and append to targets\_df*

*# initialize the status bar*

**with** tqdm(total**=**len(hist\_types), desc**=**'Processing Antidepressant Response History') **as** pbar:

 **for** hist\_type **in** hist\_types:

 filtered\_df **=** structure\_all\_ad\_hist\_table(ad\_history\_df, hist\_type, ctype)

 medical\_events\_df **=** pd**.**concat([medical\_events\_df, filtered\_df], ignore\_index**=True**)

 pbar**.**update(1)

medical\_events\_df

**Save results in a csv file**

This step ensures that succeeding steps can proceed without re-running data processing of medical events.

In [41]:

medical\_events\_df**.**to\_csv('./data/medical\_events\_df.csv', index**=False**)

# New Notebook –create a new notebook name it as : Ex: Keerti Likelihood ratio notebook and start from the given code below

# This is important because in order to calculate likelihood ratios it may take 60 hours for all the targets so again we will be using notebook\_to\_run ( notebook used to run codes in background used earlier in creating antidepressant data).

**Load data sets to SQL Server**

**Read datasets for loading**

In [42]:

**import** pandas **as** pd

targets\_df **=** pd**.**read\_csv('./data/targets\_df.csv', low\_memory**=False**)

medical\_events\_df **=** pd**.**read\_csv('./data/medical\_events\_df.csv', low\_memory**=False**)

targets\_df**.**head()

targets\_df**.**info()

targets\_df['concept\_code']**.**nunique()

medical\_events\_df**.**info()

medical\_events\_df['concept\_code']**.**nunique()

# see the results should be exactly the same important step before proceeding notebook to run:



**Create the database and load the tables**

The folder 'db' needs to be created for this step

In [48]:

*# establish the database*

**import** sqlite3

**%**sql sqlite:**///**db**/**mentalhealth**.**db

*# Establish a connection to SQLite database*

conn **=** sqlite3**.**connect('./db/mentalhealth.db')

*# Write the dataframes to SQLite*

targets\_df**.**to\_sql('targets', conn, if\_exists**=**'replace', index**=False**)

medical\_events\_df**.**to\_sql('medical\_events', conn, if\_exists**=**'replace', index**=False**)

**%%**sql

**--** Add **and** populate new\_code column **in** targets table **for** join **and** where clauses

ALTER TABLE targets

ADD COLUMN new\_code TEXT;

UPDATE targets

SET new\_code **=** standard\_vocabulary **||** standard\_concept\_code;

**--** Add **and** populate new\_code column **in** targets table **for** join **and** where clauses

ALTER TABLE medical\_events

ADD COLUMN new\_code TEXT;

UPDATE medical\_events

SET new\_code **=** standard\_vocabulary **||** standard\_concept\_code;

**#Test SQL Server**

In [52]:

**%%**sql

SELECT **\*** FROM targets LIMIT 5

**%%**sql

SELECT **\*** FROM medical\_events LIMIT 5

**%%**sql

PRAGMA table\_info(medical\_events);

**#Close the database connection**

In [56]:

conn**.**close()

**Create work input file**

This file contains a list of predictors (target variables) that should be processed. This will allow for work to be distributed and also allow for work to be stopped and resumed.

Download the main workfile for the Missing Values project from this

 URL: [https://gmuedu-my.sharepoint.com/:x:/r/personal/klybarge\_gmu\_edu/Documents/Research/PCORI%20AD/Documentation%20Regression%20Models/Files%20for%20upload%20to%20notebook/data/missing\_values\_workfile.csv?d=wd21a8c71325b4ec68fef6e71d3eb2a82&csf=1&web=1&e=4kPCZq](https://gmuedu-my.sharepoint.com/%3Ax%3A/r/personal/klybarge_gmu_edu/Documents/Research/PCORI%20AD/Documentation%20Regression%20Models/Files%20for%20upload%20to%20notebook/data/missing_values_workfile.csv?d=wd21a8c71325b4ec68fef6e71d3eb2a82&csf=1&web=1&e=4kPCZq)

Offline, retain only rows for your assigned target variables (ex. dx codes only, or rx codes only)

This work file will need to be uploaded in a new folder called 'workfile' with the name 'missing\_values\_workfile.csv'

Workfile:

concept\_code - the concept code in the coefficient file (note: dxo and dxi were compressed to dx)

lr\_completed - status of likelihood ratio calculations: 0 - not completed, 1 - completed

lasso\_completed - status of lasso regression feature selection: 0 - not completed, 1 - completed

regression\_completed - status of logistic regression model creation: 0 - not completed, 1 - completed

csv\_generated - status of csv file generation: 0 - not completed, 1 - completed

**import** pandas **as** pd

workfile\_df **=** pd**.**read\_csv('./workfile/missing\_values\_workfile.csv')

**Initial feature selection using likelihood ratios**

**Generate remaining work from the workfile**

Filter workfile based on criteria that lr\_completed is equal to zero (0)

You can optionally set a work\_limit to limit the number of rows that you will work on. By default, the work limit is set to the number of rows remaining that meet the criteria. The work limit cannot be less than 1 and greater than the rows of remaining\_work\_df

**import** pandas **as** pd

**import** sqlite3

**from** contextlib **import** contextmanager

**from** tqdm **import** tqdm *# for the status bar (there are some impatient analysts out there)*

**import** re

**import** os

**from** datetime **import** datetime

**import** random

**import** string

remaining\_work\_df **=** workfile\_df[workfile\_df['lr\_completed'] **==** 0]

#set the worklimit to 4 to check the code (testing) before running the notebook to run

work\_limit **=** remaining\_work\_df**.**shape[0] *# default is number of rows of remaining work to do*

*# you can also override the work\_limit by uncommenting the code below and changing the number assigned to the work\_limit*

work\_limit **=** 2 *#place integer here*

*#work\_limit = max(1, min(work\_limit, remaining\_work\_df.shape[0]))*

remaining\_work\_df

**Function defintions**

For modularity and ease of understanding, the project chose to use functions for each individual SQL interaction. This way, should a particular query need to be updated, only the relevant function is updated and the rest of the logic remain intact.

In [105]:

@contextmanager

**def** get\_db\_connection(db\_file):

 """Context manager for database connections"""

 conn **=** sqlite3**.**connect(db\_file)

 **try**:

 **yield** conn

 **finally**:

 conn**.**close()

In [106]:

**def** create\_filtered\_targets(conn, current\_target):

 """Create filtered targets from new\_targets"""

 cursor **=** conn**.**cursor()

 cursor**.**execute("""

 CREATE TEMPORARY TABLE filtered\_targets AS

 SELECT \*

 FROM targets

 WHERE concept\_code = ?;

 """, (current\_target,))

 conn**.**commit()

In [107]:

**def** create\_filtered\_medical\_events(conn):

 """Create filtered medical events table"""

 cursor **=** conn**.**cursor()

 cursor**.**execute("""

 CREATE TEMPORARY TABLE filtered\_medical\_events AS

 SELECT nme.\*

 FROM medical\_events nme

 WHERE nme.new\_code NOT IN (

 SELECT DISTINCT new\_code

 FROM filtered\_targets

 );

 """)

 conn**.**commit()

In [108]:

**def** add\_target\_column\_to\_filtered\_events(conn):

 """Add target column to filtered\_medical\_events based on person\_id matches"""

 cursor **=** conn**.**cursor()

 *# First add the new column*

 cursor**.**execute("""

 ALTER TABLE filtered\_medical\_events

 ADD COLUMN target INTEGER DEFAULT 0;

 """)

 *# Then update it where person\_id matches*

 cursor**.**execute("""

 UPDATE filtered\_medical\_events

 SET target = 1

 WHERE person\_id IN (

 SELECT DISTINCT person\_id

 FROM filtered\_targets

 );

 """)

 conn**.**commit()

In [109]:

**def** create\_lr\_results\_table(conn):

 """Create likelihood ratio table"""

 cursor **=** conn**.**cursor()

 cursor**.**execute("""

 CREATE TEMPORARY TABLE lr\_results AS

 SELECT concept\_code,

 new\_code,

 SUM(CASE WHEN target = 1 THEN 1 ELSE 0 END) AS CodeTarget,

 SUM(CASE WHEN target = 0 THEN 1 ELSE 0 END) AS CodeNoTarget,

 (SELECT COUNT(DISTINCT person\_id) FROM filtered\_medical\_events WHERE target = 1) AS TotalTarget,

 (SELECT COUNT(DISTINCT person\_id) FROM filtered\_medical\_events WHERE target = 0) AS TotalNoTarget,

 COUNT(\*) AS TotalCode

 FROM filtered\_medical\_events

 GROUP BY standard\_concept\_code;

 """)

 conn**.**commit()

In [110]:

**def** add\_lr\_column\_to\_lr\_results(conn):

 """Add likelihood ratio column to lr\_results table"""

 cursor **=** conn**.**cursor()

 *# First add the new column*

 cursor**.**execute("""

 ALTER TABLE lr\_results

 ADD COLUMN LR REAL;

 """)

 *# Then update it with the CASE statement*

 cursor**.**execute("""

 UPDATE lr\_results

 SET LR = CASE

 WHEN CodeTarget = 0 THEN 1.0 / (TotalCode + 1)

 WHEN CodeNoTarget = 0 THEN TotalCode + 1

 ELSE (CodeTarget \* 1.0 / TotalTarget) / (CodeNoTarget \* 1.0 / TotalNoTarget)

 END;

 """)

 conn**.**commit()

In [111]:

**def** add\_lr\_ranking\_column\_to\_lr\_results(conn):

 """create a column that will be used to rank likelihood ratios in the lr\_results table"""

 cursor **=** conn**.**cursor()

 *# Add the new column*

 cursor**.**execute("ALTER TABLE lr\_results ADD COLUMN lr\_ranking REAL")

 *# Update the column based on the condition*

 cursor**.**execute("""

 UPDATE lr\_results

 SET lr\_ranking = CASE

 WHEN LR >= 1.0 THEN LR

 ELSE 1.0 / LR

 END

 """)

 conn**.**commit()

In [112]:

**def** create\_filtered\_lr\_results(conn, n\_features\_needed, min\_count):

 """Create temporary table of filtered\_lr\_results containing top n\_features by lr\_ranking"""

 cursor **=** conn**.**cursor()

 cursor**.**execute("""

 CREATE TEMPORARY TABLE filtered\_lr\_results AS

 SELECT \*

 FROM lr\_results

 WHERE TotalCode >= ?

 ORDER BY lr\_ranking DESC

 LIMIT ?;

 """, (min\_count, n\_features\_needed))

 conn**.**commit()

In [113]:

**def** create\_cohort\_table(conn):

 """Create temporary table for the cohort from filtered\_medical\_events retainining person\_id and target"""

 cursor **=** conn**.**cursor()

 cursor**.**execute("""

 CREATE TEMPORARY TABLE cohort AS

 SELECT person\_id, target

 FROM filtered\_medical\_events

 GROUP BY person\_id;

 """)

 conn**.**commit()

In [114]:

**def** create\_selected\_events\_table(conn):

 """Create temporary table selected\_events from filtered\_medical\_events and filtered\_lr\_results"""

 cursor **=** conn**.**cursor()

 cursor**.**execute("""

 CREATE TEMPORARY TABLE selected\_events AS

 SELECT DISTINCT me.person\_id,

 me.concept\_code,

 me.new\_code

 FROM filtered\_medical\_events me

 WHERE me.new\_code IN (

 SELECT DISTINCT new\_code

 FROM filtered\_lr\_results

 );

 """)

 conn**.**commit()

In [115]:

**def** sanitize\_column\_name(name, max\_length**=**100):

 """

 Make column names SQL-safe and ensure they fit SQLite length limits.

 If the name exceeds max\_length, truncate it and append random characters for uniqueness.

 Args:

 name (str): Original column name

 max\_length (int): Maximum allowed length for the column name

 Returns:

 str: Sanitized and length-compliant column name

 """

 *# First sanitize the name by replacing invalid characters*

 sanitized **=** re**.**sub(r'[^a-zA-Z0-9\_]', '\_', name)

 *# If the name is already within length limits, return it*

 **if** len(sanitized) **<=** max\_length:

 **return** sanitized

 *# Generate random suffix (using 6 characters)*

 suffix\_length **=** 6

 random\_suffix **=** ''**.**join(random**.**choices(string**.**ascii\_lowercase **+** string**.**digits, k**=**suffix\_length))

 *# Truncate the original name to fit the max length minus the suffix length*

 truncated\_length **=** max\_length **-** suffix\_length

 truncated\_name **=** sanitized[:truncated\_length]

 *# Combine truncated name with random suffix*

 final\_name **=** f"{truncated\_name}{random\_suffix}"

 **return** final\_name

In [116]:

**def** get\_unique\_concepts(cursor, table\_name):

 cursor**.**execute(f"SELECT DISTINCT concept\_code FROM {table\_name}")

 **return** [row[0] **for** row **in** cursor**.**fetchall()]

In [117]:

**def** create\_temp\_analysis\_table(conn):

 """Create temporary table temp\_analysis with left join and handle missing values"""

 cursor **=** conn**.**cursor()

 cursor**.**execute("""

 CREATE TEMPORARY TABLE temp\_analysis AS

 SELECT

 c.person\_id,

 c.target,

 COALESCE(se.concept\_code, 'Other') as concept\_code

 FROM cohort c

 LEFT JOIN selected\_events se ON c.person\_id = se.person\_id;

 """)

 conn**.**commit()

In [118]:

**def** create\_analysis\_dummies\_table(conn):

 """Create temporary table analysis\_dummies to convert concept\_code to dummy variable columns"""

 cursor **=** conn**.**cursor()

 *# Get unique concept codes*

 unique\_concepts **=** get\_unique\_concepts(cursor, 'temp\_analysis')

 *# Create case statements for dummy variables*

 case\_statements **=** []

 **for** concept **in** unique\_concepts:

 safe\_concept **=** f"{sanitize\_column\_name(concept)}"

 case\_statements**.**append(f"""

 CASE WHEN concept\_code = '{concept}' THEN 1 ELSE 0 END as {safe\_concept}

 """)

 *# Create temporary analysis\_dummies table*

 create\_dummies\_query **=** f"""

 CREATE TEMPORARY TABLE analysis\_dummies AS

 SELECT

 \*,

 {','**.**join(case\_statements)}

 FROM temp\_analysis

 """

 cursor**.**execute(create\_dummies\_query)

 *# Drop the concept\_code column*

 cursor**.**execute(f"ALTER TABLE analysis\_dummies DROP COLUMN concept\_code")

 *# Drop the Other column*

 cursor**.**execute(f"ALTER TABLE analysis\_dummies DROP COLUMN Other")

 conn**.**commit()

In [119]:

**def** create\_final\_analysis\_table(conn):

 """Create temporary table analysis\_final group by PERSON\_ID"""

 cursor **=** conn**.**cursor()

 *# Get all column names except PERSON\_ID for MAX aggregation*

 cursor**.**execute("PRAGMA table\_info(analysis\_dummies)")

 columns **=** [col[1] **for** col **in** cursor**.**fetchall() **if** (col[1] **!=** 'person\_id' **and** col[1] **!=** 'target')]

 max\_statements **=** [f"MAX({col}) as {col}" **for** col **in** columns]

 *# Create final grouped table*

 group\_query **=** f"""

 CREATE TEMPORARY TABLE final\_analysis AS

 SELECT

 MAX(target) AS target,

 {', '**.**join(max\_statements)}

 FROM analysis\_dummies

 GROUP BY person\_id

 """

 cursor**.**execute(group\_query)

 conn**.**commit()

In [120]:

**def** create\_persistent\_analysis\_table(conn, current\_target):

 """Create persistent table from analysis"""

 cursor **=** conn**.**cursor()

 table\_name **=** f"{current\_target}\_analysis"

 cursor**.**execute(f"DROP TABLE IF EXISTS {table\_name}")

 cursor**.**execute(f"""

 CREATE TABLE {table\_name} AS

 SELECT \*

 FROM final\_analysis;

 """)

 conn**.**commit()

In [121]:

**def** write\_alert(message):

 """Helper function to write timestamped alerts to log file"""

 timestamp **=** datetime**.**now()**.**strftime('%Y-%m-%d %H:%M:%S')

 log\_message **=** f"[{timestamp}] {message}\n"

 **with** open('./output/logs.txt', 'a') **as** log\_file:

 log\_file**.**write(log\_message)

In [123]:

**def** create\_persistent\_lr\_results(conn, current\_target):

 """Create persistent table from lr\_results temporary table"""

 cursor **=** conn**.**cursor()

 table\_name **=** f"{current\_target}\_lr\_results"

 *# Drop the table if it already exists*

 cursor**.**execute(f"DROP TABLE IF EXISTS {table\_name}")

 *# Create the persistent table from lr\_results*

 cursor**.**execute(f"""

 CREATE TABLE {table\_name} AS

 SELECT \*

 FROM lr\_results;

 """)

 conn**.**commit()

In [124]:

**def** process\_likehood\_ratio(db\_file, current\_target, min\_count, n\_features\_needed):

 """Process the likelihood ratio for the current target in a single db connection"""

 **with** get\_db\_connection(db\_file) **as** conn:

 *# select only rows in new\_targets that match the current\_target*

 *# save to temp table filtered\_targets*

 create\_filtered\_targets(conn, current\_target)

 *# Count rows if target exist*

 cursor **=** conn**.**cursor()

 cursor**.**execute("SELECT COUNT(\*) FROM filtered\_targets")

 row\_count **=** cursor**.**fetchone()[0]

 **if** row\_count **==** 0:

 alert\_msg **=** f"ALERT!: No target data available for {current\_target}"

 print(f"\n{alert\_msg}")

 write\_alert(alert\_msg)

 **return**

 *# remove rows from new\_medical\_events that are the same as the current\_target*

 *# save to temp table filtered\_medical\_events*

 create\_filtered\_medical\_events(conn)

 *# add a target column to filtered\_medical\_events if person\_id is in filtered\_targets*

 add\_target\_column\_to\_filtered\_events(conn)

 *# create temp table lr\_results to calculate likelihood ratios*

 create\_lr\_results\_table(conn)

 *# add column LR to lr\_results that contain calculated likelihood ratios*

 add\_lr\_column\_to\_lr\_results(conn)

 *# add column lr\_ranking to lr\_results to rank likelihood ratios*

 add\_lr\_ranking\_column\_to\_lr\_results(conn)

 *# save lr\_results in a persistent table for later analysis*

 *# table name format {current\_target}\_lr\_results*

 create\_persistent\_lr\_results(conn, current\_target)

 *# create temp table filtered\_lr\_results that have the top n\_features\_needed values in lr\_ranking*

 *# where TotalCode >= min\_count*

 create\_filtered\_lr\_results(conn, n\_features\_needed, min\_count)

 *# create temp table cohort that contains unique persons in filtered\_medical\_events*

 *# retain only the person\_id and cohort columns*

 create\_cohort\_table(conn)

 *# create temp table selected\_events that only contain rows where the codes are part of filtered\_lr\_results*

 create\_selected\_events\_table(conn)

 *# create a temp table temp\_analysis that contains the cohort and any unique code from the selected\_events table*

 *# missing values will be recoded to 'Other'*

 create\_temp\_analysis\_table(conn)

 *# create a temp table analysis\_dummies that converts the concept\_code column to dummy variable columns*

 *# the concept\_code column and Other dummy variable are dropped at the end*

 create\_analysis\_dummies\_table(conn)

 *# create a temp table final\_analysis by grouping the columns by person\_id aggregated using MAX()*

 *# the person\_id column is dropped at the end*

 create\_final\_analysis\_table(conn)

 *# save final\_analysis into a persistent table*

 *# table name format {current\_target}\_analysis*

 create\_persistent\_analysis\_table(conn, current\_target)

**Execute likehood ratio calculations and save in database**

Upon completion of these process, the following will be done:

.A persistent table containing all likelihood ratio for the target concept\_code will be created. Table name format will be target\_lr\_results (for example: dx\_ICD9\_311\_lr\_results)

.A persistent table containing basis for an analysis dataframe for regression will be created. Table name format will be target\_analysis (for example: dx\_ICD9\_311\_analysis)

.All other tables will be created as temporary tables.

.This ensures the following: .multiple users can access the same database .The size of the database is kept at minimum

*# Define parameters*

db\_file **=** './db/mentalhealth.db'

min\_count **=** 10

n\_features\_needed **=** 500

In [126]:

*# start the SQL server*

**%**sql sqlite:**///**db**/**mentalhealth**.**db

In [127]:

**with** tqdm(work\_limit, desc**=**'Processing Likelihood Ratios') **as** pbar:

 i **=** 0

 **for** \_, row **in** workfile\_df**.**iterrows():

 *# skip if work is already done*

 **if** row['lr\_completed'] **==** 1:

 **continue**

 current\_target **=** row['concept\_code']

 process\_likehood\_ratio(db\_file, current\_target, min\_count, n\_features\_needed)

 *# update the workfile*

 workfile\_df**.**at[row**.**name, 'lr\_completed'] **=** 1

 pbar**.**update(1)

 i **+=** 1

 **if** i **==** work\_limit:

 **break**

In [132]:

workfile\_df**.**head(50)

workfile\_df**.**to\_csv('./workfile/missing\_values\_workfile.csv', index**=False**)

#run till last for initial analysis to check for any errors by limiting the work limit to 2 or 3 , after that comment the work limit line seen in the image so that we can run the entire workfile with no limits so we can get likelihood ratios for all the targets

