COVID-19 Vulnerability Index Tutorial

The CV19 Index is designed to work off of medical claims data. The underlying fields in this data are fairly well standardized, but the actual table and field layouts differ dramatically from one data set to another. In this tutorial, we take an example data set that contains synthetic medical claims data and show the operations necessary to create an input file for the CV19 Index python code.

To being, we import some standard libraries

In [8]:
```
%xmode Plain

import pandas as pd
import numpy as np
import warnings
import random
import regex as re
import json
from pkg_resources import resource_filename

warnings.filterwarnings("ignore")
```

```
Exception reporting mode: Plain
```

We next need to import the `cv19index` module. If you have installed the package using pip, then it will be available to import. If you are running from a checkout of the git repository, you can add the parent directory to your python path to gain access to the module. The code below attempts to determine the correct approach and import `cv19index` .

In [20]:
```
try:
    import cv19index
except ModuleNotFoundError:
    import sys
    sys.path.append("../")
    import cv19index
```

Input files

The example data is contained in the `data` directory. It consists of 4 comma-separated value (CSV) files, with headers. The data is synthetic and not real claims data.

The 4 files are typical of files that are available containing medical claims data. They are:

- person.csv - A person file that contains basic demographics about each member.
- eligibility.csv - A file that contains one row for each month of eligibility for each member. This file is not used in the current features but is generally present in claims data.
- inpatient.csv - Contains data on inpatient claims, including diagnosis and procedure codes. Only diagnosis codes are used in these models
- outpatient.csv - Contains data on outpatient claims, including diagnosis and procedure codes. Only diagnosis codes are used in these models

We load each file into a pandas DataFrame and transform them to match the input for our model.

In [21]:
```
person_df = pd.read_csv('data/person.csv')
eligibility_df = pd.read_csv('data/eligibility.csv')
inpatient_df = pd.read_csv('data/inpatient.csv')
outpatient_df = pd.read_csv('data/outpatient.csv')
```

Basic data exploration

The data set contains data on 1,000 people. Each person can have 0 to many claims.

In [22]:
```
person_df.shape, eligibility_df.shape, inpatient_df.shape, outpatient_df.shape
```

Out[22]:
```
((1000, 7), (21110, 2), (348, 33), (68133, 22))
```

The person table contains basic demographics

In [23]:
```
person_df.head()
```

Out[23]:

	personId	gender	birthYear	deathDate	zipCode	flag	age
0	772775338f7ee353	1	1947-10-24T00:00:00.000-06:00	NaN	9426	False	73
1	d45d10ed2ec861c4	1	1931-10-19T00:00:00.000-06:00	NaN	41330	False	89
2	590bda01eeb795ee	1	1949-03-19T00:00:00.000-06:00	NaN	37542	False	71
3	14ad855b9fc39501	2	1952-12-19T00:00:00.000-06:00	NaN	41127	True	68
4	48b8f6c7a0435491	2	1953-11-23T00:00:00.000-06:00	NaN	16400	False	67

In [24]:
```
eligibility_df.head()
```

Out[24]:

	personId	date
0	0ab879955726c125	2017-01-01T00:00:00.000-06:00
1	0ab879955726c125	2017-02-01T00:00:00.000-06:00
2	0ab879955726c125	2017-03-01T00:00:00.000-06:00
3	0ab879955726c125	2017-04-01T00:00:00.000-05:00
4	0ab879955726c125	2017-05-01T00:00:00.000-05:00

Each row of the inpatient table can have several diagnosis codes. In this model, we use all of the available diagnosis codes to build the features.

In [25]:
```
inpatient_df.head()
```

Out[25]:

	personId	claimId	admitDate	dischargeDate	drg	dx1	dx2	dx3	dx4	dx5	...	icdProc3
0	c1ba927fb6d92cfe	aefde08ab8bb5460	2018-10-30T00:00:00.000-05:00	2018-11-09T00:00:00.000-06:00	264	I2119	G546	I2720	N071	I4510	...	02H64JZ
1	ca6bfe760572c820	c5b734d756438656	2017-05-14T00:00:00.000-05:00	2017-05-17T00:00:00.000-05:00	55	I6201	Z9352	J9600	I151	I2602	...	NaN
2	ca6bfe760572c820	853bc8bfa326acd9	2017-06-18T00:00:00.000-05:00	2017-06-21T00:00:00.000-05:00	471	S82002C	S92426D	I409	I5084	I472	...	NaN
3	ca6bfe760572c820	d0de7b57f83b6c58	2017-10-12T00:00:00.000-05:00	2017-10-15T00:00:00.000-05:00	245	B5881	Z9912	Z950	Q279	I213	...	NaN
4	cd517bfb4ecaa586	b924caa27d2841fd	2018-06-11T00:00:00.000-05:00	2018-06-13T00:00:00.000-05:00	289	A3951	I221	I300	NaN	I1311	...	NaN

5 rows × 33 columns

The outpatient table is similar to the inpatient table but has a single `serviceDate` instead of `admitDate` and `dischargeDate`, which are specific to overnight hospital stays.

In [26]:
```
outpatient_df.head()
```

Out[26]:

	personId	claimId	serviceDate	hcpcs	dx1	dx2	dx3	dx4	dx5	dx6	...	dx9	dx10	dx11	dx12
0	0ab879955726c125	76f3f6ff57e6c2ae	2017-02-03T00:00:00.000-06:00	G2011	Z5111	D0582	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
1	0ab879955726c125	36e93e0e4e2705d2	2017-02-03T00:00:00.000-06:00	NaN	D0592	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
2	0ab879955726c125	3e0bc41dc76c2a55	2017-02-14T00:00:00.000-06:00	NaN	Z86000	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
3	0ab879955726c125	3e0bc41dc76c2a55	2017-02-14T00:00:00.000-06:00	NaN	Z114	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN
4	0ab879955726c125	3e0bc41dc76c2a55	2017-02-14T00:00:00.000-06:00	J8670	Z1151	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN

5 rows × 22 columns

Data Preparation

The ICD-10 codes contained in the files did not contain periods, which is common in claims databases. The function below inserts the period at the appropriate place in the code.

In [27]:
```
def cleanICD10Syntax(code):
    if len(code) > 3 and '.' not in code:
        return code[:3] + '.' + code[3:]
    else:
        code
```

The following function takes in the above DataFrames and performs the following tasks:-

1. Filters the inpatient and outpatient dataframe to have admits and claims within a year of the asOfDate passed.
2. Calculates Age
3. Formats the Dataframe for the input of the model
4. For all CCSR Codes :-
    A. Finds all the ICD-10 codes for the CCSR mentioned.
    B. Filter the inpatient and outpatient dataframe to have only those admits and claims which have the ICD10 codes for the CCSR within the year of asOfDate.
    C. Map it back to the person

In [28]:
```
def getTestDataFrame(person_df, eligibility_df, inpatient_df, outpatient_df, asOfDate, diagnosis_columns):

    #Getting diagnosis within the past year of asOfDate
    asOfPastYear = str(pd.to_datetime(asOfDate) - pd.DateOffset(years=1))
    inpatient_df = inpatient_df[(asOfPastYear <= inpatient_df['admitDate']) &  (inpatient_df['admitDate'] <=
asOfDate)]
    outpatient_df = outpatient_df[(asOfPastYear <= outpatient_df['serviceDate']) &  (outpatient_df['serviceDa
te'] <= asOfDate)]

    inpatient_er_visit = inpatient_df[inpatient_df['edAdmit']==True][['personId', 'admitDate']].groupby('pers
onId').admitDate.nunique().reset_index()
    outpatient_er_visit = outpatient_df[outpatient_df['edVisit']==True][['personId', 'serviceDate']].groupby(
'personId').serviceDate.nunique().reset_index()

    #Calculating Age, # of ER Visits, # of Admissions and Inpatient days
    person_df['Age'] = person_df['birthYear'].apply(lambda x : pd.to_datetime('now').year - pd.to_datetime(x)
.year)
    person_df = person_df.merge(inpatient_er_visit, how='left').merge(outpatient_er_visit, how='left')
    person_df['# of ER Visits (12M)'] = person_df['admitDate'] + person_df['serviceDate']

    inpatient_df['Inpatient Days'] = inpatient_df[['dischargeDate','admitDate']].apply(lambda x: (pd.to_datet
ime(x.dischargeDate) - pd.to_datetime(x.admitDate)).days, axis=1)
    inpaitent_er_days = inpatient_df[['personId', 'Inpatient Days']]
    person_df = person_df.merge(inpaitent_er_days, how='left')

    person_df = person_df.fillna(0)

    person_df = person_df[['personId', '# of ER Visits (12M)', 'gender', 'Age', 'admitDate', 'Inpatient Days'
]]
    #Number of admissions is number of unique admit dates.
    person_df = person_df.rename(columns={'gender' : 'Gender', 'admitDate' : '# of Admissions (12M)' })

    #Cleaning the diagnosis codoes
    for column in diagnosis_columns:
        inpatient_df[column] = inpatient_df[column].apply(lambda x : cleanICD10Syntax(str(x)))
        outpatient_df[column] = outpatient_df[column].apply(lambda x : cleanICD10Syntax(str(x)))

    nodes = pd.read_csv(resource_filename('cv19index', 'resources/ccsrNodes.txt'))
    edges_df = pd.read_csv(resource_filename('cv19index', 'resources/ccsrEdges.txt'))
    edges_df['code'] = edges_df['child'].apply(lambda x: x.split(':')[1])

    #Generating features for each node
```

```
    for CCSR,description in nodes.values:
        # Getting the codes
        codes = edges_df[edges_df['parent'].str.contains(CCSR)]
        selected_inpatient = inpatient_df[inpatient_df.isin(codes['code'].values).any(axis=1)]['personId'].va
lues
        selected_outpatient = outpatient_df[outpatient_df.isin(codes['code'].values).any(axis=1)]['personId']
.values
        selected_personId = np.unique(np.concatenate((selected_inpatient,selected_outpatient)))

        #Assigning the diagnosis flag to ther person
        description = re.sub("[^\P{P}-/']+", "_", description.replace(")", ""))
        column_name = "Diagnosis of "+description+ " in the previous 12 months"
        person_df[column_name] = person_df['personId'].apply(lambda x : True if x in selected_personId else F
alse)



    #Getting the column order for the model
    f = open(resource_filename("cv19index", "resources/xgboost/input.csv.schema.json"))
    column_order = [item['name'] for item in json.load(f)['schema']]
    f.close()

    #returning the needed features.
    return person_df[column_order]
```

Generate the result dataframe that will be the input for the model

In [29]:
```
asOfDate = '2018-06-01'
diagnosis_cols = [ 'dx1','dx2', 'dx3', 'dx4', 'dx5', 'dx6', 'dx7', 'dx8', 'dx9', 'dx10', 'dx11', 'dx12', 'dx13', 'dx14', 'dx15', 'dxE1']


result = getTestDataFrame(person_df, eligibility_df, inpatient_df, outpatient_df, asOfDate, diagnosis_columns=diagnosis_cols)
result = result.reset_index().drop(columns=['index'])
result.head()
```

Out[29]:

	personId	# of ER Visits (12M)	Gender	Age	Diagnosis of Certain infectious and parasitic diseases in the previous 12 months	Diagnosis of Neoplasms in the previous 12 months	Diagnosis of Diseases of the blood and blood-forming organs and certain disorders involving the immune mechanism in the previous 12 months	Diagnosis of Endocrine_ nutritional and metabolic diseases in the previous 12 months	Diagnosis of Mental_ behavioral and neurodevelopmental disorders in the previous 12 months	Diagnosis of Diseases of the nervous system in the previous 12 months	...	Diagnc Nerv sys signs sympto in previ 12 mor
0	772775338f7ee353	0.0	1	73	False	False	False	False	False	False	...	F
1	d45d10ed2ec861c4	0.0	1	89	False	False	False	False	False	False	...	F
2	590bda01eeb795ee	0.0	1	71	False	False	False	False	False	False	...	F
3	14ad855b9fc39501	0.0	2	68	True	False	False	True	True	True	...	F
4	48b8f6c7a0435491	0.0	2	67	False	False	False	False	False	False	...	F

5 rows × 565 columns

In [38]:
```
output_name = "xgboost/example_input.csv"
result.to_csv(output_name, index=False, float_format="%f")
```

Running the CV19 Index Predictor

The cv19index.predict `do_run` function is used to generate predictions. This function takes 4 parameters:

1. input_file
2. input_schema (Default schema is available in the resource)
3. model (Path to the model pickle)
4. output_file

In [34]:
```
from cv19index.predict import do_run
```

In [33]:
```
input_fpath = "xgboost/example_input.csv"
input_schema = resource_filename("cv19index", "resources/xgboost/input.csv.schema.json")
model = resource_filename("cv19index", "resources/xgboost/model.pickle")
# model = resource_filename("cv19index", "resources/model_simple/model.pickle")
output = "xgboost/example_prediction.csv"

do_run(input_fpath, input_schema, model, output)
```

```
Computing SHAP scores.  Approximate = False
SHAP values completed
```

This function writes out a CSV file that contains the predictions along with a rationale for each predicted value.

The `risk score` column contains an integer from 1 to 100 which indicates the individual's risk relative to the overall Medicare beneficiary population. A risk of 100 is the highest, and a risk of 50 means the person has the median risk within the population.

The `prediction` column contains the person's predicted probability of having the proxy outcome.

The `pos_factors` and `neg_factors` columns contain information about different factors for this person the increase or decrease, respectively, the person's risk.

In [37]:
```
output_df = pd.read_csv(output)
output_df.head()
```

Out[37]:

	personId	prediction	risk_score	pos_factors	pos_shap_scores	pos_patient_values	neg_factors	neg_shap_scores	neg_pa
0	001ef63fe5cb0cc5	0.002882	84	['Age', 'Diagnosis of Heart failure in the pre...	[0.475, 0.442, 0.158, 0.089, 0.046, 0.042, 0.0...	[84.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0,...	['Diagnosis of Respiratory signs and symptoms ...	[-0.225, -0.17, -0.129, -0.097, -0.078, -0.043...	[Fal False,
1	00669248edd53308	0.000343	10	['Diagnosis of Neoplasm-related encounters in ...	[0.053, 0.024, 0.021]	[False, False, False]	['Age', 'Diagnosis of Respiratory signs and sy...	[-0.401, -0.237, -0.192, -0.146, -0.135, -0.12...	[69.0, 0.0,
2	00cf64b1fb5d4463	0.004796	92	['Age', 'Diagnosis of Heart failure in the pre...	[0.916, 0.404, 0.289, 0.136, 0.06, 0.055, 0.03...	[89.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0]	['Diagnosis of Respiratory signs and symptoms ...	[-0.198, -0.171, -0.128, -0.111, -0.091, -0.05...	[Fal False
3	015e674b3c39f428	0.001446	69	['Age', 'Diagnosis of Diseases of the circulat...	[0.548, 0.11, 0.055, 0.049, 0.039, 0.031, 0.02...	[85.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0, 0.0]	['Diagnosis of Respiratory signs and symptoms ...	[-0.211, -0.16, -0.127, -0.102, -0.093, -0.069...	[Fals True,
4	019a142f46a5bc3a	0.000579	39	['Age', 'Diagnosis of Neoplasm-related encount...	[0.068, 0.057, 0.026]	[79.0, 0.0, 0.0]	['Diagnosis of Respiratory signs and symptoms ...	[-0.267, -0.178, -0.135, -0.131, -0.116, -0.07...	[Fals False,