# Detection of Medical Frauds in the Health Care Sector

18-11-2022

—

Sujay Kumar - 20BDS0294

**Submitted to:** Prof Nalini A

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

## Overview

Medical Frauds consists of many types but most commonly there are three categories in which most of the frauds happens which is Fraud Committed by the HealthCare Provider, Fraud Committed by Patients/individuals and Fraud involving prescriptions. Metric Lab companies owners pays 5.7\$ to settle the medical fraud claims which was on July 22 2022 many of the recent findings like this comes under the frauds committed by the healthcare provider.

## Introduction

For solving the fraud problems in health care, the commonalities of fraud should be eliminated by the previous concerns raised by the FBI(Federal Bureau of Investigation) and FDA(Food and Drug Administration). The common concerns are ...

## Aims and Objectives

1. Billing for the services are not rendered, more precisely, buying a drug from the medical store, the worker needs to render bills even if we buy a small product everytime and this is important because without rendering a product for the customers the tag comes as "Fraud committed by the healthcare-provider (drug negligence - pharmaceutical)".

2. In detailed tests done in the lab should be recorded including minor test in the lab. This part more inclined towards the pharmaceuticals and product providers of health care sector, with this the tag comes as

"Fraud Committed by the Health care provider (No tests has been done before suggesting it to the patient)"

3. Misrepresentation of services like for health care testing the provided services is based on the particular but most of the hospitals does the genetic testing for their research. In the One service has been manipulated for an another, with this the tag comes as "Fraud committed by the HealthCare Provider (Misrepresentation of service)"

4. For building a model we should be concentrating more on the difference between accidental and intensional and the model should first cluster itself to which branch of the health care sector is given.

5. Deploy the Machine Learning model into the servers to detect Fraud.

## Proposed Methodology

In the United States, advances in technology and medical sciences continue to improve the general well-being of the population. With this continued progress, programs such as Medicare are needed to help manage the high costs associated with quality healthcare. Unfortunately, there are individuals who commit fraud for nefari- ous reasons and personal gain, limiting Medicare's ability to effectively provide for the healthcare needs of the elderly and other qualifying people. To minimize fraudulent activities, the Centers for Medicare and Medicaid Services (CMS) released a number of "Big Data" datasets for different parts of the Medicare program. In this paper, we focus on the detection of Medicare fraud using the following CMS datasets: (1) Medicare Provider Utilization and Payment Data: Physician and Other Supplier (Part B), (2) Medicare Provider Utilization and Payment Data: Part D Prescriber (Part D), and (3) Medicare Provider Utilization and Payment Data: Referring Durable Medical Equipment, Prosthetics, Orthotics and Supplies (DMEPOS). Additionally, we create a fourth dataset which is a combination of the three primary datasets. We discuss data processing for all four datasets and the mapping of real-world provider fraud labels using the List of Excluded Individuals and Entities (LEIE) from the Office of the Inspector General. Our exploratory analysis on Medicare fraud detection involves building and assessing three learners on each dataset. Based on the Area under the Receiver Operating Characteris- tic (ROC) Curve performance metric, our results show that the Combined dataset with the Logistic Regression (LR) learner yielded the best overall score at 0.816, closely fol- lowed by the Part B dataset with LR at 0.805. Overall, the Combined and Part B datasets produced the best fraud detection performance with no statistical difference between these datasets, over all the learners. Therefore, based on our results and the assumption that there is no way to know within which part of Medicare a physician will commit fraud, we suggest using the Combined dataset for detecting fraudulent behavior when a physician has submitted payments through any or all Medicare parts evaluated in our study.

## Dataset analysis

In this section, we describe the CMS datasets we use (Part B, Part D and, DMEPOS). Furthermore, the data processing methodology used to create each dataset, including processing, fraud label mapping between the Medicare datasets and the LEIE, and one- hot encoding for categorical variables is discussed. The information within each data- set is based on CMS's administrative claims data for Medicare beneficiaries enrolled in the Fee-For-Service program. Note, this data does not take into account any claims submitted through the Medicare Advantage program. Since CMS records all claims information after payments are made , we assume the Medicare data is already cleansed and is correct. Note that NPI is not used in the data mining step, but rather for aggregation and identification. Additionally, for each dataset, we added a year variable which is also used for aggregation and identification.

### *Part B*

The Part B dataset provides claims information for each procedure a physician performs within a given year. Currently, this dataset is available on the CMS website for the 2012 through 2015 calendar years (with 2015 being released in 2017). Physicians are identified using their unique NPI whil,e procedures are labeled by their Healthcare Common Procedure Coding System (HCPCS) code . Other claims information includes average payments and charges, the number of procedures performed and medical specialty (also known as provider type). CMS decided to aggregate Part B data over: (1) NPI of the performing provider, (2) HCPCS code for the procedure or service performed, and (3) the place of service which is either a facility (F) or non-facility (O), such as a hospital or office, respectively. Each row, in the dataset, includes a physician's NPI, provider type, one HCPCS code split by place of service along with specific information corresponding to this breakdown (i.e. claim counts) and other non-changing attributes (i.e. gender). We have found that in practice, physicians perform the same procedure (HCPCS code) at both a facility and their office, as well as a few physicians that practice under multiple provider types (specialties) such as Internal Medicine and Cardiology. Therefore, for each physician, there are as many rows as unique combinations of NPI, Provider Type, HCPCS code and place of service and thus Part B data can be considered to provide procedure-level information.

## *Part D*

The Part D dataset provides information pertaining to the prescription drugs they administer under the Medicare Part D Prescription Drug Program within a given year. Currently, this data is available on the CMS website for the 2013 through 2015 calendar years (with 2015 being released in 2017) . Physicians are identified using their unique NPI within the data while each drug is labeled by their brand and generic name. Other information includes average payments and charges, variables describing the drug quan- tity prescribed and medical specialty. CMS decided to aggregate the Part D data over: (1) the NPI of the prescriber, and (2) the drug name (brand name in the case of trademarked drugs) and generic name. Each row in the Part D dataset lists a physician's NPI, provider type and drug name along with specific information corresponding to this breakdown (i.e. claim counts) and other static attributes (i.e. gender). Same as with Part B, we found a few physicians that practice under multiple specialties, such as Internal Medicine and Cardiology. Therefore, for each physician, there are as many rows as unique combinations of NPI, Provider Type, drug name and generic name and thus, Part D data can be considered to provide procedure-level information. In order to protect the privacy of Medicare beneficiaries, any aggregated records, derived from 10 or fewer claims, are excluded from the Part D data.

## *DMEPOS*

The DMEPOS dataset provides claims information about Medical Equipment, Pros- thetics, Orthotics and Supplies that physicians referred patients to either purchase or rent from a supplier within a given year. Note, this dataset is based on supplier's claims submitted to Medicare while the physician's role is referring the patient to the supplier. Currently this data is available on the CMS website for 2013 through 2015 calendar years (with 2015 being released in 2017) . Physicians are identified using their unique NPI within the data while products are labeled by their HCPCS code. Other claims information includes average payments and charges, the num- ber of services/products rented or sold and medical specialty (also known as pro- vider type). CMS decided to aggregate Part B data over: (1) NPI of the performing provider, (2) HCPCS code for the procedure or service performed by the DMEPOS supplier, and (3) the supplier rental indicator (value of either 'Y' or 'N') derived from DMEPOS supplier claims (according to CMS documentation). Each row provides a physician's NPI, provider type, one HCPCS code split by rental or non-rental with specific information corresponding to this breakdown (i.e. number of supplier claims) and other non-changing attributes (i.e. gender). We have found that some physicians place referrals for the same DMEPOS equipment, or HCPCS code, as both rental and non-rental as well as a few physicians that practice under multiple specialties such as Internal Medicine and Cardiology. Therefore, for each physician, there are as many rows as unique combinations of NPI, Provider Type, HCPCS code and rental status, and thus the DMEPOS data also can be considered to provide procedure-level information.

# PROPOSED WORK

1. Rndrng_Prvdr_Last_Org_Name - Organization's last name
2. Rndrng_Prvdr_First_Name - Providers First name
3. Rndrng_Prvdr_MI - Providers Middle name (a.k.a )
4. Rndrng_Prvdr_Crdntls - Credentials of the Provider
5. Rndrng_Prvdr_Gndr - Gender of the Provider
6. Rndrng_Prvdr_Ent_Cd - as an individual or an Organization

7. Rndrng_Prvdr_State_FIPS - To indicate which state does this lies on in a integer representation

https://www.bls.gov/respondents/mwr/electronic-data-interchange/appendix-d-usps-state-abbreviations-and-fips-codes.html

8. Rndrng_Prvdr_St1 - Street Address 1
9. Rndrng_Prvdr_St2 - Street Address 2
10. Rndrng_Prvdr_City - City
11. Rndrng_Prvdr_State_Abrvtn - State
12. Rndrng_Prvdr_Zip5 - Zip Code
13. Rndrng_Prvdr_RUCA - Rural Urban Communicating Area Codes

https://depts.washington.edu/uwruca/ruca-uses.php

14. Rndrng_Prvdr_RUCA_Desc - Description of RUCA

15. Rndrng_Prvdr_Cntry - The Provider Country (only US not balanced)

16. Rndrng_Prvdr_Type - The Provider Type
17. Rndrng_Prvdr_Mdcr_Prtcptg_Ind - The Provider Participation Indicator

18. Tot_HCPCS_Cds - Total HCPCS Codes for a particular provider
19. Tot_Benes - Total number of Beneficiaries provided by the Providers
20. Tot_Srvcs - TOtal number of Services provided by the Provider
21. Tot_Sbmtd_Chrg - Total Submitted Charges by the Provider
22. Tot_Mdcr_Alowd_Amt - Total ammount allowed by the medicare
23. Tot_Mdcr_Pymt_Amt - Ammount After deducting the insurance etc...
24. Tot_Mdcr_Stdzd_Amt - The ammount for which the patient paid

25. Drug_Sprsn_Ind - Supression Indicator (* -> supressed #-> Counter Suppressed)
26. Drug_Tot_HCPCS_Cds - Total no of HCPCS codes for a drug
27. Drug_Tot_Benes - Total medical beneficiaries with drug
28. Drug_Sbmtd_Chrg - The total charges submitted for drug services
29. Drug_Mdcr_Alowd_Amt -
30. Drug_Mdcr_Pymt_Amt
31. Drug_Mdcr_Stdzd_Amt

32. Med_Sprsn_Ind

33. Med_Tot_HCPCS_Cds - No of HCPCS codes applied for the drug

34. Med_Tot_Benes - Total Beneficiaries provided by the provider

35. Med_Sbmtd_Chrg

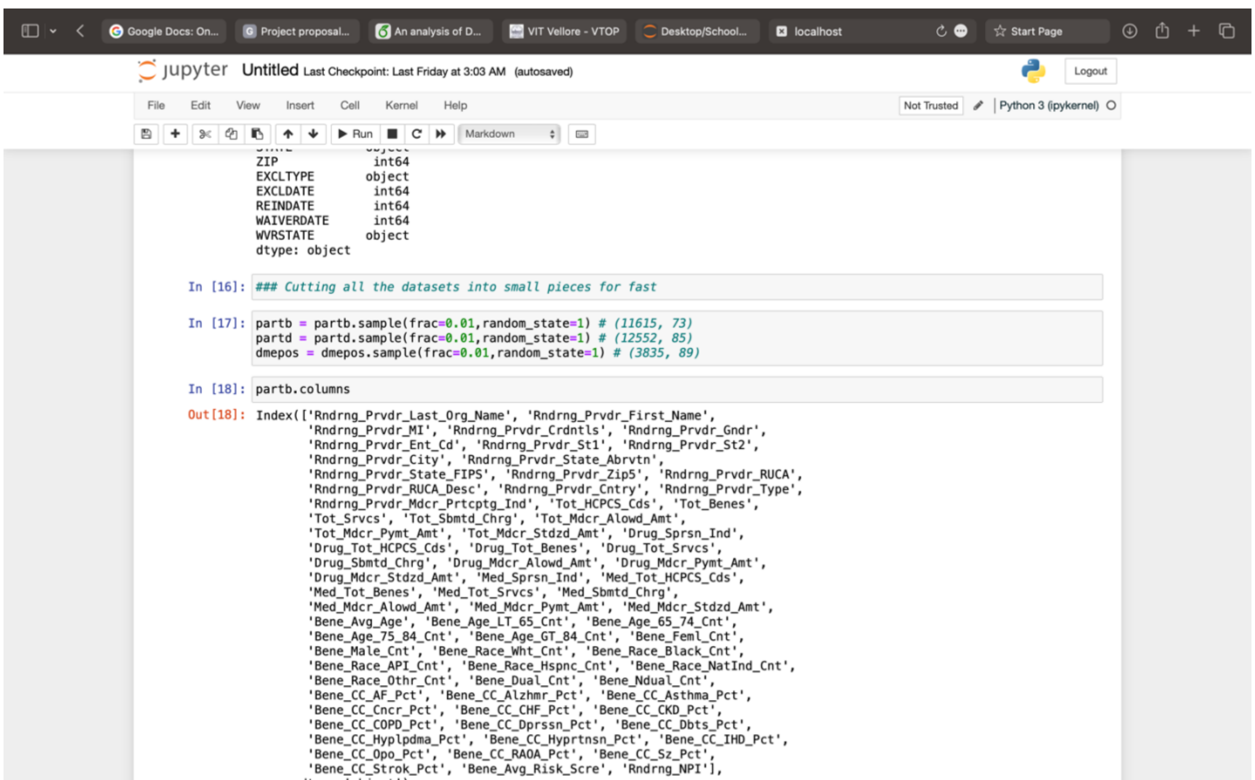
36. Med_Mdcr_Alowd_Amt

37. Med_Mdcr_Pymt_Amt

38. Med_Mdcr_Stdzd_Amt

# BENEFICIARIES

39. Bene_Dual_Cnt - No of medicaid beneficiaries qualified to receive medicare



# HEALTH CARE FRAUD DETECTION

```
In [126]: # Import Pandas
          import pandas as pd
          import numpy as np
```

## INCLUSION OF DATASETS

### MEDICARE PROVIDER UTILIZATION AND PAYMENT DATA

```
Note :
    # There are 3 Primary Datasets
        1. Part_B -> Physician and other supplies data
        2. Part_D -> Prescriber Data
        3. DMEPOS -> Durable Medical Equipment, Prosthetics, Orthotics and Supplies(DMEPOS)
```

#### Including "PART_D" Dataset

```
In [2]: partd = pd.read_csv("raw_data/Part-D.csv",encoding='ISO-8859-1')

In [3]: partd.shape
Out[3]: (1255175, 85)

In [4]: partd.dtypes
Out[4]: PRSCRBR_NPI              int64
        Prscrbr_Last_Org_Name   object
        Prscrbr_First_Name      object
        Prscrbr_MI              object
```

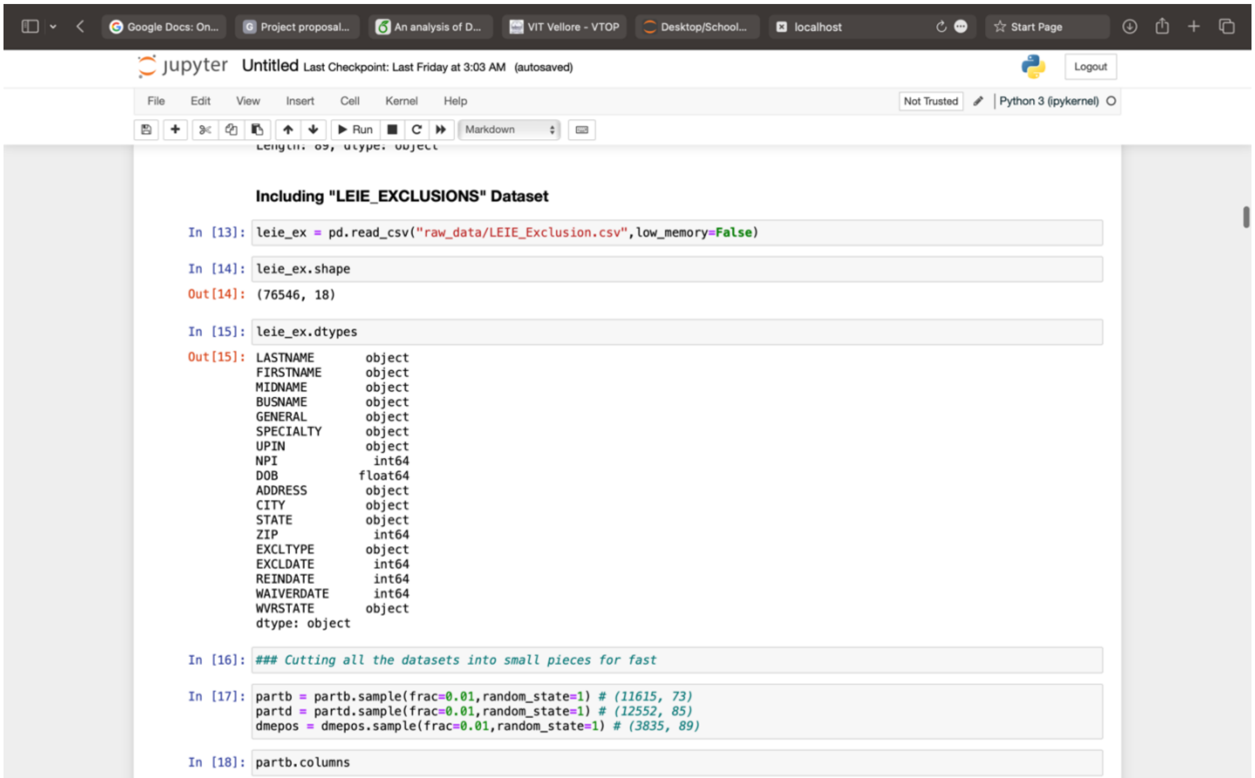
```
        Bene_Ndual_Cnt          float64
        Bene_Avg_Risk_Scre      float64
        Length: 85, dtype: object
```

#### Including "PART_B" Dataset

```
In [5]: partb = pd.read_csv("raw_data/Part-B.csv",encoding = "ISO-8859-1",low_memory=False)

In [6]: partb.shape
Out[6]: (1161542, 73)

In [7]: partb["Rndrng_NPI"] = partb["ï»¿Rndrng_NPI"]

In [8]: partb=partb.drop("ï»¿Rndrng_NPI",axis=1)

In [9]: partb.dtypes
Out[9]: Rndrng_Prvdr_Last_Org_Name     object
        Rndrng_Prvdr_First_Name        object
        Rndrng_Prvdr_MI                object
        Rndrng_Prvdr_Crdntls           object
        Rndrng_Prvdr_Gndr              object
                                        ...
        Bene_CC_RAOA_Pct               object
        Bene_CC_Sz_Pct                 object
        Bene_CC_Strok_Pct              object
        Bene_Avg_Risk_Scre            float64
        Rndrng_NPI                      int64
        Length: 73, dtype: object
```

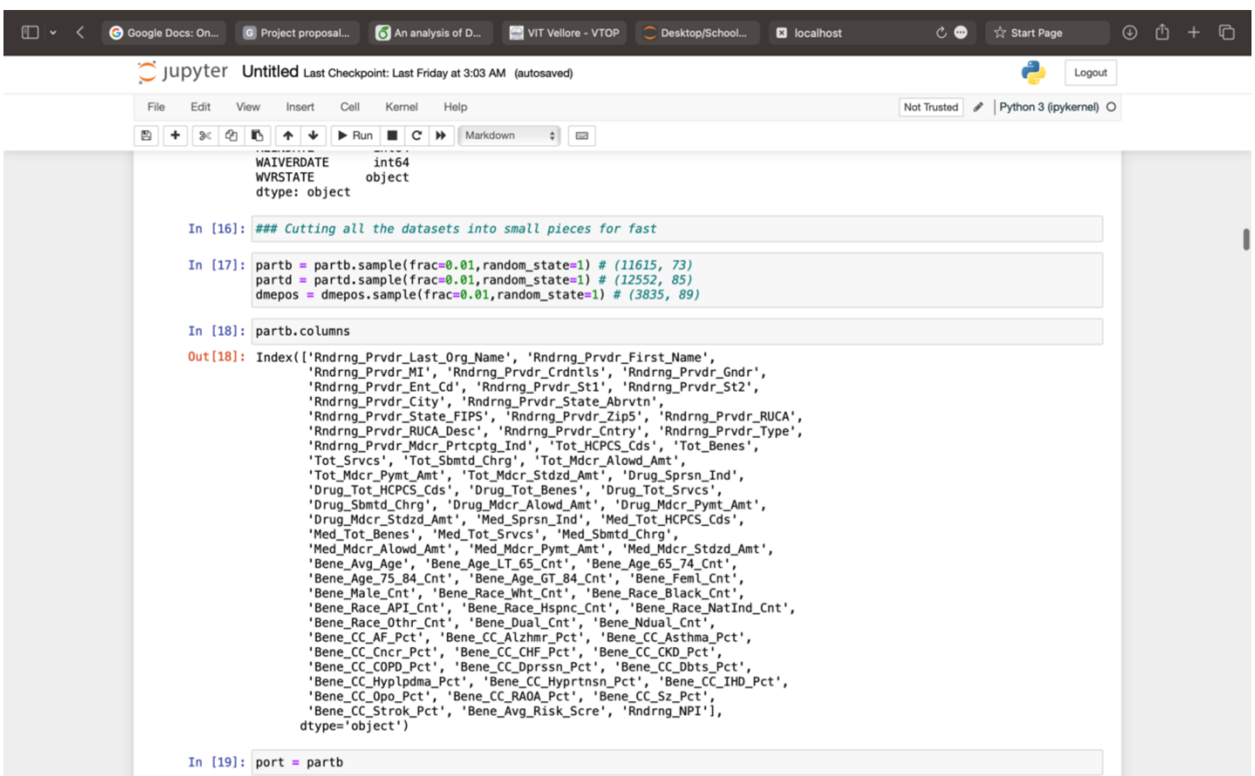
#### Including "DMEPOS" Dataset

```
In [10]: dmepos = pd.read_csv("raw_data/DMEPOS.csv",low_memory=False)

In [11]: dmepos.shape
Out[11]: (383488, 89)
```



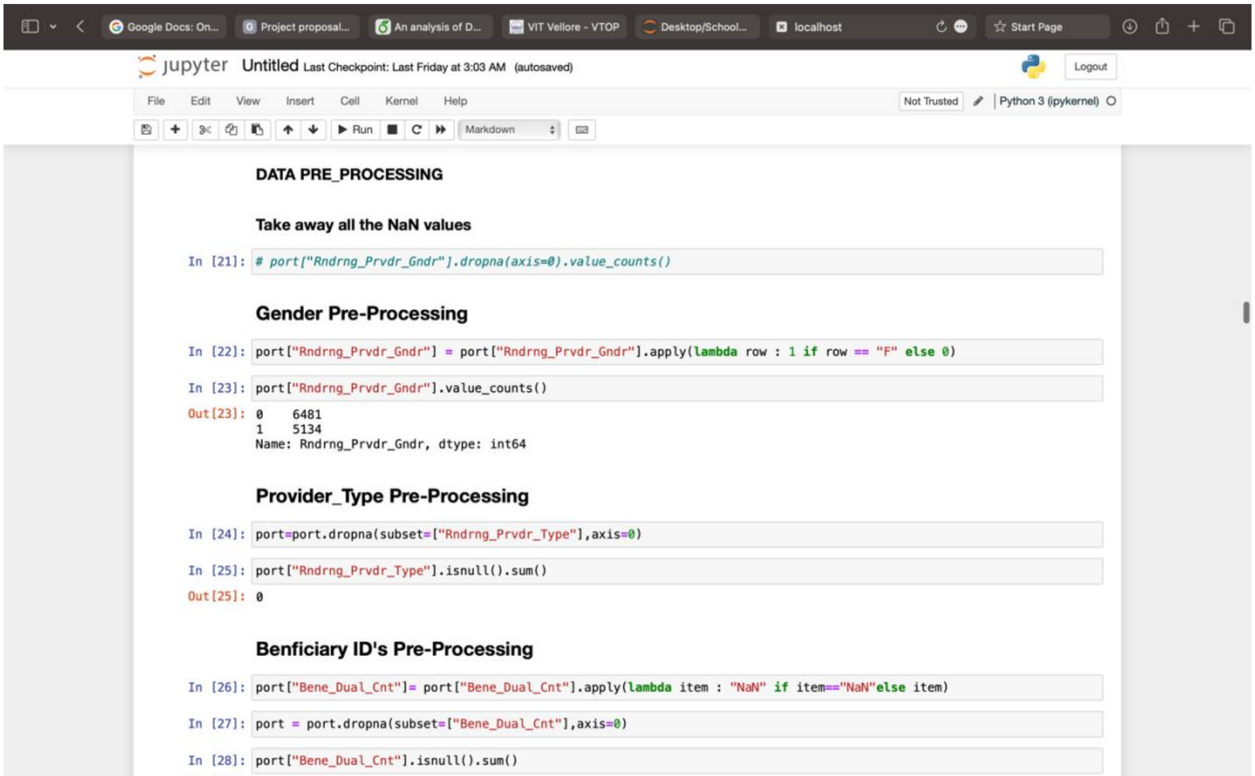
Including "LEIE_EXCLUSIONS" Dataset

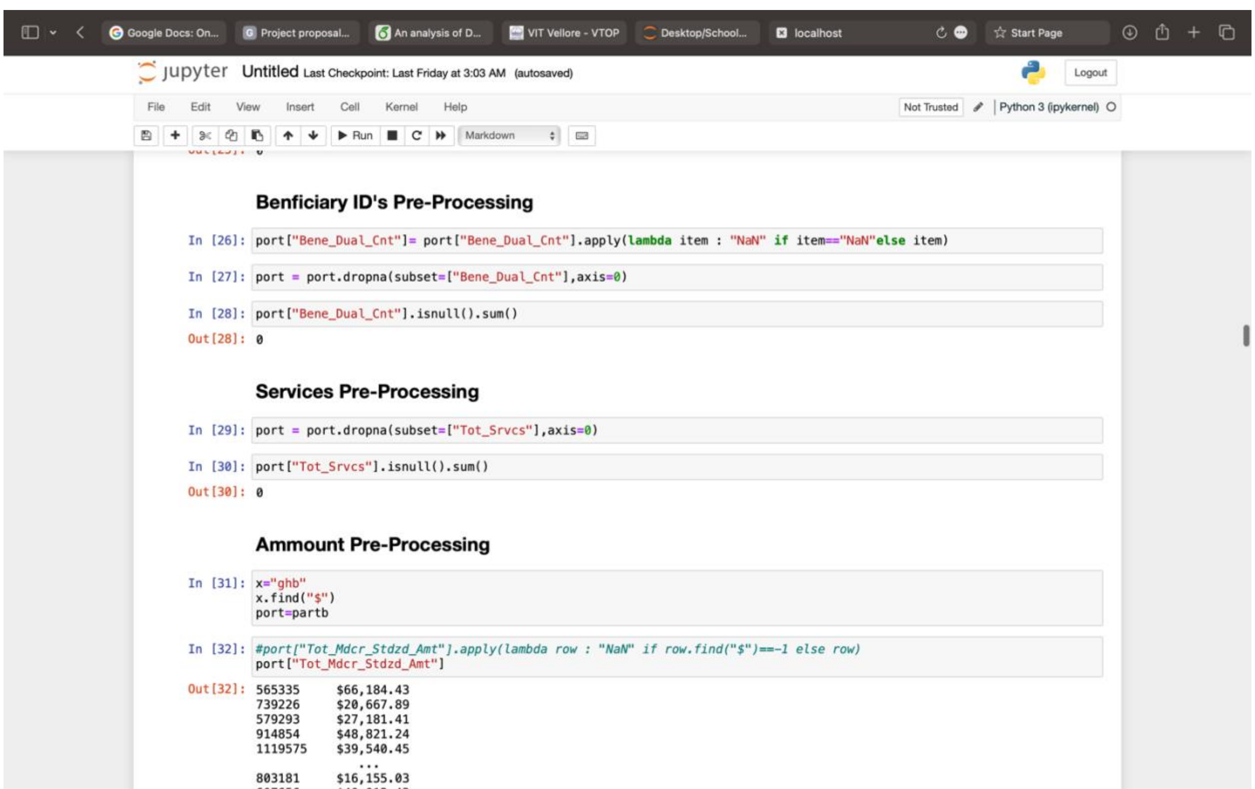
```
In [13]: leie_ex = pd.read_csv("raw_data/LEIE_Exclusion.csv",low_memory=False)

In [14]: leie_ex.shape
Out[14]: (76546, 18)

In [15]: leie_ex.dtypes
Out[15]: LASTNAME       object
         FIRSTNAME      object
         MIDNAME        object
         BUSNAME        object
         GENERAL        object
         SPECIALTY      object
         UPIN           object
         NPI             int64
         DOB           float64
         ADDRESS        object
         CITY           object
         STATE          object
         ZIP             int64
         EXCLTYPE       object
         EXCLDATE        int64
         REINDATE        int64
         WAIVERDATE      int64
         WVRSTATE       object
         dtype: object

In [16]: ### Cutting all the datasets into small pieces for fast

In [17]: partb = partb.sample(frac=0.01,random_state=1) # (11615, 73)
         partd = partd.sample(frac=0.01,random_state=1) # (12552, 85)
         dmepos = dmepos.sample(frac=0.01,random_state=1) # (3835, 89)

In [18]: partb.columns
```

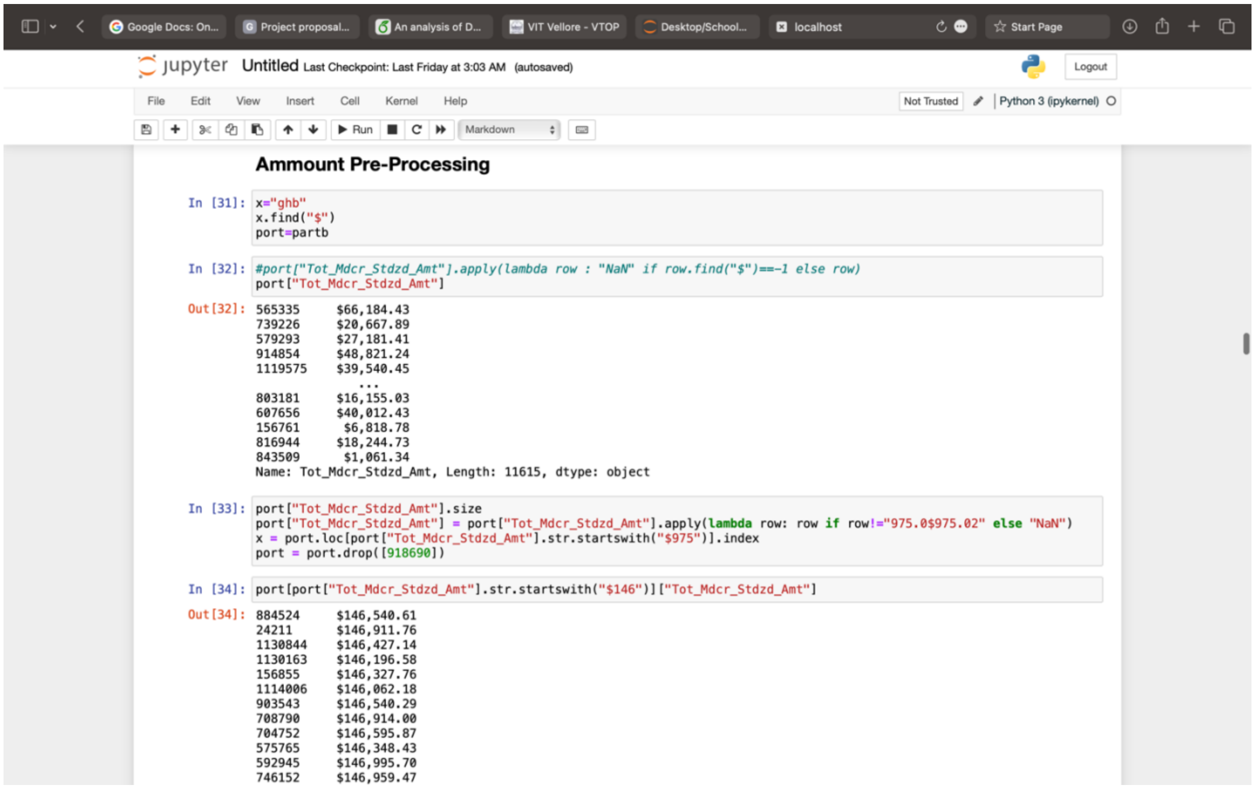



```
         WAIVERDATE      int64
         WVRSTATE       object
         dtype: object

In [16]: ### Cutting all the datasets into small pieces for fast

In [17]: partb = partb.sample(frac=0.01,random_state=1) # (11615, 73)
         partd = partd.sample(frac=0.01,random_state=1) # (12552, 85)
         dmepos = dmepos.sample(frac=0.01,random_state=1) # (3835, 89)

In [18]: partb.columns
```


```
Out[18]: Index(['Rndrng_Prvdr_Last_Org_Name', 'Rndrng_Prvdr_First_Name',
                'Rndrng_Prvdr_MI', 'Rndrng_Prvdr_Crdntls', 'Rndrng_Prvdr_Gndr',
                'Rndrng_Prvdr_Ent_Cd', 'Rndrng_Prvdr_St1', 'Rndrng_Prvdr_St2',
                'Rndrng_Prvdr_City', 'Rndrng_Prvdr_State_Abrvtn',
                'Rndrng_Prvdr_State_FIPS', 'Rndrng_Prvdr_Zip5', 'Rndrng_Prvdr_RUCA',
                'Rndrng_Prvdr_RUCA_Desc', 'Rndrng_Prvdr_Cntry', 'Rndrng_Prvdr_Type',
                'Rndrng_Prvdr_Mdcr_Prtcptg_Ind', 'Tot_HCPCS_Cds', 'Tot_Benes',
                'Tot_Srvcs', 'Tot_Sbmtd_Chrg', 'Tot_Mdcr_Alowd_Amt',
                'Tot_Mdcr_Pymt_Amt', 'Tot_Mdcr_Stdzd_Amt', 'Drug_Sprsn_Ind',
                'Drug_Tot_HCPCS_Cds', 'Drug_Tot_Benes', 'Drug_Tot_Srvcs',
                'Drug_Sbmtd_Chrg', 'Drug_Mdcr_Alowd_Amt', 'Drug_Mdcr_Pymt_Amt',
                'Drug_Mdcr_Stdzd_Amt', 'Med_Sprsn_Ind', 'Med_Tot_HCPCS_Cds',
                'Med_Tot_Benes', 'Med_Tot_Srvcs', 'Med_Sbmtd_Chrg',
                'Med_Mdcr_Alowd_Amt', 'Med_Mdcr_Pymt_Amt', 'Med_Mdcr_Stdzd_Amt',
                'Bene_Avg_Age', 'Bene_Age_LT_65_Cnt', 'Bene_Age_65_74_Cnt',
                'Bene_Age_75_84_Cnt', 'Bene_Age_GT_84_Cnt', 'Bene_Feml_Cnt',
                'Bene_Male_Cnt', 'Bene_Race_Wht_Cnt', 'Bene_Race_Black_Cnt',
                'Bene_Race_API_Cnt', 'Bene_Race_Hspnc_Cnt', 'Bene_Race_NatInd_Cnt',
                'Bene_Race_Othr_Cnt', 'Bene_Dual_Cnt', 'Bene_Ndual_Cnt',
                'Bene_CC_AF_Pct', 'Bene_CC_Alzhmr_Pct', 'Bene_CC_Asthma_Pct',
                'Bene_CC_Cncr_Pct', 'Bene_CC_CHF_Pct', 'Bene_CC_CKD_Pct',
                'Bene_CC_COPD_Pct', 'Bene_CC_Dprssn_Pct', 'Bene_CC_Dbts_Pct',
                'Bene_CC_Hyplpdma_Pct', 'Bene_CC_Hyprtnsn_Pct', 'Bene_CC_IHD_Pct',
                'Bene_CC_Opo_Pct', 'Bene_CC_RAOA_Pct', 'Bene_CC_Sz_Pct',
                'Bene_CC_Strok_Pct', 'Bene_Avg_Risk_Scre', 'Rndrng_NPI'],
               dtype='object')

In [19]: port = partb
```


Google Docs: On...
Project proposal...
An analysis of D...
VIT Vellore - VTOP
Desktop/School...
localhost
Start Page
jupyter Untitled Last Checkpoint: Last Friday at 3:03 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Markdown
DATA PRE_PROCESSING
Take away all the NaN values
In [21]: # port["Rndrng_Prvdr_Gndr"].dropna(axis=0).value_counts()
Gender Pre-Processing
In [22]: port["Rndrng_Prvdr_Gndr"] = port["Rndrng_Prvdr_Gndr"].apply(lambda row : 1 if row == "F" else 0)
In [23]: port["Rndrng_Prvdr_Gndr"].value_counts()
Out[23]: 0 6481
1 5134
Name: Rndrng_Prvdr_Gndr, dtype: int64
Provider_Type Pre-Processing
In [24]: port=port.dropna(subset=["Rndrng_Prvdr_Type"],axis=0)
In [25]: port["Rndrng_Prvdr_Type"].isnull().sum()
Out[25]: 0
Benficiary ID's Pre-Processing
In [26]: port["Bene_Dual_Cnt"]= port["Bene_Dual_Cnt"].apply(lambda item : "NaN" if item=="NaN"else item)
In [27]: port = port.dropna(subset=["Bene_Dual_Cnt"],axis=0)
In [28]: port["Bene_Dual_Cnt"].isnull().sum()


Google Docs: On...
Project proposal...
An analysis of D...
VIT Vellore - VTOP
Desktop/School...
localhost
Start Page
jupyter Untitled Last Checkpoint: Last Friday at 3:03 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Markdown
Benficiary ID's Pre-Processing
In [26]: port["Bene_Dual_Cnt"]= port["Bene_Dual_Cnt"].apply(lambda item : "NaN" if item=="NaN"else item)
In [27]: port = port.dropna(subset=["Bene_Dual_Cnt"],axis=0)
In [28]: port["Bene_Dual_Cnt"].isnull().sum()
Out[28]: 0
Services Pre-Processing
In [29]: port = port.dropna(subset=["Tot_Srvcs"],axis=0)
In [30]: port["Tot_Srvcs"].isnull().sum()
Out[30]: 0
Ammount Pre-Processing
In [31]: x="ghb"
x.find("$")
port=partb
In [32]: #port["Tot_Mdcr_Stdzd_Amt"].apply(lambda row : "NaN" if row.find("$")==-1 else row)
port["Tot_Mdcr_Stdzd_Amt"]
Out[32]: 565335 $66,184.43
739226 $20,667.89
579293 $27,181.41
914854 $48,821.24
1119575 $39,540.45
...
803181 $16,155.03



jupyter Untitled Last Checkpoint: Last Friday at 3:03 AM (autosaved)

### Ammount Pre-Processing

In [31]:
```
x="ghb"
x.find("$")
port=partb
```

In [32]:
```
#port["Tot_Mdcr_Stdzd_Amt"].apply(lambda row : "NaN" if row.find("$")==-1 else row)
port["Tot_Mdcr_Stdzd_Amt"]
```

Out[32]:
```
565335     $66,184.43
739226     $20,667.89
579293     $27,181.41
914854     $48,821.24
1119575    $39,540.45
              ...
803181     $16,155.03
607656     $40,012.43
156761      $6,818.78
816944     $18,244.73
843509      $1,061.34
Name: Tot_Mdcr_Stdzd_Amt, Length: 11615, dtype: object
```

In [33]:
```
port["Tot_Mdcr_Stdzd_Amt"].size
port["Tot_Mdcr_Stdzd_Amt"] = port["Tot_Mdcr_Stdzd_Amt"].apply(lambda row: row if row!="975.0$975.02" else "NaN")
x = port.loc[port["Tot_Mdcr_Stdzd_Amt"].str.startswith("$975")].index
port = port.drop([918690])
```

In [34]:
```
port[port["Tot_Mdcr_Stdzd_Amt"].str.startswith("$146")]["Tot_Mdcr_Stdzd_Amt"]
```

Out[34]:
```
884524     $146,540.61
24211      $146,911.76
1130844    $146,427.14
1130163    $146,196.58
156855     $146,327.76
1114006    $146,062.18
903543     $146,540.29
708790     $146,914.00
704752     $146,595.87
575765     $146,348.43
592945     $146,995.70
746152     $146,959.47
```

jupyter Untitled Last Checkpoint: Last Friday at 3:03 AM (autosaved)

File Edit View Insert Cell Kernel Help — Not Trusted — Python 3 (ipykernel)

```
In [36]: port = port.dropna(subset=["Tot_Mdcr_Stdzd_Amt"],axis=0)

In [37]: port["Tot_Mdcr_Stdzd_Amt"].isnull().sum()
Out[37]: 0
```

**NPI Pre-Processing**

```
In [38]: port = port.dropna(subset=["Rndrng_NPI"],axis=0)

In [39]: port["Rndrng_NPI"].isnull().sum()
Out[39]: 0

In [40]: part = port[["Rndrng_NPI","Rndrng_Prvdr_Type","Tot_Mdcr_Stdzd_Amt","Tot_Srvcs","Bene_Dual_Cnt","Rndrng_Prvdr_Gndr"]]

In [41]: # Bene_Dual_Cnt
part["Bene_Dual_Cnt"]= part["Bene_Dual_Cnt"].apply(lambda item : "NaN" if item=="NaN"else item)
part = part.dropna(subset=["Bene_Dual_Cnt"],axis=0)
part["Bene_Dual_Cnt"]=part["Bene_Dual_Cnt"].apply(lambda item : float(item.replace(",","")) if item.find(",")!=-1 el
part["Bene_Dual_Cnt"].astype(int)

/var/folders/l1/rp1rrpyx24d84x0k_6p7pwgw0000gn/T/ipykernel_94088/112612399.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  part["Bene_Dual_Cnt"]= part["Bene_Dual_Cnt"].apply(lambda item : "NaN" if item=="NaN"else item)

Out[41]: 565335     170
739226      61
1119575     86
730182      96
1028876     14
           ...
170008      91
141580      21
```

jupyter Untitled Last Checkpoint: Last Friday at 3:03 AM (autosaved)

File Edit View Insert Cell Kernel Help — Not Trusted — Python 3 (ipykernel)

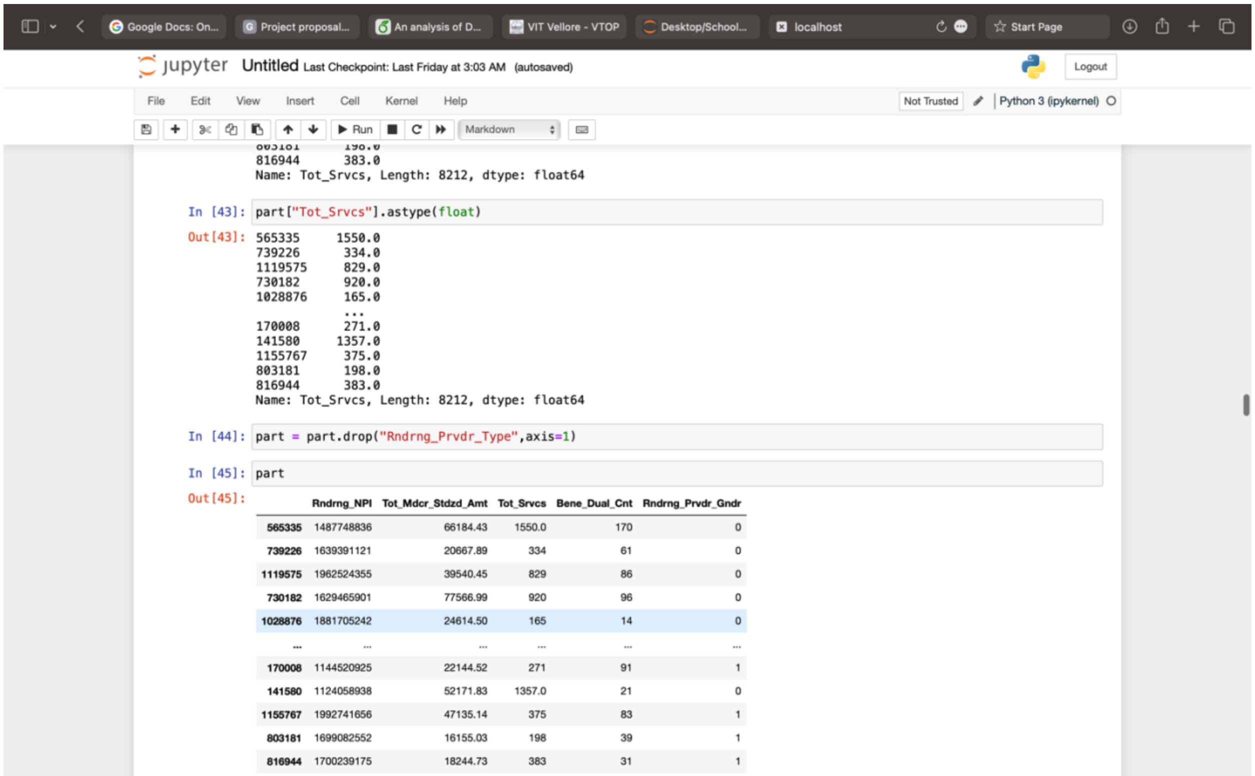
```
In [ ]:

In [ ]:

In [ ]:

In [46]: leie_ex["NPI"] = leie_ex["NPI"].apply(lambda row :  row if row!=0 else 0)

In [47]: exc = leie_ex[leie_ex["NPI"]!=0]["NPI"]
```

**Adding the Exclusion Attribute in the PART-B**

```
In [48]: partb["Rndrng_Prvdr_Last_Org_Name"]

Out[48]: 565335     Schoenkerman
739226             Bhat
579293          Goodwin
914854            Bello
1119575           Werle
               ...
803181            Reese
607656          Gersava
156761      Farnsworth
816944           Lorenz
843509          Mckenna
Name: Rndrng_Prvdr_Last_Org_Name, Length: 11615, dtype: object

In [49]: part
```

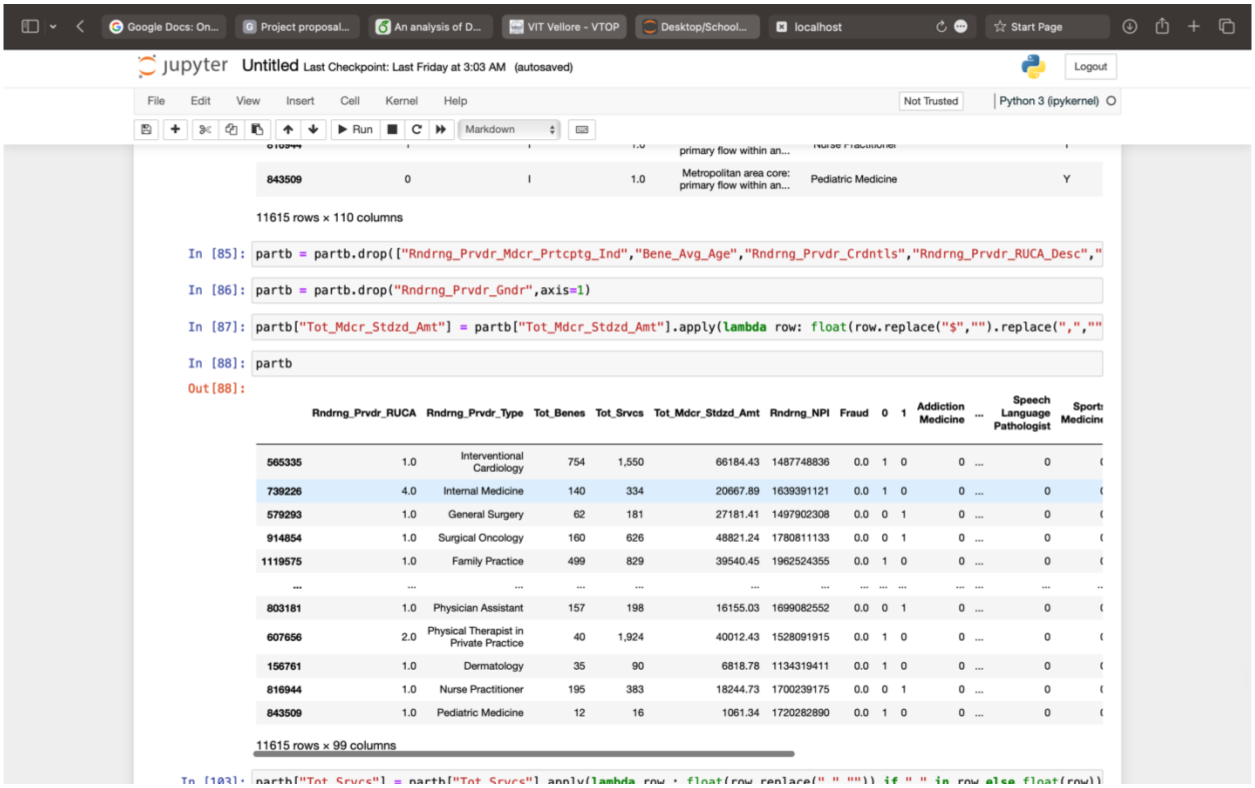
Out[49]:

| | Rndrng_NPI | Tot_Mdcr_Stdzd_Amt | Tot_Srvcs | Bene_Dual_Cnt | Rndrng_Prvdr_Gndr |
|---|---|---|---|---|---|
| **565335** | 1487748836 | 66184.43 | 1550.0 | 170 | 0 |
| **739226** | 1639391121 | 20667.89 | 334 | 61 | 0 |
| **1119575** | 1962524355 | 39540.45 | 829 | 86 | 0 |
| **730182** | 1629465901 | 77566.99 | 920 | 96 | 0 |
| **1028876** | 1881705242 | 24614.50 | 165 | 14 | 0 |



```
803181      198.0
816944      383.0
Name: Tot_Srvcs, Length: 8212, dtype: float64

In [43]: part["Tot_Srvcs"].astype(float)

Out[43]: 565335     1550.0
         739226      334.0
         1119575     829.0
         730182      920.0
         1028876     165.0
                     ...
         170008      271.0
         141580     1357.0
         1155767     375.0
         803181      198.0
         816944      383.0
         Name: Tot_Srvcs, Length: 8212, dtype: float64

In [44]: part = part.drop("Rndrng_Prvdr_Type",axis=1)

In [45]: part

Out[45]:
```

| | Rndrng_NPI | Tot_Mdcr_Stdzd_Amt | Tot_Srvcs | Bene_Dual_Cnt | Rndrng_Prvdr_Gndr |
|---|---|---|---|---|---|
| **565335** | 1487748836 | 66184.43 | 1550.0 | 170 | 0 |
| **739226** | 1639391121 | 20667.89 | 334 | 61 | 0 |
| **1119575** | 1962524355 | 39540.45 | 829 | 86 | 0 |
| **730182** | 1629465901 | 77566.99 | 920 | 96 | 0 |
| **1028876** | 1881705242 | 24614.50 | 165 | 14 | 0 |
| **...** | ... | ... | ... | ... | ... |
| **170008** | 1144520925 | 22144.52 | 271 | 91 | 1 |
| **141580** | 1124058938 | 52171.83 | 1357.0 | 21 | 0 |
| **1155767** | 1992741656 | 47135.14 | 375 | 83 | 1 |
| **803181** | 1699082552 | 16155.03 | 198 | 39 | 1 |
| **816944** | 1700239175 | 18244.73 | 383 | 31 | 1 |



jupyter Untitled Last Checkpoint: Last Friday at 3:03 AM (autosaved)

File Edit View Insert Cell Kernel Help Not Trusted Python 3 (ipykernel)

| | | | | | | |
|---|---|---|---|---|---|---|
| 843509 | 0 | I | 1.0 | Metropolitan area core: primary flow within an... | Pediatric Medicine | Y |

11615 rows × 110 columns

```
In [85]: partb = partb.drop(["Rndrng_Prvdr_Mdcr_Prtcptg_Ind","Bene_Avg_Age","Rndrng_Prvdr_Crdntls","Rndrng_Prvdr_RUCA_Desc","
In [86]: partb = partb.drop("Rndrng_Prvdr_Gndr",axis=1)
In [87]: partb["Tot_Mdcr_Stdzd_Amt"] = partb["Tot_Mdcr_Stdzd_Amt"].apply(lambda row: float(row.replace("$","").replace(",",""
In [88]: partb
```

Out[88]:

| | Rndrng_Prvdr_RUCA | Rndrng_Prvdr_Type | Tot_Benes | Tot_Srvcs | Tot_Mdcr_Stdzd_Amt | Rndrng_NPI | Fraud | 0 | 1 | Addiction Medicine | ... | Speech Language Pathologist | Sport Medicin |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 565335 | 1.0 | Interventional Cardiology | 754 | 1,550 | 66184.43 | 1487748836 | 0.0 | 1 | 0 | 0 | ... | 0 | ( |
| 739226 | 4.0 | Internal Medicine | 140 | 334 | 20667.89 | 1639391121 | 0.0 | 1 | 0 | 0 | ... | 0 | ( |
| 579293 | 1.0 | General Surgery | 62 | 181 | 27181.41 | 1497902308 | 0.0 | 0 | 1 | 0 | ... | 0 | ( |
| 914854 | 1.0 | Surgical Oncology | 160 | 626 | 48821.24 | 1780811133 | 0.0 | 0 | 1 | 0 | ... | 0 | ( |
| 1119575 | 1.0 | Family Practice | 499 | 829 | 39540.45 | 1962524355 | 0.0 | 1 | 0 | 0 | ... | 0 | ( |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 803181 | 1.0 | Physician Assistant | 157 | 198 | 16155.03 | 1699082552 | 0.0 | 0 | 1 | 0 | ... | 0 | ( |
| 607656 | 2.0 | Physical Therapist in Private Practice | 40 | 1,924 | 40012.43 | 1528091915 | 0.0 | 1 | 0 | 0 | ... | 0 | ( |
| 156761 | 1.0 | Dermatology | 35 | 90 | 6818.78 | 1134319411 | 0.0 | 1 | 0 | 0 | ... | 0 | ( |
| 816944 | 1.0 | Nurse Practitioner | 195 | 383 | 18244.73 | 1700239175 | 0.0 | 0 | 1 | 0 | ... | 0 | ( |
| 843509 | 1.0 | Pediatric Medicine | 12 | 16 | 1061.34 | 1720282890 | 0.0 | 1 | 0 | 0 | ... | 0 | ( |

11615 rows × 99 columns

```
In [103]: partb["Tot_Srvcs"] = partb["Tot_Srvcs"].apply(lambda row : float(row.replace(",","")) if "," in row else float(row))
```

jupyter Untitled Last Checkpoint: Last Friday at 3:03 AM (autosaved)

```
843509    1.0    Pediatric Medicine    12    16    1061.34    1720282890    0.0    1    0    0 ...    0

11615 rows × 99 columns
```

```
In [103]: partb["Tot_Srvcs"] = partb["Tot_Srvcs"].apply(lambda row : float(row.replace(",","")) if "," in row else float(row))
```

```
In [95]: partb["Tot_Benes"].astype("float")
```

```
Out[95]: 565335     754.0
739226     140.0
579293      62.0
914854     160.0
1119575    499.0
           ...
803181     157.0
607656      40.0
156761      35.0
816944     195.0
843509      12.0
Name: Tot_Benes, Length: 11615, dtype: float64
```

```
In [101]: partb = partb.drop("Rndrng_Prvdr_Type",axis=1)
```

```
In [104]: partb.dtypes
```

```
Out[104]: Rndrng_Prvdr_RUCA                    float64
Tot_Benes                             object
Tot_Srvcs                            float64
Tot_Mdcr_Stdzd_Amt                   float64
Rndrng_NPI                             int64
                                       ...
Undersea and Hyperbaric Medicine       uint8
Urology                                uint8
Vascular Surgery                       uint8
Individual                             uint8
Organization                           uint8
Length: 98, dtype: object
```

```
In [ ]:
```

jupyter Untitled1 Last Checkpoint: Last Friday at 5:54 AM (autosaved)

## GETTING ALL THE FRAUDELENT DATA

```
In [90]: x = partb["Rndrng_NPI"].size
count = 0
y =[];
while (count < x):
    chk = partb["Rndrng_NPI"].iloc[count]
    chk1 = partb["Rndrng_Prvdr_First_Name"].iloc[count]
    chk2 = partb["Rndrng_Prvdr_Last_Org_Name"].iloc[count]
    chk3 = partb["Rndrng_Prvdr_MI"].iloc[count]
    if(chk in ex_npi.unique()):
        y.append(1)
    elif chk1 in ex_fnames.unique():
        y.append(1)
    elif chk2 in ex_lnames.unique():
        y.append(1)
    elif chk3 in ex_mi.unique():
        y.append(1)
    else:
        y.append(0)
    count = count+1
```

```
In [91]: partbx = pd.DataFrame(y, columns =['Fraud'])
```

```
In [95]: partbx.value_counts() # 1088 Fraudulent NPI's
```

```
Out[95]: Fraud
0        1160454
1           1088
dtype: int64
```

```
The Distribution of Fraud is
    1088/1160454 ~ 0.00090
```

## Labelling the Fraudulent Data

jupyter Untitled1 Last Checkpoint: Last Friday at 5:54 AM (autosaved)

```
Out[95]: Fraud
0        1160454
1           1088
dtype: int64
```

```
The Distribution of Fraud is
    1088/1160454 ~ 0.00090
```

### Labelling the Fraudulent Data

```
In [160]: x=partbx.size
c=0
partb["Fraud"]=partbx
while(c<x):
    partb["Fraud"].iloc[c]=partbx["Fraud"].iloc[c]
    c=c+1
```

```
In [161]: partb["Fraud"].value_counts()
```

```
Out[161]: 0    1160454
1       1088
Name: Fraud, dtype: int64
```

```
In [162]: partb.sample(frac=0.1,random_state=8700)["Fraud"].value_counts()
```

```
Out[162]: 0    116053
1       101
Name: Fraud, dtype: int64
```

```
In [163]: partb[partb["Fraud"]==0]
```

Out[163]:

| | Rndrng_Prvdr_Last_Org_Name | Rndrng_Prvdr_First_Name | Rndrng_Prvdr_MI | Rndrng_Prvdr_Crdntls | Rndrng_Prvdr_Gndr | Rndrng_Prvdr_Ent_Cd | Rndrng_ |
|---|---|---|---|---|---|---|---|
| 0 | Enkeshafi | Ardalan | NaN | M.D. | M | I | 6410 Ro |
| 1 | Cibull | Thomas | L | M.D. | M | I | 2650 |
| | | | | | | | 4126 |

jupyter Untitled1 Last Checkpoint: Last Friday at 5:54 AM (autosaved)

## PRE-PROCESSING OF PART_B DATASET

### Pre-Processing of Gender Attribute

```
Before Gender Processing The fraud Distribution
    1088/1160454 ~ 0.00093
After Gender Processing, The fraud Distribution
    1085/1099715 ~ 0.00098
```

```
In [164]: partb = partb.dropna(subset=["Rndrng_Prvdr_Gndr"],axis=0)
```

```
In [165]: partb["Fraud"].value_counts()
```

```
Out[165]: 0    1099715
1       1085
Name: Fraud, dtype: int64
```

```
In [166]: partb.isnull().sum()
```

```
Out[166]: Rndrng_Prvdr_Last_Org_Name         0
Rndrng_Prvdr_First_Name            0
Rndrng_Prvdr_MI               347060
Rndrng_Prvdr_Crdntls           71548
Rndrng_Prvdr_Gndr                  0
                               ...
Bene_CC_Sz_Pct                625351
Bene_CC_Strok_Pct             532783
Bene_Avg_Risk_Scre                 0
Rndrng_NPI                         0
Fraud                              0
Length: 74, dtype: int64
```

### Pre-Processing for Dprssn_Pct

```
Before Gender Processing The fraud Distribution
    1088/1099715 ~ 0.00098
```

```
In [167]: partb = partb.dropna(subset=["Bene_CC_Dprssn_Pct"],axis=0)

In [170]: partb["Fraud"].value_counts()

Out[170]: 0    920955
          1       941
          Name: Fraud, dtype: int64

In [171]: partb.isnull().sum()

Out[171]: Rndrng_Prvdr_Last_Org_Name         0
          Rndrng_Prvdr_First_Name            0
          Rndrng_Prvdr_MI               288444
          Rndrng_Prvdr_Crdntls           57400
          Rndrng_Prvdr_Gndr                  0
                                         ...
          Bene_CC_Sz_Pct                548801
          Bene_CC_Strok_Pct             423807
          Bene_Avg_Risk_Scre                 0
          Rndrng_NPI                         0
          Fraud                              0
          Length: 74, dtype: int64
```

**Pre-Processing for Middle name**

```
Before Gender Processing The fraud Distribution
    941/920955 ~ 0.0010
After Gender Processing, The fraud Distribution
    892/632560 ~ 0.0014
```

```
In [175]: partb.dropna(subset=["Rndrng_Prvdr_MI"],axis=0)["Fraud"].value_counts()

Out[175]: 0    632560
          1       892
          Name: Fraud, dtype: int64
```

**Pre-Processing for only the 'MOST_WANTED_ATTRIBUTES'**

```
892/632560 ~ 0.0014

In [175]: partb.dropna(subset=["Rndrng_Prvdr_MI"],axis=0)["Fraud"].value_counts()
Out[175]: 0    632560
          1       892
          Name: Fraud, dtype: int64
```

#### Pre-Processing for only the 'MOST_WANTED_ATTRIBUTES'

```
Note : The attributes are...
    1. NPI
    2. Provider_type
    3. Gender
    4. No_of_Services
    5. No_of_Bene_Days
    6. Submitted Charges
```

```
In [176]: partb = partb.drop(['Bene_Race_Wht_Cnt','Bene_Race_Black_Cnt', 'Bene_Race_API_Cnt', 'Bene_Race_Hspnc_Cnt','Bene_Race

In [179]: partb = partb.drop(["Rndrng_Prvdr_Mdcr_Prtcptg_Ind","Bene_Avg_Age","Rndrng_Prvdr_Crdntls","Rndrng_Prvdr_RUCA_Desc","

In [183]: partb = partb.drop(['Rndrng_Prvdr_Last_Org_Name', 'Rndrng_Prvdr_First_Name', 'Rndrng_Prvdr_MI',"Rndrng_Prvdr_St2","R
```

#### Pre-Processing for Total Beneficiaries Provided by the Provider

```
In [186]: partb["Tot_Benes"] = partb["Tot_Benes"].apply(lambda row : float(row.replace(",","")) if "," in row else float(row))

In [187]: partb["Tot_Benes"].isnull().sum()
Out[187]: 0
```

#### Pre-Processing for Gender

#### Pre-Processing for Total Beneficiaries Provided by the Provider

```
In [186]: partb["Tot_Benes"] = partb["Tot_Benes"].apply(lambda row : float(row.replace(",","")) if "," in row else float(row))

In [187]: partb["Tot_Benes"].isnull().sum()
Out[187]: 0
```

#### Pre-Processing for Gender

```
In [188]:  partb["Rndrng_Prvdr_Gndr"] = partb["Rndrng_Prvdr_Gndr"].apply(lambda row : 1 if row=="M" else 0)
```

#### ONE-HOT ENCODING FOR GENDER ATRIBUTE

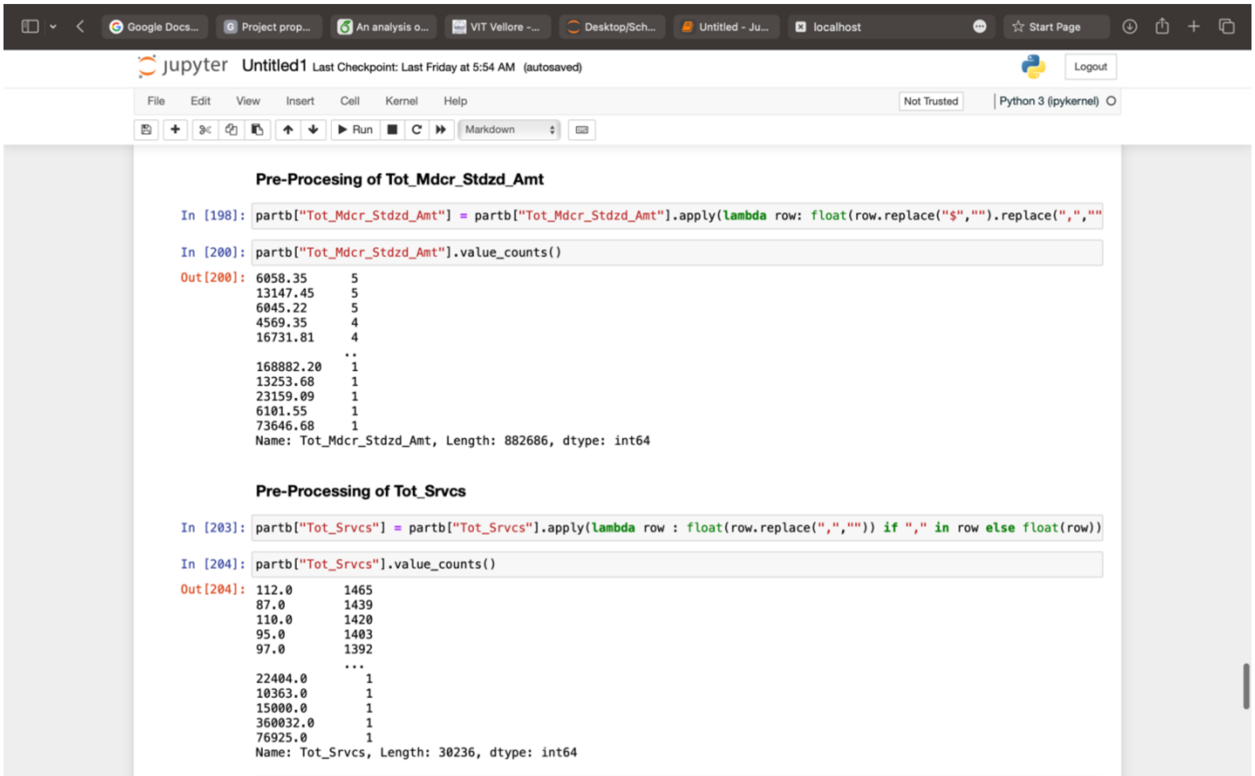
```
In [190]: partb[["Male","Female"]] = pd.get_dummies(partb["Rndrng_Prvdr_Gndr"])

In [192]: partb["Male"].isnull().sum()
Out[192]: 0
```

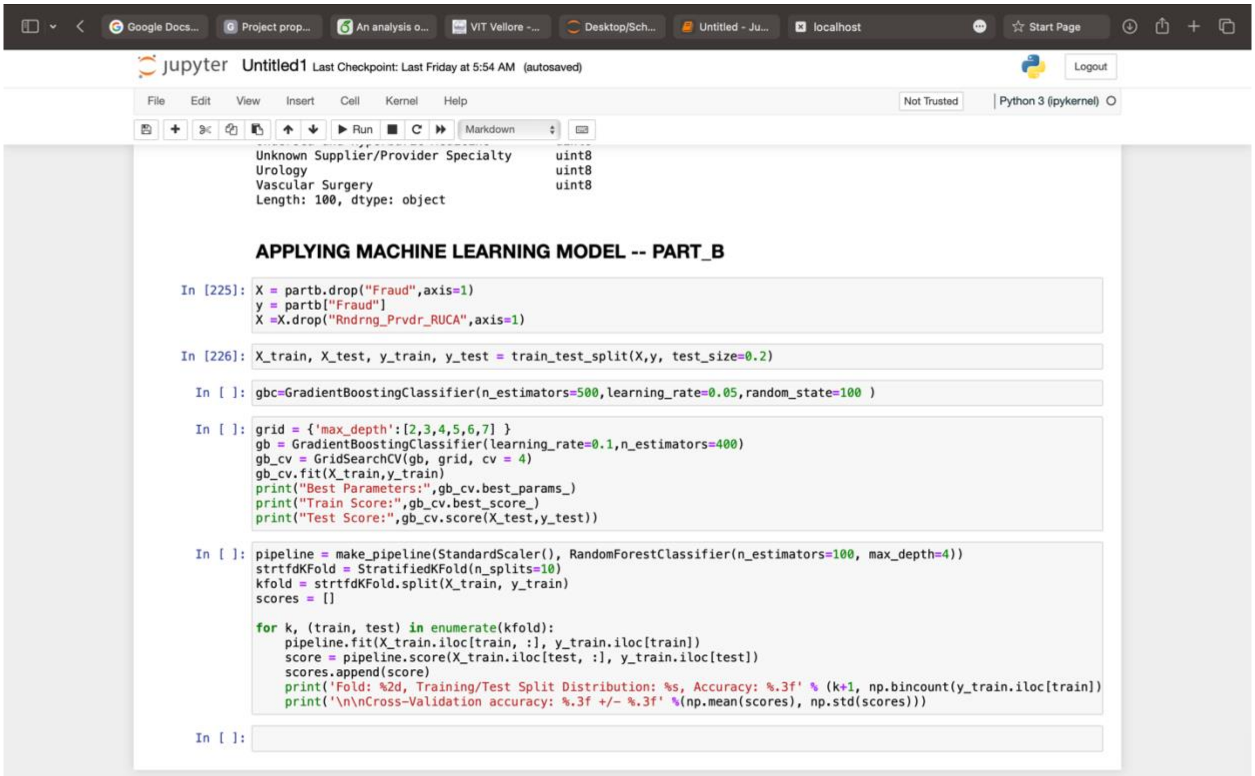
#### Pre-Processing of Rndrng_Prvdr_Type

```
In [194]: partb = pd.concat([partb,pd.get_dummies(partb["Rndrng_Prvdr_Type"])],axis=1)

In [195]: partb.shape
Out[195]: (921896, 102)
```

#### Pre-Procesing of Tot_Mdcr_Stdzd_Amt

```
In [198]: partb["Tot_Mdcr_Stdzd_Amt"] = partb["Tot_Mdcr_Stdzd_Amt"].apply(lambda row: float(row.replace("$","").replace(",",""

In [200]: partb["Tot_Mdcr_Stdzd_Amt"].value_counts()
```


Google Docs...
Project prop...
An analysis o...
VIT Vellore -...
Desktop/Sch...
Untitled - Ju...
localhost
Start Page
jupyter Untitled1 Last Checkpoint: Last Friday at 5:54 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Markdown
Pre-Procesing of Tot_Mdcr_Stdzd_Amt
In [198]: partb["Tot_Mdcr_Stdzd_Amt"] = partb["Tot_Mdcr_Stdzd_Amt"].apply(lambda row: float(row.replace("$","").replace(",",""
In [200]: partb["Tot_Mdcr_Stdzd_Amt"].value_counts()
Out[200]: 6058.35 5
13147.45 5
6045.22 5
4569.35 4
16731.81 4
..
168882.20 1
13253.68 1
23159.09 1
6101.55 1
73646.68 1
Name: Tot_Mdcr_Stdzd_Amt, Length: 882686, dtype: int64
Pre-Processing of Tot_Srvcs
In [203]: partb["Tot_Srvcs"] = partb["Tot_Srvcs"].apply(lambda row : float(row.replace(",","")) if "," in row else float(row))
In [204]: partb["Tot_Srvcs"].value_counts()
Out[204]: 112.0 1465
87.0 1439
110.0 1420
95.0 1403
97.0 1392
...
22404.0 1
10363.0 1
15000.0 1
360032.0 1
76925.0 1
Name: Tot_Srvcs, Length: 30236, dtype: int64



Google Docs...
Project prop...
An analysis o...
VIT Vellore -...
Desktop/Sch...
Untitled - Ju...
localhost
Start Page
jupyter Untitled1 Last Checkpoint: Last Friday at 5:54 AM (autosaved)
Logout
File Edit View Insert Cell Kernel Help
Not Trusted
Python 3 (ipykernel)
Run
Markdown
Unknown Supplier/Provider Specialty uint8
Urology uint8
Vascular Surgery uint8
Length: 100, dtype: object
APPLYING MACHINE LEARNING MODEL -- PART_B
In [225]: X = partb.drop("Fraud",axis=1)
y = partb["Fraud"]
X =X.drop("Rndrng_Prvdr_RUCA",axis=1)
In [226]: X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2)
In [ ]: gbc=GradientBoostingClassifier(n_estimators=500,learning_rate=0.05,random_state=100 )
In [ ]: grid = {'max_depth':[2,3,4,5,6,7] }
gb = GradientBoostingClassifier(learning_rate=0.1,n_estimators=400)
gb_cv = GridSearchCV(gb, grid, cv = 4)
gb_cv.fit(X_train,y_train)
print("Best Parameters:",gb_cv.best_params_)
print("Train Score:",gb_cv.best_score_)
print("Test Score:",gb_cv.score(X_test,y_test))
In [ ]: pipeline = make_pipeline(StandardScaler(), RandomForestClassifier(n_estimators=100, max_depth=4))
strtfdKFold = StratifiedKFold(n_splits=10)
kfold = strtfdKFold.split(X_train, y_train)
scores = []
for k, (train, test) in enumerate(kfold):
pipeline.fit(X_train.iloc[train, :], y_train.iloc[train])
score = pipeline.score(X_train.iloc[test, :], y_train.iloc[test])
scores.append(score)
print('Fold: %2d, Training/Test Split Distribution: %s, Accuracy: %.3f' % (k+1, np.bincount(y_train.iloc[train])
print('\n\nCross-Validation accuracy: %.3f +/- %.3f' %(np.mean(scores), np.std(scores)))
In [ ]:

## Conclusion and Future Works:

The importance of reducing Medicare fraud, in particular for individuals 65 and older, is paramount in the United States as the elderly population continues to grow. Medicare is necessary for many citizens, and therefore, the importance placed on quality research into fraud detection to keep healthcare costs fair and reasonable. CMS has made avail- able several Big Data Medicare claims datasets for public use over an ever-increasing number of years. Throughout this work, we provide a unique approach (combining mul- tiple Medicare datasets and leverage state-of-the-art Big Data processing and machine learning approaches) for determining the fraud detection capabilities of three Medicare datasets, individually and combined, using three learners, against real-world fraudulent physicians and other medical providers taken from the LEIE dataset.

We present our methods for processing each dataset from CMS, the Combined data- set, as well as the mapping of provider fraud labels. We ran experiments on all four data- sets: Part B, Part D, DMEPOS, and Combined. Each dataset was considered Big Data, requiring us to employ Spark on top of a Hadoop YARN cluster for running and validat- ing our models. Each dataset was trained and evaluated using three learners: Random Forest, Gradient Boosted Trees and Logistic Regression. The Combined dataset had the best overall fraud detection performance with an AUC of 0.816 using LR, indicating bet- ter performance than each of its individual Medicare parts, and scored similarly to Part B with no significant difference in average AUC. The DMEPOS dataset had the lowest overall results for all learners. Therefore, from these experimental findings and obser- vations, coupled with the notion that a physician/provider can commit fraud using any part of Medicare, we show that using the Combined dataset with LR provides the best overall fraud detection performance. Future work will include employing data sampling techniques to combat the imbalanced nature of known fraud events in evaluating the different Medicare datasets.

# References:

1. U.S. Government, U.S. Centers for Medicare & Medicaid Services. The Official U.S. Government Site for Medicare. https://www.medicare.gov/. Accessed 21 Jan 2017.
2. Feldstein M. Balancing the goals of health care provision and financing. Health Affairs. 2006;25(6):1603–11.
3. Administration for Community Living: Profile of older Americans. 2015. http://www.aoa.acl.gov/Aging_Statistics/Profile/2015/. Accessed 2015.
4. Dieleman JL, Squires E, Bui AL, Campbell M, Chapin A, Hamavid H, Horst C, Li Z, Matyasz T, Reynolds A. Factors associated with increases in us health care spending, 1996–2013. Jama. 2017;318(17):1668–78.
5. Medicare Fraud Strike Force. Office of Inspector General. https://www.oig.hhs.gov/fraud/strike-force/. Accessed 9 Feb 2017.
6. Aetna. The facts about rising health care costs. http://www.aetna.com/health-reform-connection/aetnasvision/facts-about-costs.html. Accessed 2015.
7. Morris L. Combating fraud in health care: an essential component of any cost containment strategy. Health Affairs. 2009;28(5):1351–6.
8. CMS: Center for Medicare and Medicaid Services. https://www.cms.gov/. Accessed 2017.
9. Medicare: US Medicare Program. https://www.medicare.gov. Accessed 2017.
10. Henry J. Kaiser family foundation: Medicare advantage. https://www.kff.org/medicare/fact-sheet/medicareadvantage/. Accessed 2017.
11. CMS: Research, Statistics, Data, and Systems. https://www.cms.gov/research-statistics-data-and-systems/research- statistics-data-and-systems.html. Accessed 18 Nov 2015.
12. Medicare.gov. What's medicare. https://www.medicare.gov/sign-up-change-plans/decide-how-to-get-medicare/ whats-medicare/what-is-medicare.html. Accessed 2017.
13. Bauder RA, Khoshgoftaar TM, Seliya N. A survey on the state of healthcare upcoding fraud analysis and detection. Health Serv Outcomes Res Methodol. 2017;17(1):31–55.
14. Rashidian A, Joudaki H, Vian T. No evidence of the effect of the interventions to combat health care fraud and abuse: a systematic review of literature. PLoS ONE. 2012;7(8):41988.