Project Report On

Development of AI Framework for Medicare Fraud Detection

Carried out at

CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING, ADVANCED COMPUTING TRAINING SCHOOL, KNOWLEDGE PARK, BANGALORE.

Under The Supervision of

Mr. Abhay Mane

Submitted By

Kiran Salunke (220350125039)

Priya Joshi (220350125056)

Akash Sali (220350125006)

Shreyash Kukde (220350125068)

Nandlal Panchal (220350125046)

PG DIPLOMA IN BIG DATA ANALYTICS
C-DAC, BANGALORE

Declaration

We hereby certify that the work being presented in the report entitled Cardiovascular stroke prediction System, in partial fulfilment of the requirements for the award of PG Diploma Certificate and submitted in the department of PGDBDA of the C-DAC Bangalore, is an authentic record of our work carried out during the period, 26th July 2022 to 26th September 2022 under the supervision of Mr. Abhay Mane, C-DAC Bangalore. The matter presented in the report has not been submitted by us for the award of any degree of this or any other Institute/University.

Mr. Kiran Salunke	**PRN: 220350125039**
Ms. Priya Joshi	**PRN: 220350125056**
Mr. Akash Sali	**PRN: 220350125006**
Mr. Shreyash Kukde	**PRN: 220350125068**
Mr. Nandlal Panchal	**PRN: 220350125046**

Acknowledgement

“Outstanding achievements are not possible in vacuum. It needs lots of help and assistance besides a healthy environment, luckily we had one.” We take this opportunity to express our profound sense of gratitude and respect to **C-DAC ACTS (Bangalore) and all the faculty** for providing us the environment that helped us learn and implement our learning in the project.

Working on this project **Medicare Fraud Detection** was a great learning experience for us. We would like to thank our project guide **Mr. Abhay Mane** for his valuable guidance that encouraged us to select such a project to work on. His support helped us overcome various obstacles and intricacies encountered during the course of this project work.

Kiran Salunke: 220350125039
Priya Joshi: 220350125056
Akash Sali: 210950125006
Shreyash Kukde: 210950125006
Nandlal Panchal: 210950125060

Chapter 1

Introduction and Overview of Project

Healthcare is a major industry in the U.S. with both private and government run programs. The costs of healthcare continue to rise, in part due to the increasing population of the elderly. U.S. healthcare spending from 2016 to 2020 has increased by 6.7% to reach $3 trillion and Medicare spending accounts for 20% of all health-care spending in the U.S. at about $600 billion. This rising elderly population, combined with the increased costs of Medicare, need cost-cutting solutions, where the reduction in fraud is one way to help recover costs and reduce overall payments. The impact of healthcare fraud is estimated to be between 3% to 10% of the nation's total healthcare spending and continuing to adversely impact the Medicare program and its beneficiaries (NHCAA 2021). Government has initialized the programs, such as the Medicare Fraud Strike Force (OIG 2021), enacted to help combat fraud, but continued efforts are needed to better mitigate the effects of fraud. These are our project's basic introduction and background; also it shows that there are huge business opportunities for Medicare Fraud Detection systems.

1.1 Objectives of the project

- Build a basic data Model to show the connections among the distinctive features.
- Build a thorough ML model to recognize fraud claims dependent on the various highlights: Service Providers (Doctors, Pharmacies), Insurance supporters (patients), Geo-segment and usually misuse drugs medicines.
- Setup a benchmark measurement to quantify and assess the test result.
- Build a end a end pipeline to bring data from source of claim, and generate a monthly dashboard to view fraud **National Provider Identifier (NPI).**
- Market-prepared item.

1.2 Technology and framework used

- Spark: Spark streaming, Pyspark
- Database: Mongo db
- Apache Airflow
- Apache Kafka
- Django

1.3 Software Required

- Jupyter Lab
- Mongo DB Compass
- Linux OS
- Power BI

1.4 Machine Learning

In a world where nearly, all manual tasks are being automated, the definition of ”manual” is changing. We are living in an era of constant technological progress, and one such advancement is in the field of machine learning.

Machine learning is a data analytics technique that teaches computers to do what comes naturally to humans and animals: learn from experience. Machine learning algorithms use computational methods to “learn” information directly from data without relying on a predetermined equation as a model. The algorithms adaptively improve their performance as the number of samples available for learning increases. Machine learning algorithms find natural patterns in data that generate insight and help you make better decisions and predictions.

Machine learning uses two types of techniques: Supervised learning, which trains a model on known input and output data so that it can predict future outputs, and Unsupervised learning, which finds hidden patterns or intrinsic structures in input data.

Supervised machine learning builds a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions for the response to new data.

Supervised learning uses classification and regression techniques to develop machine learning models. Classification techniques predict discrete responses. we use classification if our data can be tagged, categorized, or separated into specific groups or classes. Here in our problem, we have two discrete responses as ”stroke” or ”no stroke”. So, we used classification technique.

Regression techniques predict continuous responses. Unsupervised learning finds hidden patterns or intrinsic structures in data. It is used to draw inferences from datasets consisting of input data without labelled responses.

1.3.1 Advantages of Machine Learning

- Machine learning algorithms are continuously improving in accuracy and efficiency, which makes better decisions.
- Machine learning is responsible for cutting the workload and time. By automating things, we let the algorithm do the hard work for us.

- Machine learning algorithms are good at handling data that is multi-dimensional and multi-variety, and they can do this in dynamic or uncertain environments
- Machine learning has a wide variety of applications. This means that we can apply Machine learning to any of the major fields. Machine learning has a role everywhere, from medical, business, and banking to science and tech.

Chapter 2

Evaluation Metrics

2.1 Accuracy

Accuracy is one metric for evaluating classification models. Informally, Accuracy is the fraction of predictions our model got right. Formally, accuracy has the following definition:

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions} \tag{2.1}$$

For binary classification, accuracy can also be calculated in terms of positives and negatives as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.2}$$

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.

2.2 Precision and Recall

Precision is a good measure to determine, when the costs of False Positive is high. For instance, email spam detection. In email spam detection, a false positive means that an email that is non-spam (actual negative) has been identified as spam (predicted spam). The email user might lose important emails if the precision is not high for the spam detection model.

In the field of information retrieval precision is the fraction of retrieved documents that are relevant to the query:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{True\ Positive}{Total\ Predicted\ Positive} \tag{2.3}$$

Recall calculates how many of the Actual Positives our model capture through labelling it as Positive (True Positive). Applying the same understanding, we know that Recall shall be the model metric we use to select our best model when there is a high cost associated with False Negative.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{True\ Positive}{Total\ Actual\ Positive} \tag{2.4}$$

2.3 F1 Score

F1 is a function of Precision and Recall.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.5}$$

F1 Score might be a better measure to use if we need to seek a balance between Precision and Recall AND there is an uneven class distribution (large number of Actual Negatives).

2.4 Confusion matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values:

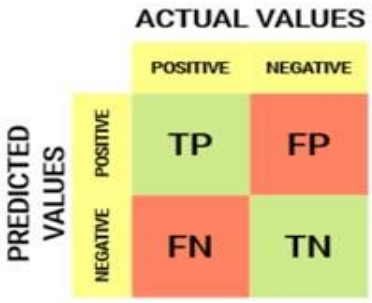


Figure 2.1: Confusion Matrix

Let's decipher the matrix:

- The target variable has two values: Positive or Negative
- The columns represent the actual values of the target variable
- The rows represent the predicted values of the target variable

Chapter 3

Data Pre-processing

Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model.

When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data pre-processing task.

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

Before going into pre-processing and data exploration we will explain some of the concepts that allowed us to select our features.

3.1 Pre-processing

3.1.1 Loading data

After downloading the **Provider Drug Data** from official site we come to know that the size of dataset is very large, so that we chose spark for loading the data set and we store that pre-processed dataset into MongoDB for further use here below is the screenshot

Imports

Here we are importing the all of the basic libraries for data pre-processing.

```
In [6]: import pandas as pd
        from pyspark import SparkFiles
        from pyspark.sql import SparkSession
        from pyspark.sql import types as T
        from pyspark.sql import Row
        #from pyspark.sql.functions import *
        from pyspark.ml.feature import *
        import pickle
        #import functions
        from pyspark.sql import functions
        from pyspark.sql.functions import isnan, when, count, col
        import warnings
        warnings.filterwarnings("ignore")
```

Loading dataset

1) Explore the first data set Provider drug data

A series of datasets that provide information on prescription drugs prescribed by individual physicians and other healthcare professionals and paid for under the Medicare Provider Drug data Prescription Drug program. These datasets contain information on total number of prescriptions dispensed, and the total drug cost by National Provider Identifier (NPI), prescription drug, and geography.

```
input_uri="mongodb://127.0.0.1:27017/"
output_uri="mongodb://127.0.0.1:27017/"
```

```
spark=SparkSession.builder.appName("Drug_data").config("spark.mongodb.input.uri",input_uri) \
        .config("spark.mongodb.output.uri",output_uri) \
        .config("spark.jars.packages","org.mongodb.spark:mongo-spark-connector_2.12:2.4.2") \
        .getOrCreate()
```

```
spark = SparkSession.builder.appName("DataLoad").getOrCreate()
```

Loading data with readcsv

```
df = spark.read.csv("D:/Laptop/Data/MUP_DPR_RY22_P04_V10_DY20_NPIBN_0.csv",inferSchema=True,header=True)
```

After loading dataset through csv, we have to explore the schema of the files for understanding the data. Used spark to load data, due to huge data size and unified data processing capability of spark.

Below is the schema of the dataset.

```
In [10]: df.printSchema()

root
 |-- Prscrbr_NPI: integer (nullable = true)
 |-- Prscrbr_Last_Org_Name: string (nullable = true)
 |-- Prscrbr_First_Name: string (nullable = true)
 |-- Prscrbr_City: string (nullable = true)
 |-- Prscrbr_State_Abrvtn: string (nullable = true)
 |-- Prscrbr_State_FIPS: string (nullable = true)
 |-- Prscrbr_Type: string (nullable = true)
 |-- Prscrbr_Type_Src: string (nullable = true)
 |-- Brnd_Name: string (nullable = true)
 |-- Gnrc_Name: string (nullable = true)
 |-- Tot_Clms: integer (nullable = true)
 |-- Tot_30day_Fills: double (nullable = true)
 |-- Tot_Day_Suply: integer (nullable = true)
 |-- Tot_Drug_Cst: double (nullable = true)
 |-- Tot_Benes: integer (nullable = true)
 |-- GE65_Sprsn_Flag: string (nullable = true)
 |-- GE65_Tot_Clms: integer (nullable = true)
 |-- GE65_Tot_30day_Fills: double (nullable = true)
 |-- GE65_Tot_Drug_Cst: double (nullable = true)
 |-- GE65_Tot_Day_Suply: integer (nullable = true)
 |-- GE65_Bene_Sprsn_Flag: string (nullable = true)
 |-- GE65_Tot_Benes: integer (nullable = true)
```

For better understanding we are changing the column names that is Total claims, city, drug name, and speciality. As shown below

Renaming column names of drug dataframe

```
In [11]:
df_rename = df.withColumnRenamed("Tot_Clms","Total_claims").withColumnRenamed("Prscrbr_First_Name","First_name")

In [12]: df_rename = df_rename.withColumnRenamed("Prscrbr_City","City").withColumnRenamed("Prscrbr_State_Abrvtn","State")

In [13]: df_rename = df_rename.withColumnRenamed("Gnrc_Name","Drug_name").withColumnRenamed("Prscrbr_NPI","NPI").withColumnRenamed("Prscrb

In [14]: df_rename = df_rename.withColumnRenamed("Prscrbr_Type","Speciality")

In [15]: df_rename.printSchema()

root
 |-- NPI: integer (nullable = true)
 |-- last_org_name: string (nullable = true)
```

After changing column names we observed that total drug cost, total claims, total daily supply , total 30 day supply are important columns so we are grouping all the rows on NPI with sum, min, max, on Provider drug dataset. As we have data of fraud NPI not of fraud claims, we have grouped all claims of every NPI. As shown below

Grouping all rows on npi with sum , mean and max for drug values

[16]:
```
group_cols = ['NPI']

agg_dict = {'Tot_Drug_Cst':['sum','mean','max'],
            'Total_claims':['sum','mean','max'],
            'Tot_Day_Suply':['sum','mean','max'],
            'Tot_30day_Fills':['sum','mean','max']
            }
```

[17]:
```
df_agg = df_rename.groupby("NPI").agg(functions.max('Tot_Drug_Cst'),
                                      functions.sum('Tot_Drug_Cst'),
                                      functions.mean('Tot_Drug_Cst'),
                                      functions.max('Total_claims'),
                                      functions.sum('Total_claims'),
                                      functions.mean('Total_claims'),
                                      functions.max('Tot_Day_Suply'),
                                      functions.sum('Tot_Day_Suply'),
                                      functions.mean('Tot_Day_Suply'),
                                      functions.max('Tot_30day_Fills'),
                                      functions.sum('Tot_30day_Fills'),
                                      functions.mean('Tot_30day_Fills'))

df_agg.show(1)
```

After grouping the data on NPI basis as we created df_agg data frame. We are merging the provider drug data with df_agg data frame.As shown below

Joining string and aggregated dataframe

In [25]:
```
df_join = df_names.join(df_agg,df_names.NPI == df_agg.NPI_1,"inner")
```

In [27]:
```
df_join.show(1)
```

```
+----------+----------+-------------+--------+-----+----------------+----------+----------------+----------------+----------
-------+----------------+----------------+----------------+----------------+----------------+----------------+--------
-----------+--------------------+--------------------+
|       NPI|First_name|LAST_ORG_NAME|    CITY|STATE|       SPECIALITY|     NPI_1|max(Tot_Drug_Cst)|sum(Tot_Drug_Cst)|avg(Tot_Dr
ug_Cst)|max(Total_claims)|sum(Total_claims)|avg(Total_claims)|max(Tot_Day_Suply)|sum(Tot_Day_Suply)|avg(Tot_Day_Suply)|max(Tot_
30day_Fills)|sum(Tot_30day_Fills)|avg(Tot_30day_Fills)|
+----------+----------+-------------+--------+-----+----------------+----------+----------------+----------------+----------
-------+----------------+----------------+----------------+----------------+----------------+----------------+--------
-----------+--------------------+--------------------+
|1003000126|   Ardalan|    Enkeshafi|Bethesda|   MD|Internal Medicine|1003000126|          4792.85|          5979.07|        74
7.38375|              30|             124|            15.5|              937|             3721|           465.125|
32.4|             138.2|              17.275|
+----------+----------+-------------+--------+-----+----------------+----------+----------------+----------------+----------
-------+----------------+----------------+----------------+----------------+----------------+----------------+--------
-----------+--------------------+--------------------+
only showing top 1 row
```

After joining the data frame we store data frame in csv format for next processes.

Saving dataframe to csv

```
In [35]: df_join.write.format("com.mongodb.spark.sql.DefaultSource") \
             .option("database","Medicare") \
             .option("collection","Drug_data") \
             .mode("overwrite").save()
```

```
In [36]: df_join.write.csv("../data/spark_csv/drug_data_final.csv")
```

2) Explore the Payment Drug Data

Physicians in the US are required to declare all payments received from pharmaceutical companies. In addition to "payments received" basic information (ex: amount, date, type) this dataset contains many other useful data elements such as physician ownership in company, consulting fees, charity indicator, dispute status etc. The whole datasets are including three parts: General Payment, Research Payment and Physician Ownership Details.

So, now to get the insight about the payments done to provider and the supplier we are taking payment drug data from the official site. As shown below

Loading of payment drug data

```
[ ] input_uri="mongodb://127.0.0.1:27017/"
    output_uri="mongodb://127.0.0.1:27017/"
```

```
[ ] spark=SparkSession.builder.appName("payment_data").config("spark.mongodb.input.uri",input_uri) \
            .config("spark.mongodb.output.uri",output_uri) \
            .config("spark.jars.packages","org.mongodb.spark:mongo-spark-connector_2.12:2.4.2") \
            .getOrCreate()
```

```
[ ] spark = SparkSession.builder.appName("Dataprocess").getOrCreate()
```

Loading payments data

```
[ ] df_payment= spark.read.csv("../data/OP_DTL_GNRL_PGYR2020_P06302022.csv",inferSchema=True,header = True)


    #PaymentRawData = "c:\\FIP\\OP_DTL_GNRL_PGYR2017_P01172020.csv"
```

Shape of data is shown below

```
[9]: len(data)
[9]: 11421366
[7]: len(data.columns)
[7]: 91
```

After loading the data we have to always check for null values so we are checking for null values. Null values are checked as shown below.

```
df_pay_req_cols.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_pay_req_cols.columns]
   ).show()
```

```
+--------------------+---------+--------+----+-----+-------------------------------+
|Covered_Recipient_NPI|FirstName|LastName|city|state|Total_Amount_of_Payment_USDollars|
+--------------------+---------+--------+----+-----+-------------------------------+
|                35838|    32637|   32542|   0|  242|                              3|
+--------------------+---------+--------+----+-----+-------------------------------+
```

```
[ ]  df_not_null = df_pay_req_cols.filter(df_pay_req_cols.Covered_Recipient_NPI.isNotNull())
```

```
[ ]  df_not_null.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df_not_null.columns]
   ).show()
```

```
+--------------------+---------+--------+----+-----+-------------------------------+
|Covered_Recipient_NPI|FirstName|LastName|city|state|Total_Amount_of_Payment_USDollars|
+--------------------+---------+--------+----+-----+-------------------------------+
|                   0|      187|      92|   0|  239|                              3|
+--------------------+---------+--------+----+-----+-------------------------------+
```

Here we can see that there are null values in the above data.as we can see that there are a smaller number of null values as compared to the total number of rows.so we can drop the null values.

As NPI is the only common column between drug data and payment. Any row with null NPI is and invalid input for us. Hence all rows with null NPI should be dropped. Rows dropped as shown below.

```
[14]: #filtering dataframe and selecting rows which have NPI.
      #NPI is the only valid column to on which we can join drug data and payment data

      df_not_null = df_pay_req_cols.filter(df_pay_req_cols.Covered_Recipient_NPI.isNotNull())
```

To get the information about total payment received by provider we are grouping the payment data on the basis of NPI and saving the data frame into a new csv file. As shown below

```
In [21]: #Grouping with NPI

         df_group = df_not_null.groupby("Covered_Recipient_NPI").agg(functions.sum('Total_Amount_of_Payment_USDollars').astype("float"))

In [22]: df_group.printSchema()

         root
          |-- Covered_Recipient_NPI: integer (nullable = true)
          |-- CAST(sum(Total_Amount_of_Payment_USDollars) AS FLOAT): float (nullable = true)

In [23]: df_group.show()

         +--------------------+-----------------------------------------------------+
         |Covered_Recipient_NPI|CAST(sum(Total_Amount_of_Payment_USDollars) AS FLOAT)|
         +--------------------+-----------------------------------------------------+
         |          1174523807|                                                23.88|
         |          1477545572|                                              6446.39|
         |          1063472678|                                              3904.13|
         |          1407880198|                                                65.85|
         |          1538555099|                                               626.33|
         |          1942389515|                                                12.21|
         |          1649444266|                                              2218.95|
         |          1760563407|                                              2512.32|
         |          1194871780|                                                20.48|
         |          1023188463|                                              3206.83|
         |          1538128616|                                            75.439995|
         |          1437130499|                                               268.26|
         |          1699722892|                                            300.58002|
         |          1598921892|                                               571.18|
         |          1093810996|                                                104.0|
```

After processing above two data sets we got two new data frames to get our final data we are going to merge those two data sets. For join the data sets we are using left join and it is joining on NPI column. As shown below

```
In [11]: pay_data_cols = df_pay.withColumnRenamed("NPI","NPI_1")

In [12]: df_drug_pay = df_drug.join(pay_data_cols,df_drug.NPI == pay_data_cols.NPI_1,"left")

In [13]: df_drug_pay.printSchema()

         root
          |-- NPI: integer (nullable = true)
          |-- First_name_drug: string (nullable = true)
          |-- City: string (nullable = true)
          |-- State: string (nullable = true)
          |-- Speciality: string (nullable = true)
          |-- max_Tot_Drug_Cst: float (nullable = true)
          |-- sum_Tot_Drug_Cst: float (nullable = true)
          |-- avg_Tot_Drug_Cst: float (nullable = true)
          |-- max_Total_claims: integer (nullable = true)
          |-- sum_Total_claims: long (nullable = true)
          |-- avg_Total_claims: double (nullable = true)
          |-- max_Tot_Day_Suply: integer (nullable = true)
          |-- sum_Tot_Day_Suply: long (nullable = true)
          |-- avg_Tot_Day_Suply: float (nullable = true)
          |-- max_Tot_30day_Fills: float (nullable = true)
          |-- sum_Tot_30day_Fills: float (nullable = true)
          |-- avg_Tot_30day_Fills: float (nullable = true)
          |-- NPI_1: integer (nullable = true)
          |-- First_name: string (nullable = true)
          |-- Last_name: string (nullable = true)
          |-- Total_payments: float (nullable = true)
```

3) Exploring the LELE data

This database contains a list of individuals and entities that are excluded from participating in federally funded healthcare programs (i.e. Medicare) due to previous healthcare fraud. We could treat the LEIE dataset as the semi-labelled data, because LEIE is the fraudster-based target but not a fraud one.

Loading of LELE dataset

LELE data

```
In [16]: df_fraud_1 = spark.read.csv("../data/UPDATED.csv",inferSchema=True,header=True)

In [17]: df_fraud_1.printSchema()

root
 |-- LASTNAME: string (nullable = true)
 |-- FIRSTNAME: string (nullable = true)
 |-- MIDNAME: string (nullable = true)
 |-- BUSNAME: string (nullable = true)
 |-- GENERAL: string (nullable = true)
 |-- SPECIALTY: string (nullable = true)
 |-- UPIN: string (nullable = true)
 |-- NPI: integer (nullable = true)
 |-- DOB: integer (nullable = true)
 |-- ADDRESS: string (nullable = true)
 |-- CITY: string (nullable = true)
 |-- STATE: string (nullable = true)
 |-- ZIP: string (nullable = true)
 |-- EXCLTYPE: string (nullable = true)
 |-- EXCLDATE: string (nullable = true)
 |-- REINDATE: integer (nullable = true)
 |-- WAIVERDATE: integer (nullable = true)
 |-- WVRSTATE: string (nullable = true)
```

We are filling the missing values of first name, last name, speciality, and city as shown below

```
In [148]: df_test["FIRSTNAME"].fillna("Empty",inplace=True)

In [149]: df_test["LASTNAME"].fillna("Empty",inplace=True)

In [150]: df_test["SPECIALTY"].fillna("Empty",inplace=True)

In [151]: df_test["CITY"].fillna("Not_filled",inplace=True)
```

Adding Target Class (Fraud Feature)

As we can see in dataset there is no target variable in the so, for that we have to make Target variable as “Fraud” with help of NPI column. We have defined frauds as “0” and non-fraud as “1”. As shown below.

```
[ ] df_second["fraud_2"] = 1
```

```
[ ] df_fraud_1.na.fill({"NPI":0})

[ ] df_fraud_npi = df_fraud_1.filter(df_fraud_1["NPI"] != 0).select("NPI")
```

We merged the processed the LELE data with a final data frame received from provider drug and payments datasets. As shown below.

```
[ ] df_final = df_drug_pay.join(df_fraud_value,df_drug_pay.NPI == df_fraud_value.NPI_fraud,"left")
```

Uploading data to mongo db

After all the processing, we uploaded final dataset to mongo DB. As shown below

[ ]:
```
final_df.write.format("com.mongodb.spark.sql.DefaultSource") \
    .option("database","Medicare") \
    .option("collection","final_dataframe") \
    .mode("overwrite").save()
```

[ ]:
```
#Second option of loading data in to mongo db

# client = MongoClient('localhost',27017)
# db = client["Medicare"]
# collection = db["final_dataframe"]
# collection.insert_many(df_finall1.apply(lambda x: x.to_dict(), axis=1).to_list())
```

[ ]:
```
dfraud = spark.createDataFrame(df_fraud_1)
```

[11]:
```
dffraud.write.format("com.mongodb.spark.sql.DefaultSource") \
    .option("database","Medicare") \
    .option("collection","fraud_data") \
    .mode("overwrite").save()
```

Chapter 4

Exploratory Data Analysis (EDA)

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

1. Understanding Data
2. Removing duplicates
3. Descriptive analysis
4. Univariate analysis
5. Bivariate analysis

Here we are importing the all of the basic libraries for data pre-processing as well as sklearn.

```
import pandas as pd
import numpy as np
from scipy import stats

#Data analysis
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import RobustScaler, StandardScaler, MinMaxScaler,PowerTransformer
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.metrics import confusion_matrix, accuracy_score,classification_report

#Sampling
from collections import Counter
from imblearn.under_sampling import RandomUnderSampler,NearMiss,NeighbourhoodCleaningRule,ClusterCentroids,CondensedNearestNeighbour
from imblearn.under_sampling import OneSidedSelection,AllKNN,EditedNearestNeighbours,RepeatedEditedNearestNeighbours,TomekLinks
from imblearn.combine import SMOTETomek,SMOTEENN
from imblearn.over_sampling import ADASYN

#Visualizing sampling output, dimensionality reduction
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA

# For model buidling
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier

# Custom module for prediction and model evalution
from utils import predict_and_evaluate
```

Understanding Data

The very first step in exploratory data analysis is to identify the type of variables in the dataset. Variables are of two types Numerical and Categorical. info() method to identify the data type of the variables in the dataset as well as it will tell the count of non-null values as well.
We can get the size of the dataset using the shape method.

```
[7]: df_ml.shape

[7]: (965056, 120)
```

Removing Duplicates

duplicates () method of pandas library is used to find out the number of duplicate records. After finding the number of duplicate records we will drop all those records using dropDuplicates() method.

```
1 print(f"Number of duplicate records = {df.duplicated().sum()}")
2 # Dropping duplicate records
3 df.drop_duplicates(inplace=True)
```

```
Number of duplicate records = 24
```

Descriptive analysis

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 967524 entries, 0 to 967523
Data columns (total 19 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   NPI                  967524 non-null  int32
 1   City                 967524 non-null  object
 2   State                967524 non-null  object
 3   Speciality           967523 non-null  object
 4   max_Tot_Drug_Cst     967524 non-null  float64
 5   sum_Tot_Drug_Cst     967524 non-null  float64
 6   avg_Tot_Drug_Cst     967524 non-null  float64
 7   max_Total_claims     967524 non-null  int32
 8   sum_Total_claims     967524 non-null  int32
 9   avg_Total_claims     967524 non-null  float64
 10  max_Tot_Day_Suply    967524 non-null  int32
 11  sum_Tot_Day_Suply    967524 non-null  int32
 12  avg_Tot_Day_Suply    967524 non-null  float64
 13  max_Tot_30day_Fills  967524 non-null  float64
 14  sum_Tot_30day_Fills  967524 non-null  float64
 15  avg_Tot_30day_Fills  967524 non-null  float64
 16  Name                 967511 non-null  object
 17  payments             344268 non-null  float64
 18  FRAUD                967524 non-null  int64
dtypes: float64(9), int32(5), int64(1), object(4)
```

	count	mean	std	min	25%	50%	75%	max
NPI	967524.0	1.487525e+09	2.808617e+08	1.003000e+09	1.245335e+09	1.487767e+09	1.730147e+09	1.972672e+09
max_Tot_Drug_Cst	967524.0	4.692474e+04	1.765130e+05	0.000000e+00	3.268900e+02	4.674190e+03	3.117565e+04	3.304279e+07
sum_Tot_Drug_Cst	967524.0	1.561615e+05	4.618680e+05	0.000000e+00	6.225700e+02	8.672450e+03	1.154560e+05	3.636709e+07
avg_Tot_Drug_Cst	967524.0	5.014290e+03	2.207238e+04	0.000000e+00	1.999200e+02	1.016710e+03	3.551860e+03	4.201276e+06
max_Total_claims	967524.0	1.448100e+02	4.622800e+02	1.100000e+01	2.300000e+01	5.400000e+01	1.630000e+02	1.887030e+05
sum_Total_claims	967524.0	1.332860e+03	3.064900e+03	1.100000e+01	4.800000e+01	2.050000e+02	1.206000e+03	3.304070e+05
avg_Total_claims	967524.0	3.423000e+01	3.549000e+01	1.100000e+01	1.745000e+01	2.629000e+01	4.225000e+01	7.528160e+03
max_Tot_Day_Suply	967524.0	7.550370e+03	1.363226e+04	1.100000e+01	4.350000e+02	1.850000e+03	8.393000e+03	7.888400e+05
sum_Tot_Day_Suply	967524.0	6.705178e+04	1.479187e+05	1.100000e+01	7.500000e+02	6.212000e+03	5.618200e+04	5.295565e+06
avg_Tot_Day_Suply	967524.0	1.331190e+03	1.603120e+03	1.100000e+01	3.176100e+02	8.664500e+02	1.865140e+03	3.204823e+05
max_Tot_30day_Fills	967524.0	2.677200e+02	6.057100e+02	1.100000e+01	2.900000e+01	7.600000e+01	2.910000e+02	1.908457e+05
sum_Tot_30day_Fills	967524.0	2.367580e+03	5.218790e+03	1.100000e+01	5.650000e+01	2.710000e+02	1.997600e+03	3.307991e+05
avg_Tot_30day_Fills	967524.0	5.273000e+01	5.691000e+01	1.100000e+01	2.100000e+01	3.670000e+01	6.800000e+01	1.068687e+04
payments	344268.0	2.993600e+03	6.186471e+04	1.000000e+00	4.300000e+01	1.390000e+02	5.000000e+02	1.834759e+07
FRAUD	967524.0	0.000000e+00	2.000000e-02	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00

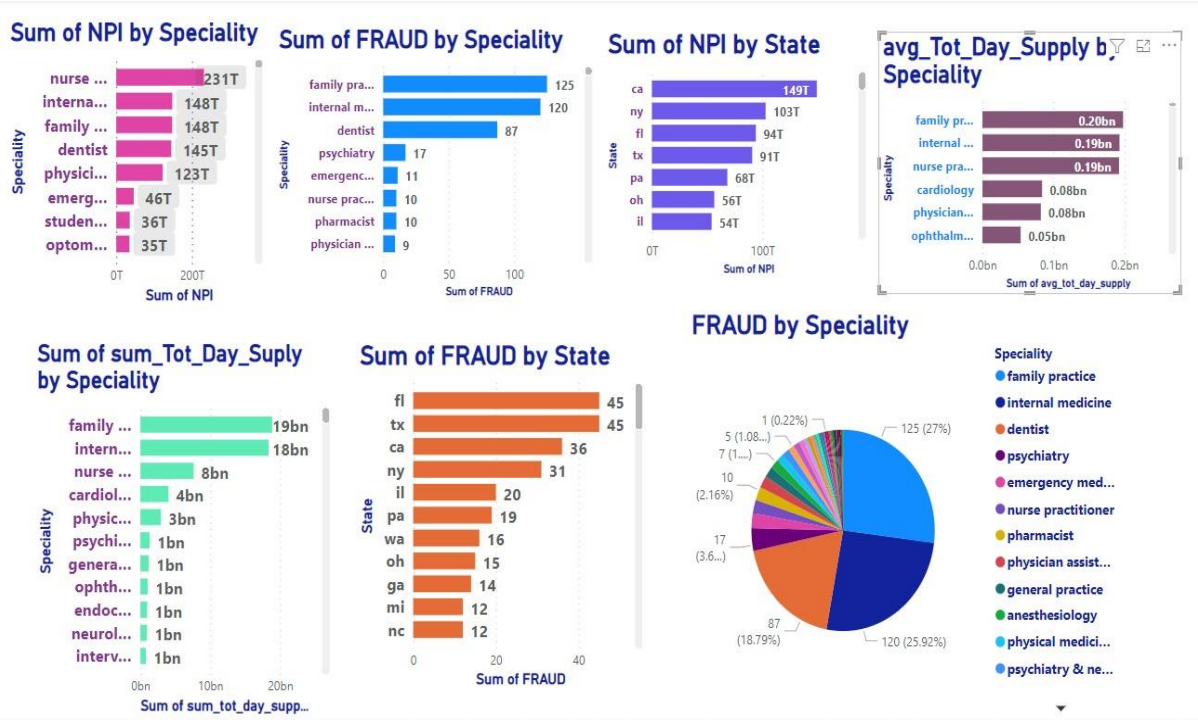
Data has high standard deviation, and is not uniformly distributed.

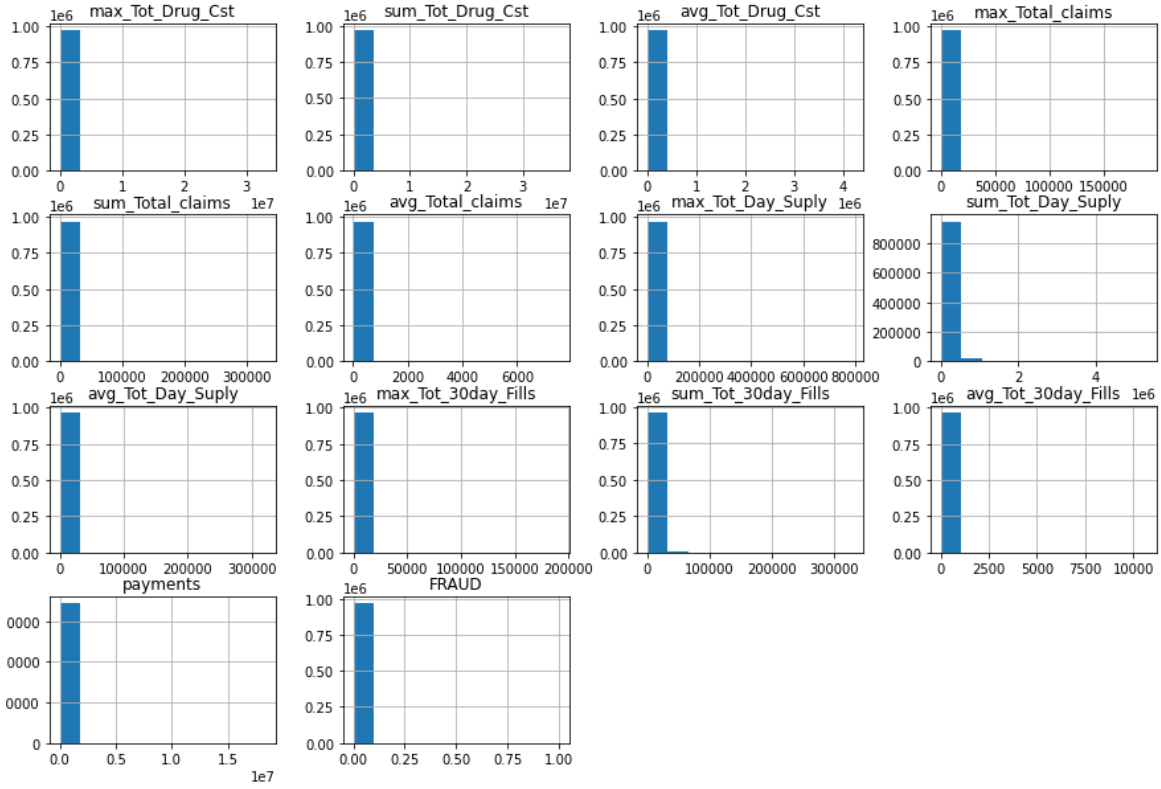
Univariant analysis

All features are positive skewed. Not uniformly distributed.

Nurse practitioner are among the top in fraud counts.

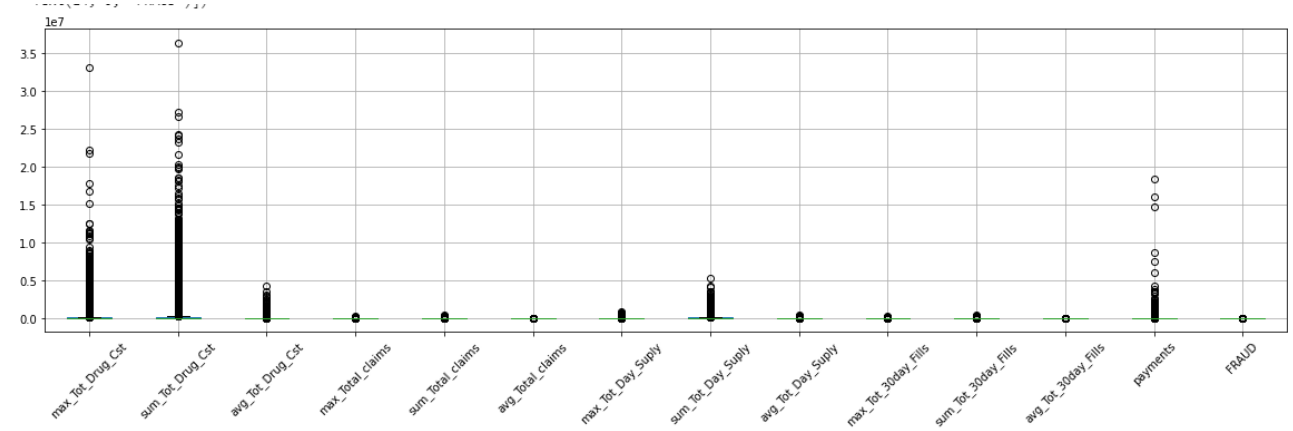
California being the top state in total frauds. As shown below.





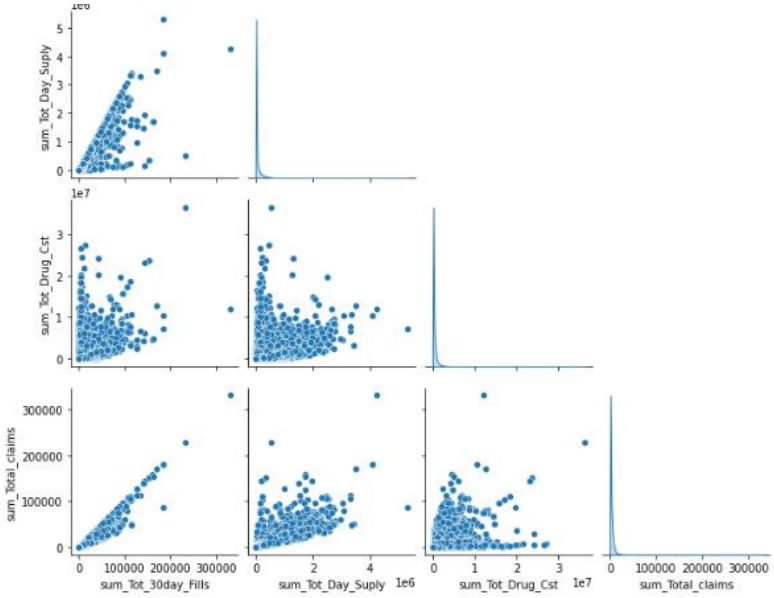
Outlier visualization

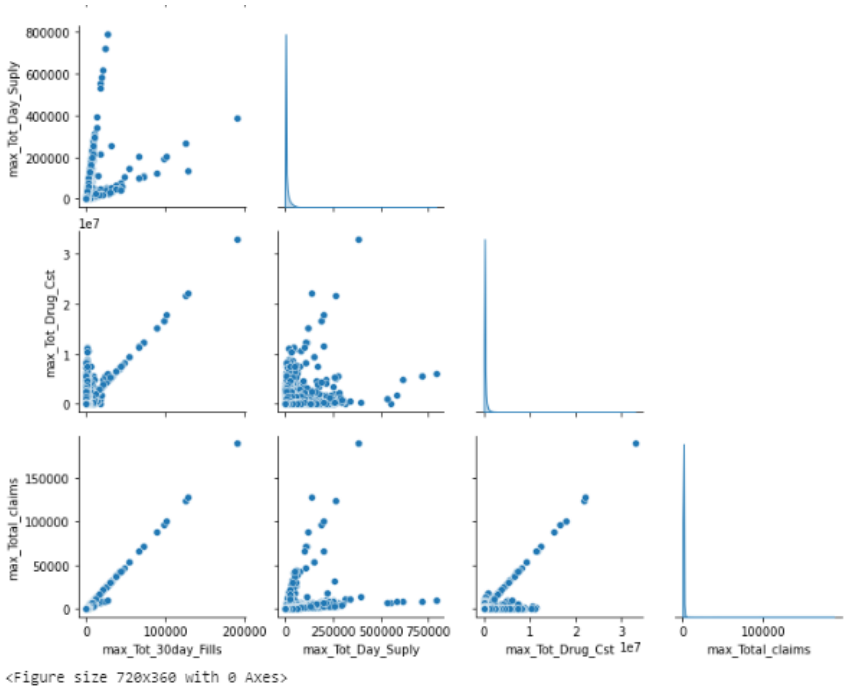
As we found more outliers in max, sum and average of total drug cost, sum total day supply and payments



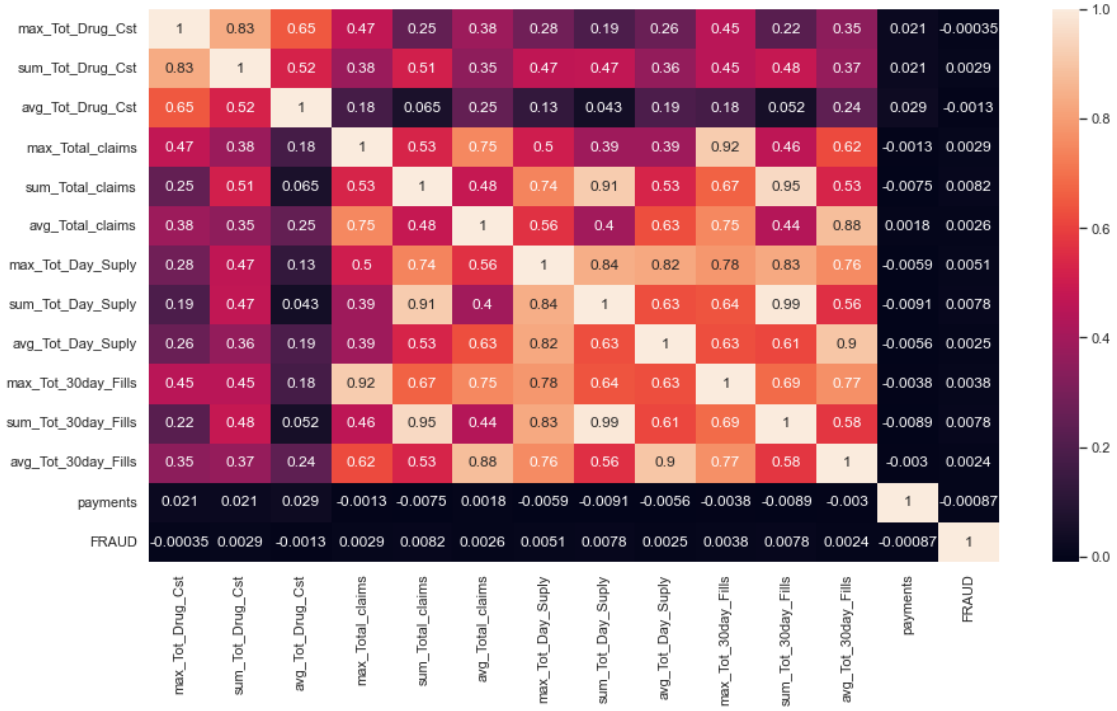
Bi variant analysis

Target variable has very low correlation with all attributes. Almost all Attributes have positive correlation, since they are related i.e., if total claims increase: total day supply and 30 day supply will increase: and total drug cost too. As shown below.





<Figure size 720x360 with 0 Axes>


max_Tot_Drug_Cst 1 0.83 0.65 0.47 0.25 0.38 0.28 0.19 0.26 0.45 0.22 0.35 0.021 -0.00035
sum_Tot_Drug_Cst 0.83 1 0.52 0.38 0.51 0.35 0.47 0.47 0.36 0.45 0.48 0.37 0.021 0.0029
avg_Tot_Drug_Cst 0.65 0.52 1 0.18 0.065 0.25 0.13 0.043 0.19 0.18 0.052 0.24 0.029 -0.0013
max_Total_claims 0.47 0.38 0.18 1 0.53 0.75 0.5 0.39 0.39 0.92 0.46 0.62 -0.0013 0.0029
sum_Total_claims 0.25 0.51 0.065 0.53 1 0.48 0.74 0.91 0.53 0.67 0.95 0.53 -0.0075 0.0082
avg_Total_claims 0.38 0.35 0.25 0.75 0.48 1 0.56 0.4 0.63 0.75 0.44 0.88 0.0018 0.0026
max_Tot_Day_Suply 0.28 0.47 0.13 0.5 0.74 0.56 1 0.84 0.82 0.78 0.83 0.76 -0.0059 0.0051
sum_Tot_Day_Suply 0.19 0.47 0.043 0.39 0.91 0.4 0.84 1 0.63 0.64 0.99 0.56 -0.0091 0.0078
avg_Tot_Day_Suply 0.26 0.36 0.19 0.39 0.53 0.63 0.82 0.63 1 0.63 0.61 0.9 -0.0056 0.0025
max_Tot_30day_Fills 0.45 0.45 0.18 0.92 0.67 0.75 0.78 0.64 0.63 1 0.69 0.77 -0.0038 0.0038
sum_Tot_30day_Fills 0.22 0.48 0.052 0.46 0.95 0.44 0.83 0.99 0.61 0.69 1 0.58 -0.0089 0.0078
avg_Tot_30day_Fills 0.35 0.37 0.24 0.62 0.53 0.88 0.76 0.56 0.9 0.77 0.58 1 -0.003 0.0024
payments 0.021 0.021 0.029 -0.0013 -0.0075 0.0018 -0.0059 -0.0091 -0.0056 -0.0038 -0.0089 -0.003 1 -0.00087
FRAUD -0.00035 0.0029 -0.0013 0.0029 0.0082 0.0026 0.0051 0.0078 0.0025 0.0038 0.0078 0.0024 -0.00087 1
max_Tot_Drug_Cst
sum_Tot_Drug_Cst
avg_Tot_Drug_Cst
max_Total_claims
sum_Total_claims
avg_Total_claims
max_Tot_Day_Suply
sum_Tot_Day_Suply
avg_Tot_Day_Suply
max_Tot_30day_Fills
sum_Tot_30day_Fills
avg_Tot_30day_Fills
payments
FRAUD
1.0
0.8
0.6
0.4
0.2
0.0

Chapter 5

Feature Engineering

Feature engineering is the pre-processing step of machine learning, which is used to transform raw data into features that can be used for creating a predictive model using Machine learning or statistical Modelling. Feature engineering in machine learning aims to improve the performance of models.

Feature engineering is the process of selecting, manipulating, and transforming raw data into features that can be used in supervised learning. In order to make machine learning work well on new tasks, it might be necessary to design and train better features. As you may know, a "feature" is any measurable input that can be used in a predictive model — it could be the colour of an object or the sound of someone's voice. Feature engineering, in simple terms, is the act of converting raw observation.

Feature engineering is a machine learning technique that leverages data to create new variables that aren't in the training set. It can produce new features for both supervised and unsupervised learning, with the goal of simplifying and speeding up data transformations
In this section we transform features and make them model ready.

1. Handling missing values.
2. Encoding the features.
3. Scaling the features.
4. Sampling: Under/Over/Combination
5. Outlier treatment

5.1 Handling missing values

```
df.isnull().sum()
```

```
Unnamed: 0                 0
NPI                        0
City                       0
State                      0
Speciality                 1
max_Tot_Drug_Cst           0
sum_Tot_Drug_Cst           0
avg_Tot_Drug_Cst           0
max_Total_claims           0
sum_Total_claims           0
avg_Total_claims           0
max_Tot_Day_Suply          0
sum_Tot_Day_Suply          0
avg_Tot_Day_Suply          0
max_Tot_30day_Fills        0
sum_Tot_30day_Fills        0
avg_Tot_30day_Fills        0
Total_payments        624634
FRAUD                      0
dtype: int64
```

```
df.Total_payments.isnull().sum()/965056
```

```
0.6472515584587838
```

Missing values in total payments are more than 60%. Also being a important feature, decided not to drop the feature for now.
Using Misforest algorithm to impute missing values.

To check the performance of Missforest, assigning null values to know payment values.

```
inds = df_sample.index.to_list()
for i in inds:
    df_1.at[i,"Total_payments"] = np.nan
```

Fitting Missforest on all numerical features and dropping target variable:

```
imputer = MissForest()
```

```
#df_imputed = imputer.fit_transform(df_1.drop(["NPI","State","Speciality","FRAUD"],axis = 1))
df_imputed = imputer.fit_transform(df_1.drop(["NPI","City","State","Speciality","FRAUD"],axis = 1))
```

Fitting algorithm on specific features:

```
df_imputed_2 = imputer.fit_transform(df_1.loc[:,["avg_Tot_Drug_Cst","avg_Total_claims","avg_Tot_Day_Suply","avg_Tot_30day_Fills","Total_payments"]])
```

Error metrics of imputed know total payments value.

```
MSE with all numeric features :  23568038.22044275
RMSE with all numeric features :  4854.692391948511
MSE with avg numeric features :  21743392.210048605
RMSE with avg numeric features :  4662.981043286431
```

Observed improved model performance with payments column as compared to dropping payments column. Hence we have kept payments column in data frame.

5.2 Encoding:

5.2.1 Medicines feature:

As categories in medicine feature are 1722. Traditional encoding technique will result in dimensionality explosion.

```
# storing unique drug names in set
#is instance will return false if drug name is not a string && set datatype - for no duplicacy
drugs = set([ drugx for drugx in partD_drug['Drug_Name'].values if isinstance(drugx, str)])
print(len(drugs))
```

```
1722
```

Hypothesis test

Hypothesis testing is an act in statistics whereby an analyst tests an assumption regarding a population parameter. The methodology employed by the analyst depends on the nature of the data used and the reason for the analysis.

Hypothesis testing is used to assess the plausibility of a hypothesis by using sample data. Such data may come from a larger population, or from a data-generating process. The word "population" will be used for both of these cases in the following descriptions.

A hypothesis is a proposed explanation for a phenomenon. A statistical hypothesis, sometimes called confirmatory data analysis, is a hypothesis that is testable on the basis of observing a process that is modelled via a set of random variables A statement about the parameters describing a population (not a sample).

For drawing some inferences, we have to make some assumptions that lead to two terms that are used in the hypothesis testing.

To derive statistical inference of if a certain category of medicine has impact on predicting the result.

To add more features from medicines column.

Purpose is to check if a medicine category has variance when claim made by fraud vs non fraud.

If variance is significant: which implies it will helpful in final prediction - so we will add those medicine in to data frame.

This is a test for the null hypothesis that 2 independent samples have identical average (expected) values. This test assumes that the populations have identical variances by default.

Null hypothesis: Mean of purchase of a certain medicine by a non-fraud person is identical to fraud person.

Alternate hypothesis: Mean of purchase of a certain medicine by a non-fraud person is not identical to fraud person. Hence this category of medicine can help in predicting the fraud.

```
for drugx in drug_with_isfraud:
    for colx in cols:
        fraud_0 = partD_drug_Group.get_group((drugx,0.0))[colx].values
        fraud_1 = partD_drug_Group.get_group((drugx,1.0))[colx].values
        #print(len(fraud_0), len(fraud_1))

        #If column has values more than 2 then do tt test on those and add result to dictionary with drug name
        if (len(fraud_0)>2) & (len(fraud_1)>2) :
            tt = ttest_ind(fraud_0, fraud_1)
            re_drug_tt[(drugx, colx)] = tt
```

Error metric in hypothesis

Type I: The null hypothesis is true but the hypothesis test incorrectly rejects it**.**

Type II: The null hypothesis is false but the hypothesis test fails rejects it**.**

Level of significance (alpha): Probability of making type I error.

Confidence coefficient: Probability of correctly failing to reject the null hypothesis. (1 – alpha).

1-beta: Represents the probability of correctly rejecting the null hypothesis when it is indeed false. Also called power of the test.

Larger sample size, large 1-beta.

Level of significance (alpha) = 0.01

As our sample size is huge.

This will help us reduce type I and type II error.

```
#Setting Probilities
#If p value is less than alpha = 0.05 then rejecting null hypothesis and adding that column in to list of prob_005

Prob_005 = [(key, pvalue) for (key, (stat, pvalue)) in re_drug_tt.items() if pvalue <=0.01]
print(len(Prob_005))
```

```
102
```

A p-value larger than a chosen threshold (1%) indicates that our observation is not so unlikely to have occurred by chance. Therefore, we do not reject the null hypothesis of equal population means. If the p-value is smaller than our threshold, then we have evidence against the null hypothesis of equal population means. Hence, we can reject null hypothesis and add medicine as a feature.

P-value of 0.01 indicates 1% probability that outcomes from sample data happened coincidently. And 99% probability that data is good and we can rely on it.

Lower P-value indicates data did not happen coincidently.

On getting medicines which are statistically significant

We add sum of values of respective medicines i.e., total drug cost, claims extra to our data frame.

5.2.3 Speciality

Again, as total speciality categories are large, encoding all will result in dimensionality explosion.

```
print("Unique speciality throughout dataset:" )
print(df_scale["Speciality"].value_counts().count())

#speciality_cat = pd.DataFrame.from_dict(df_ml["Speciality"].value_counts().to_dict(),orient='index',columns=["count"])
#print(df_ml[df_ml["FRAUD"] == 1]["Speciality"].value_counts().to_dict())

#df_ml["Speciality"].value_counts().sort_values(ascending=False).head(60)
```

```
Unique speciality throughout dataset:
181
```

Selecting top 30 speciality

```
# make list with top 30 variables
top_30 = [x for x in df_scale.Speciality.value_counts().sort_values(ascending=False).head(30).index]
#top_30.append('anesthesiology')
top_30
```

```
# for all categorical variables we selected
def top_x(df2,variable,top_x_labels):
    for label in top_x_labels:
        df2[variable+'_'+label] = np.where(df_scale[variable]==label,1,0)
```

5.3 Scaling

Need for Normalizing is we have values of different columns lies within different range of value. Normalizing or scaling will make all data values to fall within common limits. The machine learning model which are about implement may divert because of different range of

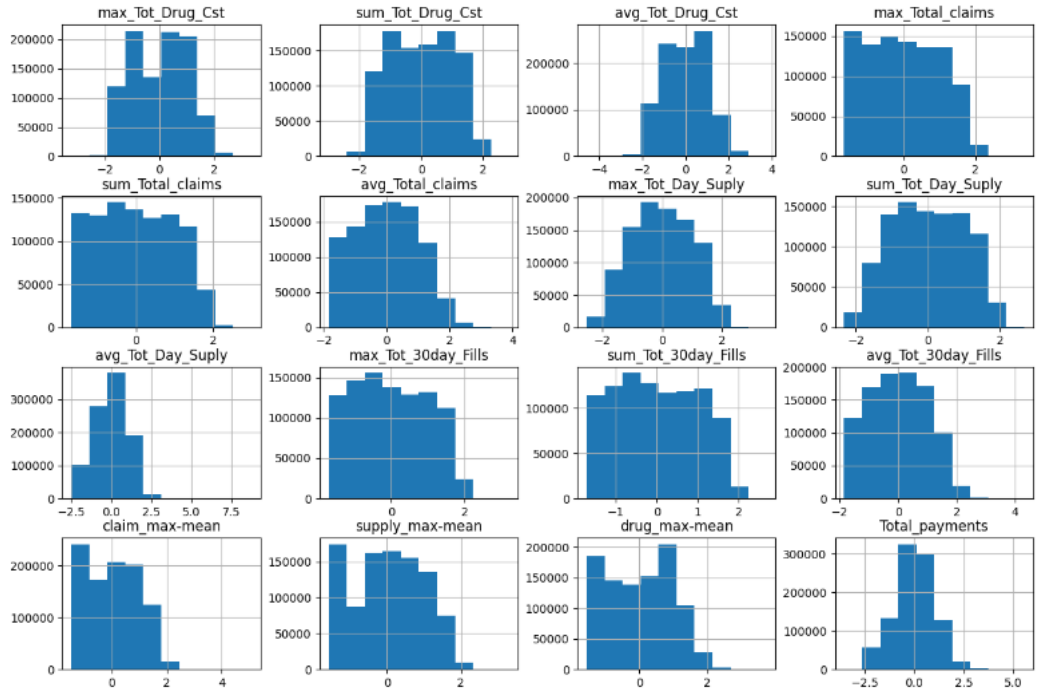
Values from different columns. We normalize or scale data from numerical columns only. For the scaling purpose we tried:

- Standard scaler
- Robust scaler
- Power transformer

Power transformer scaler

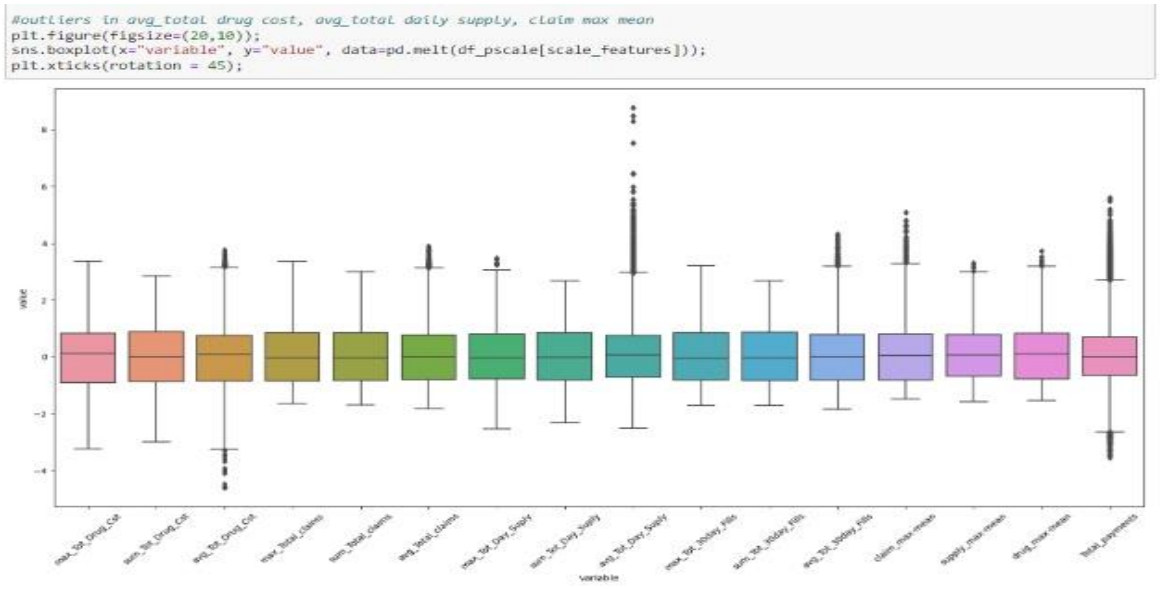
Since all features has positive skew, log scale will help to make distribution normal. As it is more desirable to feed model a normally distributed data for better results.

Result of power transformer, skewness has reduced. As shown below



Outlier visualization after implementing power transformer

The distribution of data is uniform as shown below.

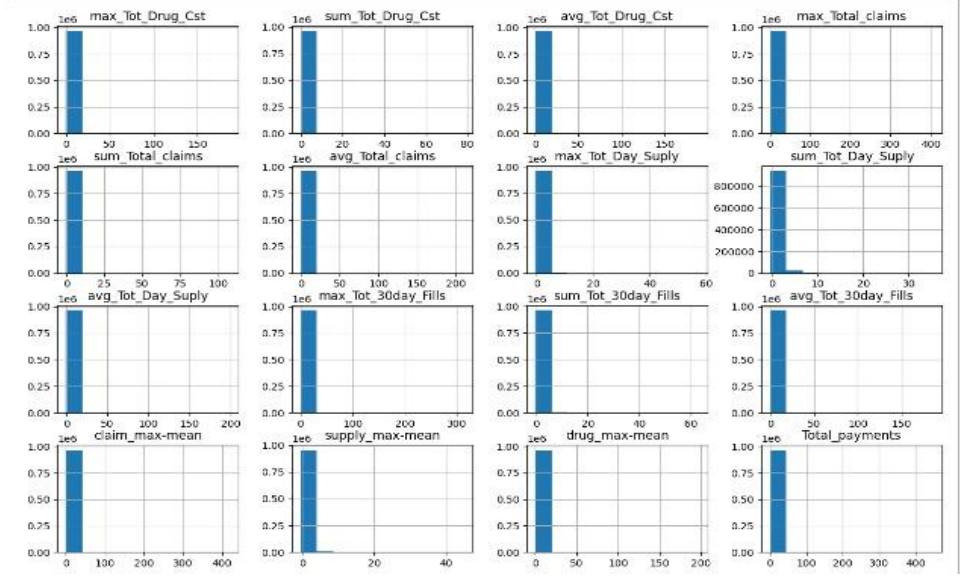


Standard scaler

The data are scaled to a variance of 1 after the mean is reduced to 0 via StandardScaler. Output after standard scaler is shown below and no change in skewness is observed.

```
df_stdscale =  df_m1.copy()
df_stdscale[scale_features] = pd.DataFrame(
    ss.fit_transform(df_stdscale.loc[:,scale_features]), columns=scale_features
)
```

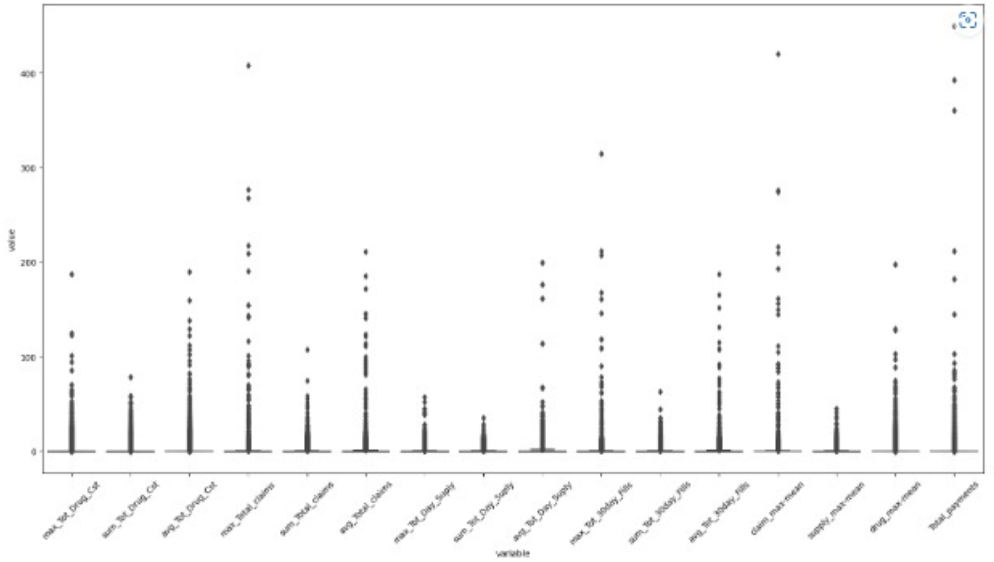
```
df_stdscale[scale_features].hist(figsize=(15, 10));
```



Outlier visualization after implementing standard scaler

No change in data distribution observed as shown below.

```
#outliers in avg_total drug cost, avg_total daily supply, claim max mean
plt.figure(figsize=(20,10));
sns.boxplot(x="variable", y="value", data=pd.melt(df_stdscale[scale_features]));
plt.xticks(rotation = 45);
```

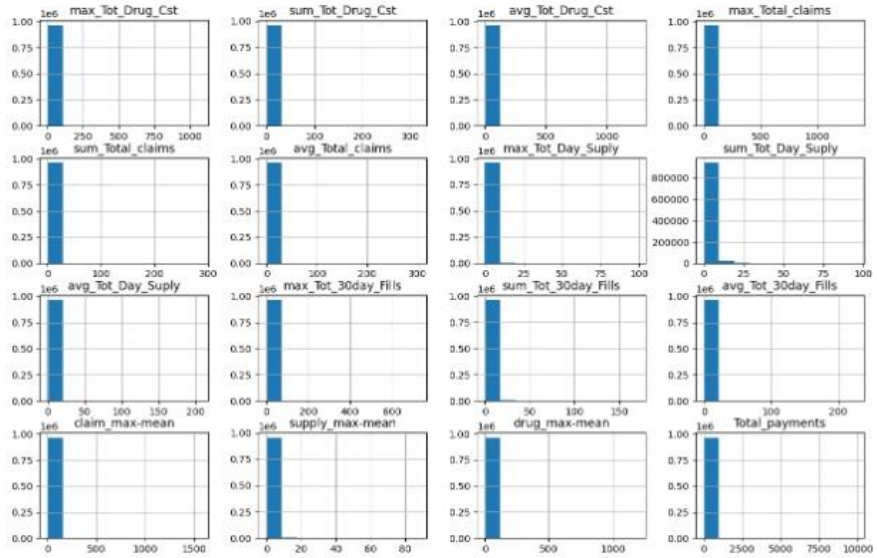


Robust Scaler

This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range) As shown in figure below.

```
#No change in skewness again

df_rbscale =  df_m1.copy()
df_rbscale[scale_features] = pd.DataFrame(
    rb.fit_transform(df_rbscale.loc[:,scale_features]), columns=scale_features
)
```

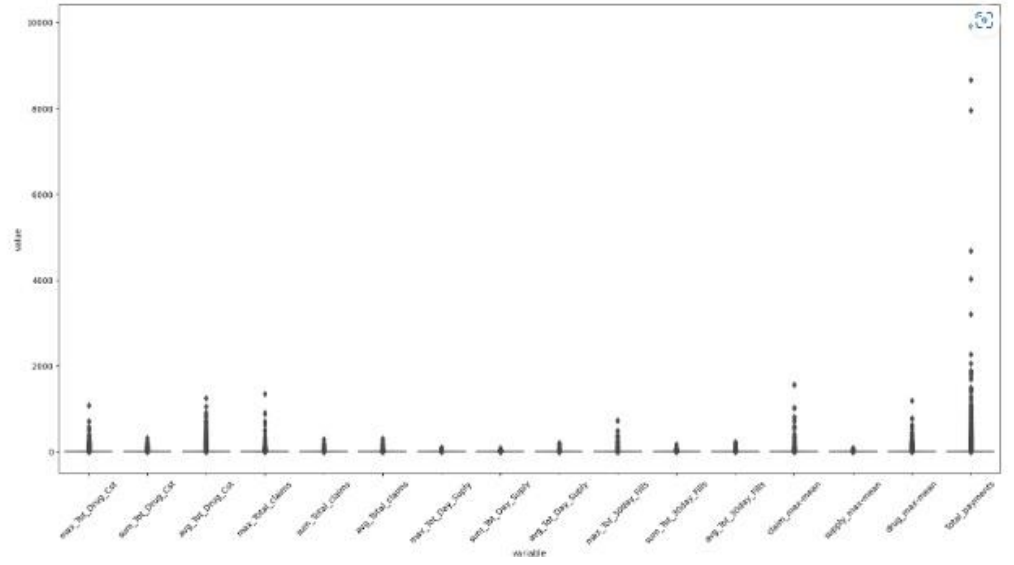
```
df_rbscale[scale_features].hist(figsize=(15, 10));
```



Outlier visualization after implementing robust scaler

No change in data distribution observed as shown below.

```
#outliers in avg_total drug cost, avg_total daily supply, claim max mean
plt.figure(figsize=(20,10));
sns.boxplot(x="variable", y="value", data=pd.melt(df_rbscale[scale_features]));
plt.xticks(rotation = 45);
```



Since power transformer has reduced the skewness of the data and gave better model performance as compared to other scalers. Hence we have selected power transformer in data set.

5.5 Sampling

We wanted to test model on un sampled and untreated dataset hence, sampling is done only on train dataset.

Separating Data in train and test

Step carried out before sampling.

Any machine learning algorithm needs to be tested for accuracy. In order to do that, we divide our data set into two parts (With the help of "STRATIFIED SHUFFLE SPLIT"): training learn the behaviours present in the data and check the correctness of the algorithm by testing-on-testing set.

Here we will check the distribution of class label in complete dataset as well distribution in training and testing dataset.

```
[9]: ss = StratifiedShuffleSplit(n_splits=1,
                                test_size=0.2,
                                train_size=0.8,
                                random_state=42)

[10]: X = df1.drop('fraud', axis=1)
      y = df1['fraud']
```

```
[18]: print('Distributions: \n')
      print("Train Set")
      print(train_df.fraud.value_counts())
      print("\nTest Set")
      print(test_df.fraud.value_counts())
      print("\nPercentage:")
      print("\nTrain Set")
      print((train_df.fraud.value_counts()/ len(train_df))*100)
      print("\nTest Set")
      print((test_df.fraud.value_counts()/ len(test_df))*100)

      Distributions:

      Train Set
      0    771335
      1       709
      Name: fraud, dtype: int64

      Test Set
      0    192834
      1       177
      Name: fraud, dtype: int64

      Percentage:

      Train Set
      0    99.908166
      1     0.091834
      Name: fraud, dtype: float64

      Test Set
      0    99.908295
      1     0.091705
      Name: fraud, dtype: float64
```

We have applied several sampling techniques and checked model performance.

Over sampling techniques

We have applied Adasyn, random over sampler and smote over sampling techniques but the performance of model was not satisfactory.
Result had high false negative hence not a desirable output.
Below is a sample result.

MODEL	sampling	FP	FN	precision	recall	f1 score	accuracy
decision tree	SMOTE	831	156	0.02	0.12	0.04	0.9
cataboost	ADASyn	191	171	0.03	0.03	0.03	1
gradient boosting classifier	Smote	23	176	0.04	0.01	0.01	0.9
cataboost	ADASyn	191	171	0.03	0.03	0.03	1

Combination of over sampling and under sampling

We applied Smotetomek and Smoteenn techiques but results were not satisfactory. As it gave high false negative.
Below is a sample result.

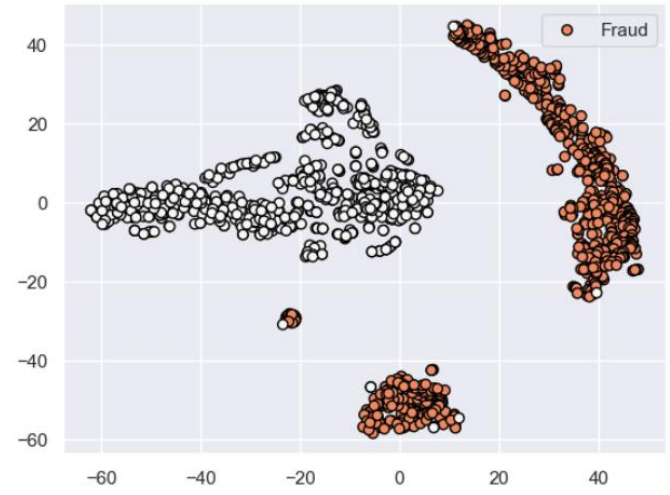
```
#Under sampling : SMOTETomek
#features : 40
#Outlier treatment : no
#Payments : added
```

	Algorithm	False Positives	False Negatives	Precision	Recall	F1 Score	Accuracy
0	DecisionTreeClassifier	1919	158	0.01	0.11	0.02	0.99
1	RandomForestClassifier	172	157	0.10	0.11	0.11	1.00
2	XGBClassifier	16584	123	0.00	0.31	0.01	0.91
3	CatBoostClassifier	3471	148	0.01	0.16	0.02	0.98
4	LogisticRegression	65412	51	0.00	0.71	0.00	0.66
5	GradientBoostingClassifier	49731	61	0.00	0.66	0.00	0.74
6	GaussianNB	31607	128	0.00	0.28	0.00	0.84
7	MLPClassifier	118561	23	0.00	0.87	0.00	0.39

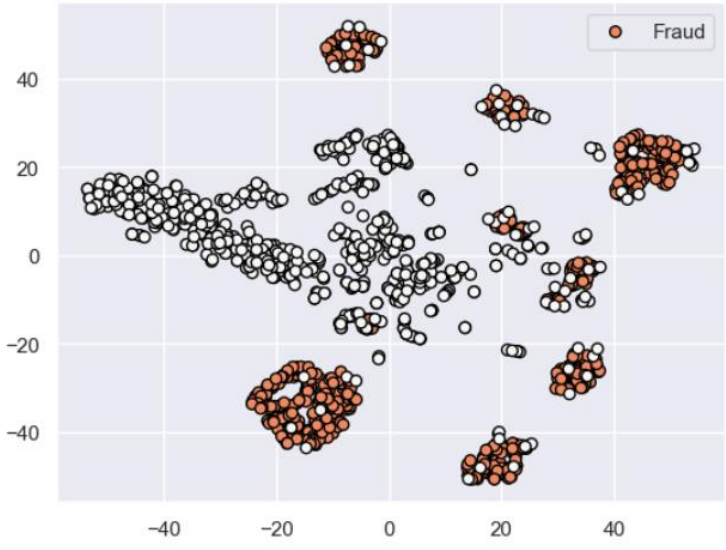
Under sampling Techniques

Near miss 1,2,3

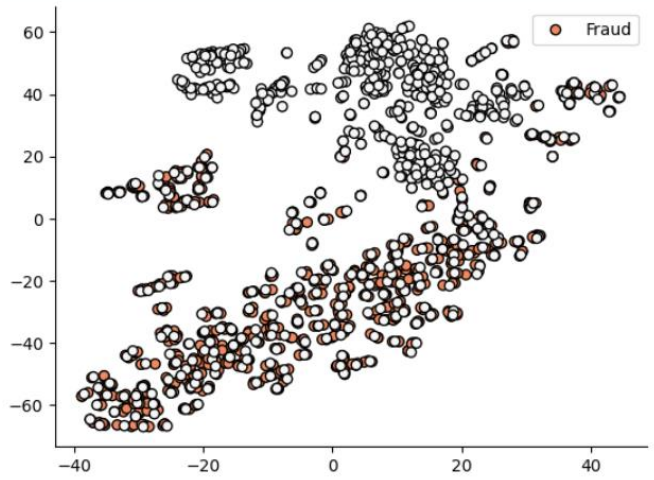
Below is visualization after dimensionality reduction of features using **t-Distributed Stochastic Neighbor Embedding (t-SNE)**



Near miss 2



Near miss 1



Near miss 3

As seen in results near miss give high false positive. Below is best result from near miss under sampling

```
#Scaler : power transformer
#Under sampling : near miss 2
#features : all
#Outlier treatment : no
#Payments : added
```

:

	Algorithm	False Positives	False Negatives	Precision	Recall	F1 Score	Accuracy
0	DecisionTreeClassifier	191377	1	0.0	0.99	0.0	0.01
1	RandomForestClassifier	188561	2	0.0	0.99	0.0	0.02
2	XGBClassifier	191657	1	0.0	0.99	0.0	0.01
3	CatBoostClassifier	191850	1	0.0	0.99	0.0	0.01
4	LogisticRegression	153074	22	0.0	0.88	0.0	0.21
5	GradientBoostingClassifier	192005	1	0.0	0.99	0.0	0.01
6	GaussianNB	42364	119	0.0	0.33	0.0	0.78
7	MLPClassifier	182951	8	0.0	0.95	0.0	0.05

We have also applied Aiknn, cluster centroids and edited nearest neighbour in under sampling technique, but results were not satisfactory.

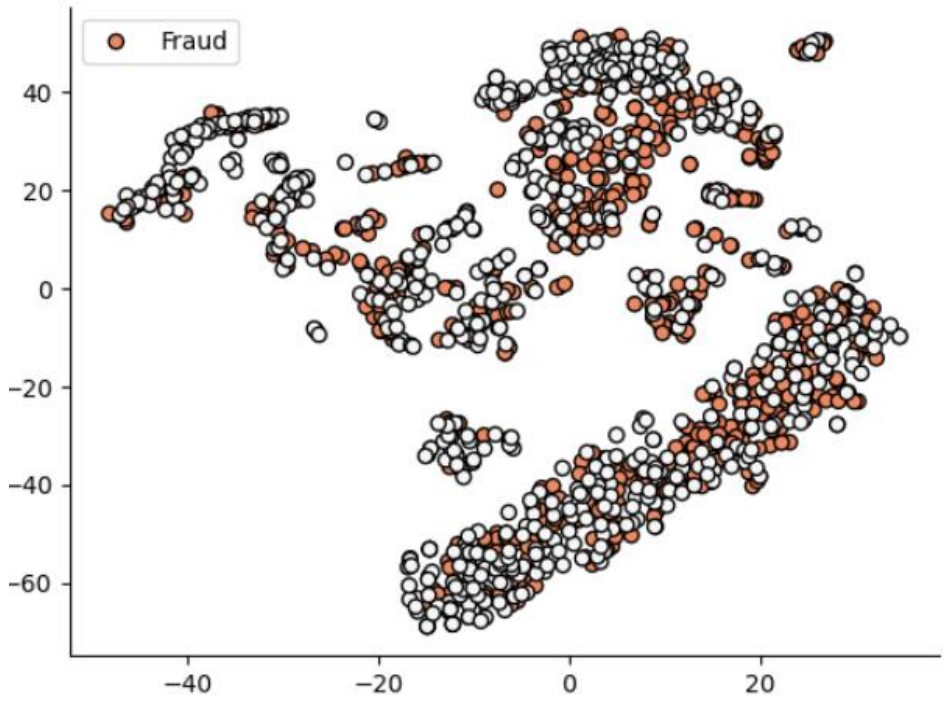
Random under sampling

Random under sampling involves randomly selecting examples from the majority class to delete from the training dataset.
This has the effect of reducing the number of examples in the majority class in the transformed version of the training dataset. This process can be repeated until the desired class distribution is achieved, such as an equal number of examples for each class.
This approach may be more suitable for those datasets where there is a class imbalance although enough examples in the minority class, such a useful model can be fit.
A limitation of under sampling is that examples from the majority class are deleted that may be useful, important, or perhaps critical to fitting a robust decision boundary.
Given that examples are deleted randomly, there is no way to detect or preserve “good” or more information-rich examples from the majority class.

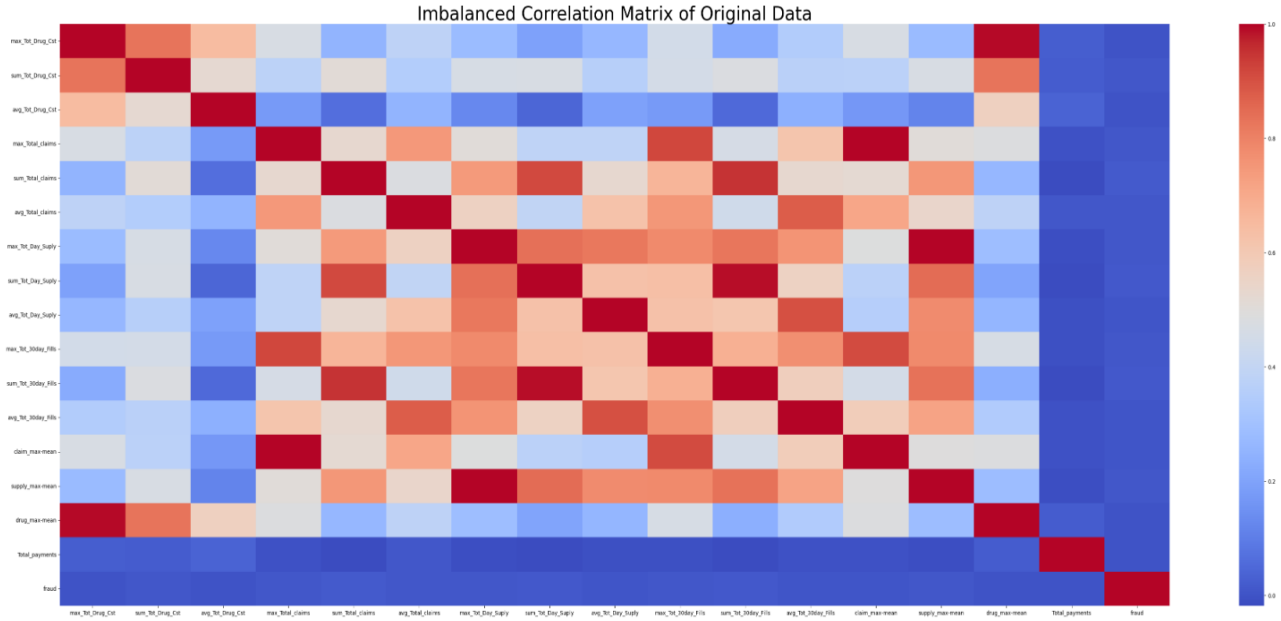
```
test_df.fraud.value_counts()
```

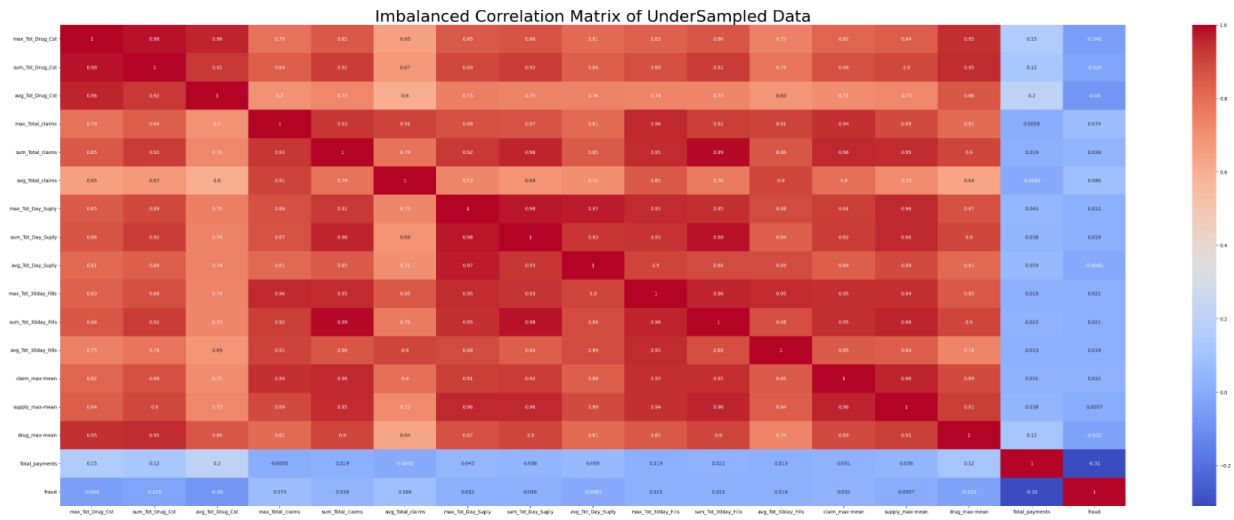
```
0    192835
1       177
Name: fraud, dtype: int64
```

Dimensionality reduction for visualization the result of under sampling.



.Correlation analysis of original vs under sampled dataset:





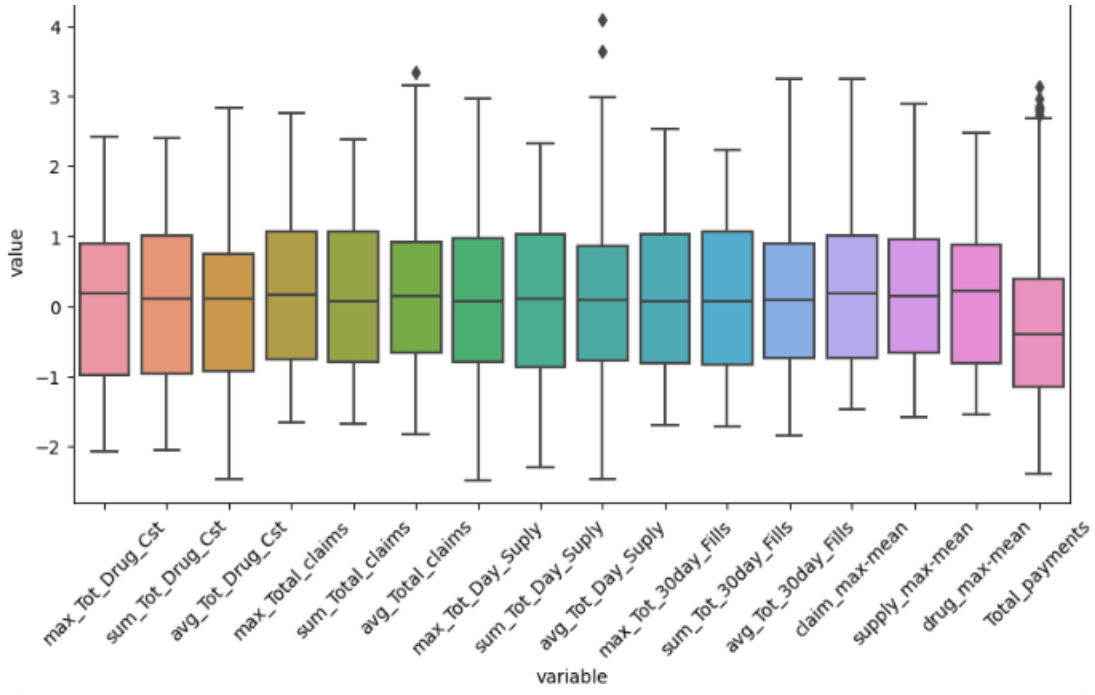
Under sampling has improved correlation between features and target variable. As it has given less false negative and false positive are in range**. Hence we have selected random under sampler in final training data set.** Below is sample result

```
#Scaler : power transformer
#Under sampling : random under sampler
#features : all
#Outlier treatment : no
#Payments : added
```

	Algorithm	False Positives	False Negatives	Precision	Recall	F1 Score	Accuracy
0	DecisionTreeClassifier	61029	57	0.0	0.68	0.00	0.68
1	RandomForestClassifier	52244	34	0.0	0.81	0.01	0.73
2	XGBClassifier	56925	37	0.0	0.79	0.00	0.70
3	CatBoostClassifier	52465	32	0.0	0.82	0.01	0.73
4	LogisticRegression	108447	42	0.0	0.76	0.00	0.44
5	GradientBoostingClassifier	53101	38	0.0	0.79	0.01	0.72
6	GaussianNB	22172	136	0.0	0.23	0.00	0.88
7	MLPClassifier	139708	19	0.0	0.89	0.00	0.28

5.5 Handling Outlier

Below is an image of outliers after random under sampling.as seen ouliers are present in sum total payments, sum total day supply, and total claims



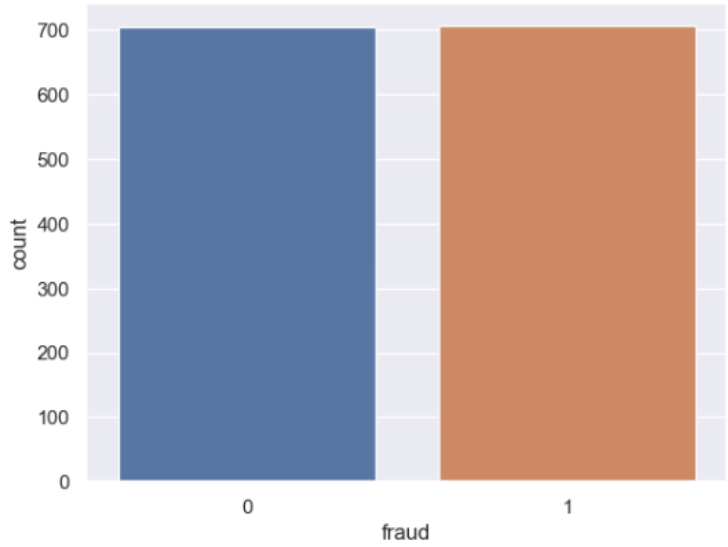
Its formula: Parameters: arr : [array_like] Input array or object for which Z-score is to be calculated.

Here IQR method tends to remove too many outlier and hence will go with Z-score method. score(arr, axis=0, ddof=0) function computes the relative Z-score of the input data, relative to the sample mean and standard deviation.

Z-Score method: IQR method tends to remove too many outlier and hence will go with Z-score method.

```
#Now target labels being same, lets see result
#Z-Score method : IQR method tends to remove too many outlier and hence will go with Z-score method.


z = np.abs(stats.zscore(df_under[scale_features]))

df_under_filtered = df_under[(z < 3).all(axis=1)]
print("Before outlier removal",df_under.shape)
print("Remaining after outlier removal",df_under_filtered.shape)
sns.set(style="darkgrid")
ax = sns.countplot(x="fraud", data=df_under_filtered)
```

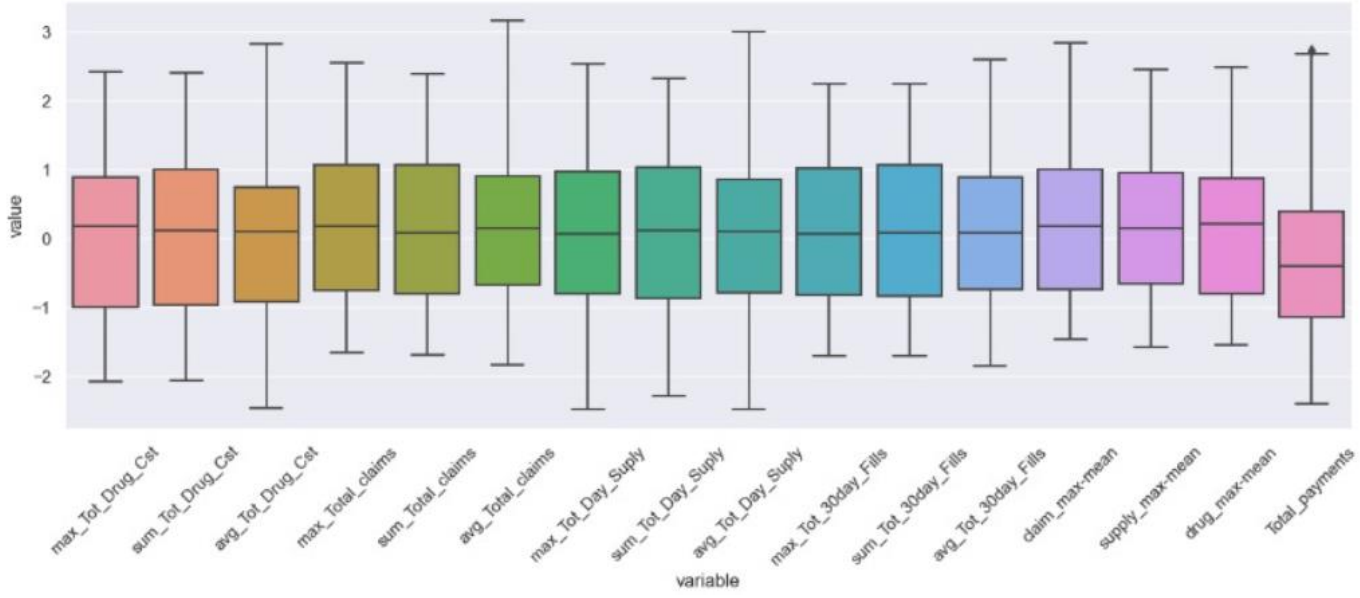
```
Before outlier removal (1418, 149)
Remaining after outlier removal (1410, 149)
```



No change in target features on dropping values more than z score.

After outlier removal plot

```
#after outlier treatment

plt.figure(figsize=(15,5));
sns.boxplot(x="variable", y="value", data=pd.melt(df_under_filtered[scale_features]));
plt.xticks(rotation = 45);
```



Chapter 6

Model Building

The modelling process was divided into two main parts: Building different model of machine learning to find out best algorithm and hyper parameter optimization of selected model.

6.1 Building Different Model

In Machine Learning, we have different classification algorithm. Few of them were basic algorithms such as Naive-Bayes, Decision Tree, and Logistic Regression. Also based on this basic algorithm we have different ensemble algorithms namely Random Forest, Boost, Ad boost, Support Vector Machine, Bagging Classifier, Gradient Boosting. We also implemented Multilayer Perceptron algorithm and logistic Regression.

Base line model

Models without sampling, scaling and outlier treatment. Models were not able to detect fraud NPI's due to heavy unbalanced target attribute. Hence had high false negative.
Below are the results.

Algorithm	FP	FN	Precision	Recall	F1	Accuracy
RF	1	158	0.95	0.11	0.19	0.99
GB	8	176	0.11	0.01	0.01	0.98
XG	1	172	0.83	0.03	0.05	1
CatB	0	168	1	0.05	0.1	1
AdaBoost	15327	145	0	0.18	0	0.92
DT	201	157	0.09	0.11	0.1	1
NB	15327	145	0	0.18	0	0.92
LogR	19	177	0	0	0	1
MLP	0	177	0	0	0	1

Model building with sampling techniques

Then we build ML models with various sampling techniques. There was minor improvement in results but still not satisfactory.

MODEL	sampling	FP	FN	precision	recall	f1 score	accuracy
decision tree	SMOTE	831	156	0.02	0.12	0.04	0.9
cataboost	ADASyn	191	171	0.03	0.03	0.03	1
gradient boosting classifier	Smote	23	176	0.04	0.01	0.01	0.9
cataboost	ADASyn	191	171	0.03	0.03	0.03	1
MLP	ADASYN	0	177	0	0	0	1
XG-boost	Random under sampler	14908	80	0.01	0.55	0.01	0.92
random forest	Random under sampler	65403	45	0.03	0.78	0.1	0.7
gradient boosting classifier	Smote	23	176	0.04	0.01	0.01	0.9

Random under sampler gave better result as compared to other sampling techniques. To improve model performance we performed hyper parameter optimization on machine learning models.

6.2 Hyper parameter Optimization

A Machine Learning model is defined as a mathematical model with a number of parameters that need to be learned from the data. By training a model with existing data, we are able to fit the model parameters.

However, there is another kind of parameters, known as hyper parameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Machine learning models are not intelligent enough to know what hyperparameters would lead to the highest possible accuracy on the given dataset. However, hyperparameter values when set right can build highly accurate models, and thus we allow our models to try different combinations of hyperparameters during the training process and make predictions with the best combination of hyperparameter values. Some of the hyperparameters in Random Forest Classifier are n estimators (total number of trees in a forest), max depth (the depth of each tree in the forest), and criterion (the method to make splits in each tree). n estimators set

to 1 or 2 doesn't make sense as a forest must have a higher number of trees, but how do we know what number of trees will yield the best results? And for this purpose, we try different values like [100, 200, 300]. The model will try all three of the given values and we can easily identify the optimal number of trees in our forest.

```
[ ] rf_clf  =  GridSearchCV(RandomForestClassifier(),random_grid, n_jobs = -1)

    rf_clf.fit(Xx_train, yy_train)
    #rf_clf.estimators_[0]
```

```
[ ] print(rf_clf.best_params_)

    {'bootstrap': True, 'max_depth': 60, 'max_features': 'auto', 'min_samples_leaf': 6, 'min_samples_split': 2, 'n_estimators': 400}
```

```
[ ] print(rf_clf.best_estimator_)

    RandomForestClassifier(max_depth=60, max_features='auto', min_samples_leaf=6,
                           n_estimators=400)
```

6.2.1 Random Search and Grid Search

Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly. This can be simply applied to the discrete setting described above, but also generalizes to continuous and mixed spaces. It can outperform Grid search, especially when only a small number of hyper parameters affects the final performance of the machine learning algorithm. In this case, the optimization problem is said to have a low intrinsic dimensionality. Random Search is also embarrassingly parallel, and additionally allows the inclusion of prior knowledge by specifying the distribution from which to sample.

Grid Search uses a different combination of all the specified hyperparameters and their values and calculates the performance for each combination and selects the best value for the hyperparameters.

This makes the processing time-consuming and expensive based on the number of hyperparameters involved. In GridSearchCV, along with Grid Search, cross-validation is also performed. Cross-Validation is used while training the model. As we know that before training

the model with data, we divide the data into two parts – train data and test data. In cross-validation, the process divides the train data further into two parts – the train data and the validation data.

```
[ ] rf_clf  =  GridSearchCV(RandomForestClassifier(),random_grid, n_jobs = -1)

    rf_clf.fit(Xx_train, yy_train)
    #rf_clf.estimators_[0]
```

```
[ ] print(rf_clf.best_params_)

    {'bootstrap': True, 'max_depth': 60, 'max_features': 'auto', 'min_samples_leaf': 6, 'min_samples_split': 2, 'n_estimators': 400}
```

```
[ ] print(rf_clf.best_estimator_)

    RandomForestClassifier(max_depth=60, max_features='auto', min_samples_leaf=6,
                           n_estimators=400)
```

```
[ ] rf_res = predict_and_evaluate(rf_clf, X_test, y_test)
```

Algorithm	False Positives	False Negatives	Precision	Recall	F1 Score	Accuracy
GridSearchCV	58027	32	0	0.82	0	0.7

Fig: Implementing the Grid search

The traditional way of performing hyperparameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a hold-out validation set. Since the parameter space of a machine learner may include real-valued or unbounded value spaces for certain parameters, manually set bounds and discretization may be necessary before applying grid search.

After we have implemented all this algorithm to find out the best performing model which will further tuned to get best model. And then this model will be integrated with Django framework. Here we have attached excel sheet of record of all algorithms.

```
results
#Full features
```

	Algorithm	False Positives	False Negatives	Precision	Recall	F1 Score	Accuracy
0	RandomForest	55956	31	0.0	0.82	0.01	0.71
1	XGBoost	51074	34	0.0	0.81	0.01	0.74
2	CatBoost	55275	37	0.0	0.79	0.01	0.71
3	LogisticRegerssion	57220	46	0.0	0.74	0.00	0.70

Figure: Different algorithms with their best metrics

From above chart we can see that Random Forest is the best choice among all other machine learning model. Now this model will be used for hyperparameter optimization.

Chapter 7

Deployment

Spark Streaming

Apache Spark Streaming is a scalable fault-tolerant streaming processing system that natively supports both batch and streaming workloads. Spark Streaming is an extension of the core Spark API that allows data engineers and data scientists to process real-time data from various sources including (but not limited to) Kafka, Flume, and Amazon Kinesis. This processed data can be pushed out to file systems, databases, and live dashboards. Its key abstraction is a Discretized Stream or, in short, a DStream, which represents a stream of data divided into small batches.

DStreams are built on RDDs, Spark's core data abstraction. This allows Spark Streaming to seamlessly integrate with any other Spark components like MLlib and Spark SQL. Spark Streaming is different from other systems that either have a processing engine designed only for streaming, or have similar batch and streaming APIs but compile internally to different engines. Spark's single execution engine and unified programming model for batch and streaming lead to some unique benefits over other traditional streaming systems.

Apache Kafka

Apache Kafka is a distributed data store optimized for ingesting and processing streaming data in real-time. Streaming data is data that is continuously generated by thousands of data sources, which typically send the data records in simultaneously. A streaming platform needs to handle this constant influx of data, and process the data sequentially and incrementally.

Kafka provides three main functions to its users:

- Publish and subscribe to streams of records
- Effectively store streams of records in the order in which records were generated
- Process streams of records in real time

Kafka is primarily used to build real-time streaming data pipelines and applications that adapt to the data streams. It combines messaging, storage, and stream processing to allow storage and analysis of both historical and real-time data

Apache Airflow

Apache Airflow is an open-source tool to programmatically author, schedule, and monitor workflows. It is one of the most robust platforms used by Data Engineers for orchestrating workflows or pipelines. You can easily visualize your data pipelines' dependencies, progress, logs, code, trigger tasks, and success status.

With Airflow, users can author workflows as Directed Acyclic Graphs (DAGs) of tasks. Airflow's rich user interface makes it easy to visualize pipelines running in production, monitor progress, and troubleshoot issues when needed. It connects with multiple data sources and can send an alert via email or Slack when a task completes or fails. Airflow is distributed, scalable, and flexible, making it well suited to handle the orchestration of complex business logic.

Hadoop Distributed File System (Hdfs)

HDFS is a distributed file system for storing very large data files, running on clusters of commodity hardware. It is fault tolerant, scalable, and extremely simple to expand. Hadoop comes bundled with HDFS (Hadoop Distributed File Systems).

When data exceeds the capacity of storage on a single physical machine, it becomes essential to divide it across a number of separate machines. A file system that manages storage specific operations across a network of machines is called a distributed file system. HDFS is one such software.

Integration with Django

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. With the help of Django, HTML and CSS, we build interactive user interfaces for our project's Medicare Fraud Detection system.

In this, we have designed two py files. First view.py and second url.py to display the contents of a text file on the Django template. On the predication page, displayed all the fraud NPI numbers.

We have provided all the path details in the urls.py python file for redirection from one page to another.

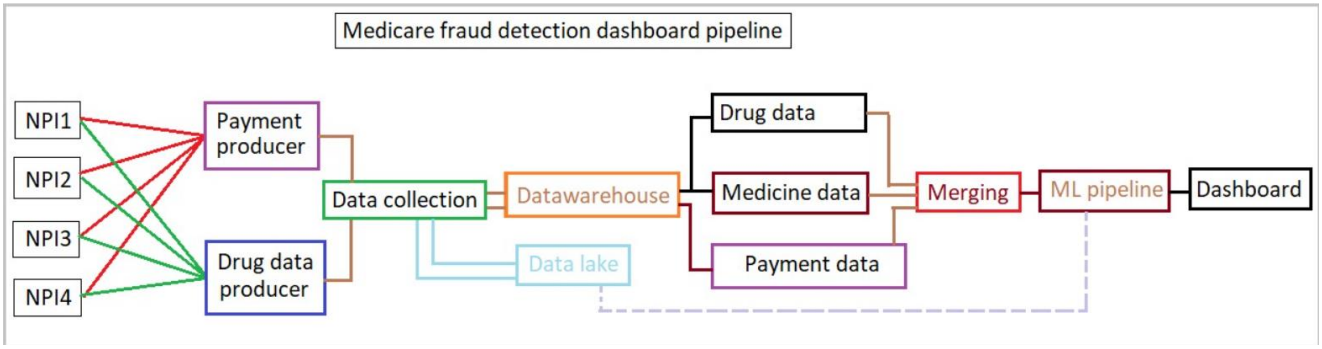
url.py

```
from django.contrib import admin
from django.urls import path

from . import views
urlpatterns = [
    path("admin/", admin.site.urls),
    path('new_file', views.read_file, name="new_file"),

]
```

Views, In Django views are Python functions that take http request and return http response.A web page that uses Django is full of views with different tasks and missions views are usually put in a file called views.py located on your app folder.

Once one of the URL patterns matches, Django imports and calls the given view, which is a Python function (or a class-based view). The view gets passed the following arguments: An instance of HTTP Request If the matched URL pattern contains no named groups, then the matches from the regular expression are provided as positional arguments. All function we have created in under views.py python file. In this file we write all functions and methods that we are going use. This file contain all working code for project.

.



Data producers

NPI holder will make claims, Payment authority and drug authority will have data for all claims.
Hence payment data producer and drug data producer will be our data producers.

They will generate data in form of event and send via Kafka streams.

Data consumers

Servers running 24x7 to receive data from **data collection.**
spark streaming job:

1. Consume and pre-process the data.
2. Dump data in to data warehouse (Mongo dB).
3. Dump data to data lake (Hadoop).

ML pipeline

Once every month pipeline will run,
It will read data from data warehouse.
Transform data:

1. Sort drug data.
2. Sort payment data
3. Generate medicine columns
4. Merge the three datasets.

Feature engineering:

1. Scale the attributes.
2. Encode the speciality attribute.

Predictions

Feed the data to pre trained ML model. And generate predictions of Fraud NPI numbers.
Also, the data in data warehouse can be cleaned once the predictions are done.
And past data from data lake can be read with hive tables, and be used to

1. Improve the performance of model.
2. Data analysis
3. Other business needs

Dashboard

Will read the updated NPI and display on dashboard with Django framework.

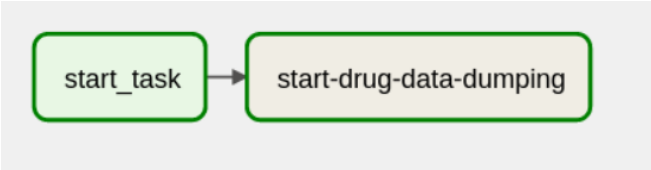
Airflow:

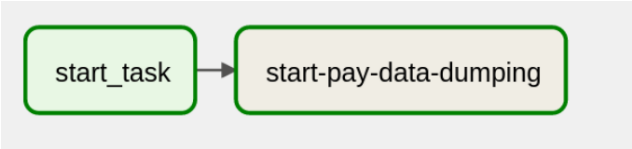
It will orchestrate the data pipeline and trigger the ML pipeline and its dependencies once every month.

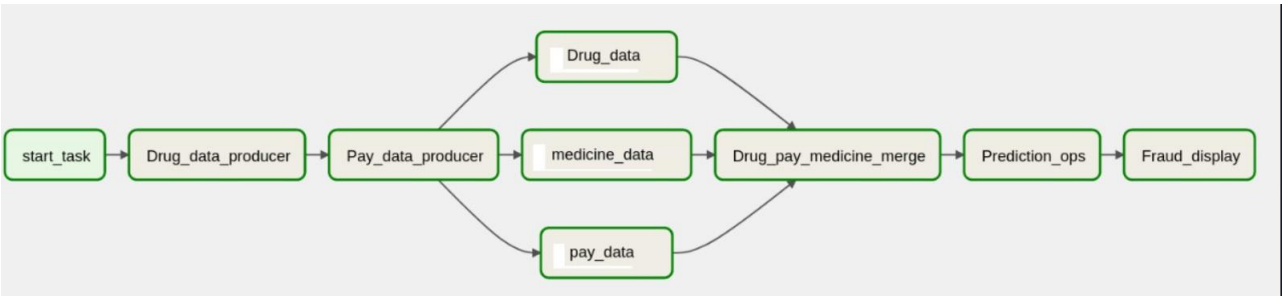
In project implementation

Data collection

This are brokers/servers which are on 24x7 and waiting for events (Row of data) from data producers.
Servers have spark structured streaming session running.
Why spark streaming not Kafka consumer? Because, spark is a unified engine, which can pre-process the received event, store the events to data base (mongo dB) and to data lake (Hadoop HDFS).







Data producer

Drug authority and pay authority are producing data from sample respective dataset.

Every row or every claim is an event.

Drug authority is sending events on Kafka topic “drug data.”

Payment authority is sending events on Kafka topic “pay_data.”

As data is limited to number of rows in excel.

Data producer will stop sending data once end of file is reached.

As they have a limit on reduce time, in our use case we have used data producer as a trigger to start ML pipeline.

Drug_data_sort, Medicine_data_sort and pay_data_sort starts in parallel.

Drug data

Once the producers are finished sending data, Drug_data_sort starts.

It starts by extracting drug data from mongo db.

Selects the required features.

Groups data on NPI.

Stores data locally.

Medicine data

Once the producers are finished sending data, Medicine_data_sort starts.

It starts by extracting drug data from mongo db.

Selects the required columns

Constructs an empty data frame with all medicine columns present in training dataset.
Adds respective values from loaded data frame to all cells.
Stores data locally.

```
npi_index = 0
drug_index = 0
row_ind = 0
flag = "start"
while True:
    #we will go through each index and with index[0] i.e NPI changes we will exit inner loop
    npi =  df.index[npi_index][0]
    ck = df.index[drug_index][0]

    #setting all row values to 0 , to avoid null
    temp.loc[row_ind,:] = 0
    temp.loc[row_ind,"NPI"] = npi

    while npi == ck:
        #setting npi number to NPI column
        # temp.loc[drug_index,"NPI"] = ck

        #add values to dataframe if columns are present in list
        for c in range(3):
            col_name = df.index[drug_index][1] + "_" + df.columns[c]
            if col_name in drug_n:
                temp.loc[row_ind,col_name] = df.iloc[drug_index,c]
                temp.loc[row_ind,:]
        if drug_index < 34487:
            drug_index += 1
            ck = df.index[drug_index][0]
        else:
            flag = "done"
            break
    if flag == "start":
        npi_index  = drug_index
        row_ind +=1
    else:
        break

print("Transformation data")
```

Pay data

Once the producers are finished sending data, pay_data_sort starts.
It starts by extracting payment data from mongo db.
Selects rows with not null NPI.

Selects the required features (NPI, total payments)

Groups data on NPI.

Stores data locally.

Drug pay medicine merge

Reads output data of Drug_data_sort, Medicine_data_sort and pay_data_sort.

Merges the three datasets.

Stores the data locally.

Prediction ops

Loads output data of drug_pay_medicine_merge.

Performs scaling of selected numeric features.

Encoding speciality column.

Feeds data to ML model.

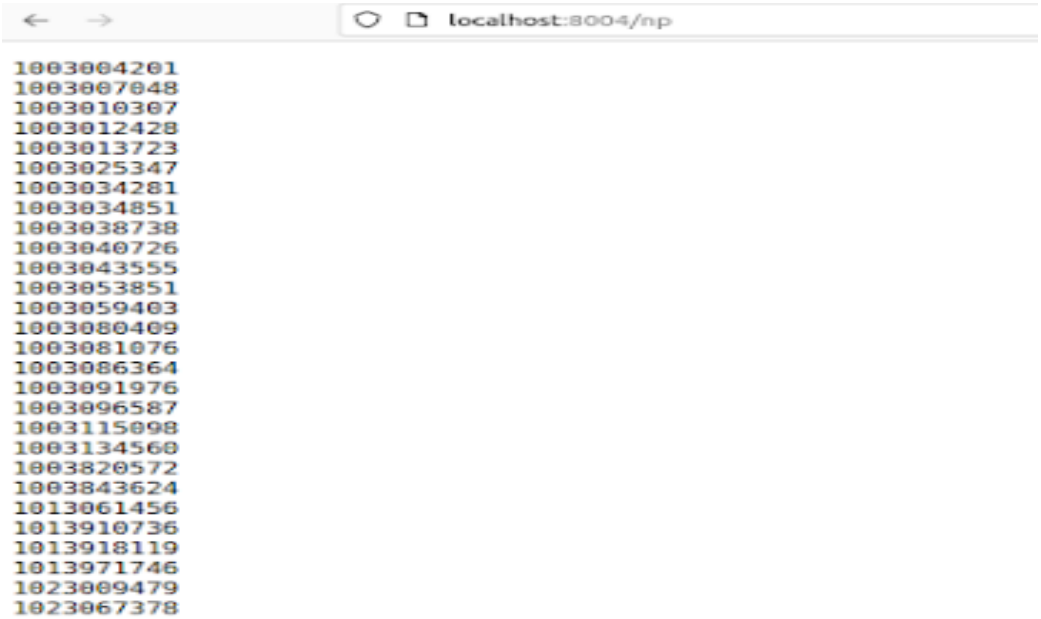
ML model output is joined with respective NPI and stored.

Fraud display

Reads the output of Prediction ops.

Starts Django server.

Displays the fraud NPIs.

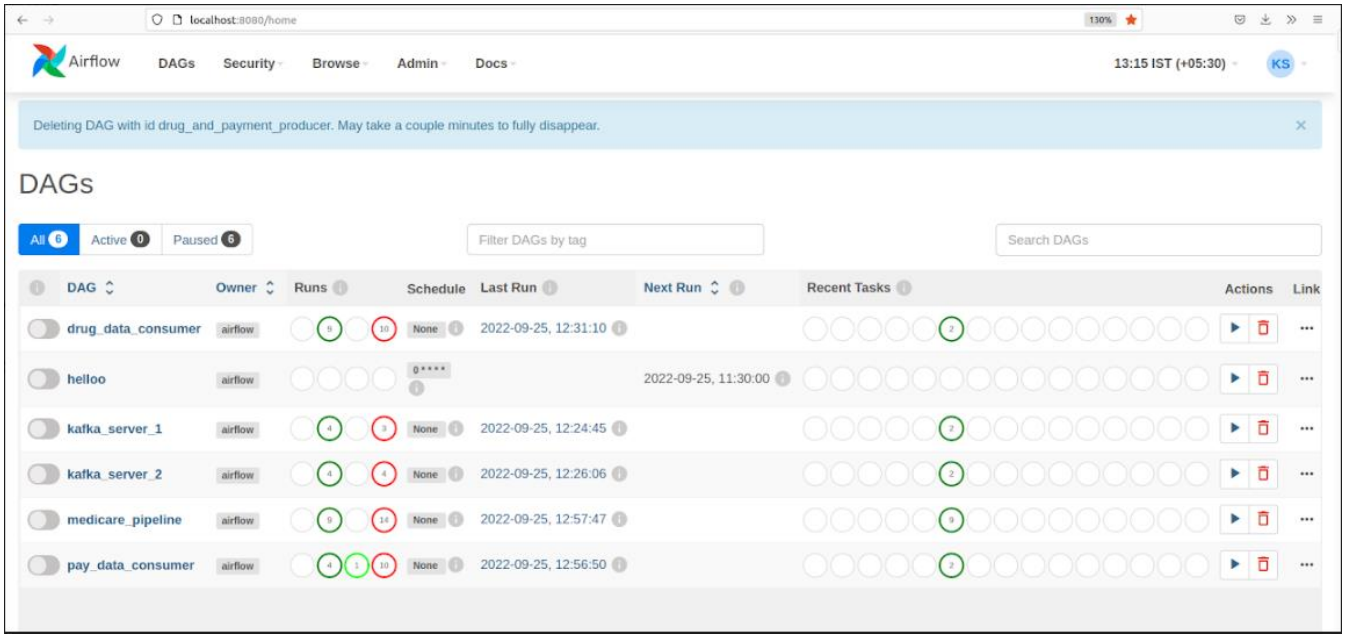


Airflow

It orchestrates the entire pipeline.

Servers and Medicare pipeline states can be managed and visualized with airflow.

Its directly the flow with dags in the direction specified.



Chapter 8

Conclusion

- With the increasing number of populations of over 65 in USA, Medicare Fraud Detection is essential.
- All types of Fraud Patterns have been covered.
- Most Fraud Cases committed are in bay area.
- Out of 5 Models Performed, best resulting model is Random Forest with Accuracy 72 % and recall 83%. The motivation for the study was to find the most efficient ML algorithm for the detection of Fraud NPI. This study compares the different evaluation metrics of ensemble algorithms such as Random Forest, Gradient Boosting, XG Boost, Bagging algorithms for predicting. The result of this study indicates that the Random Forest algorithm is the most efficient algorithm for the prediction of fraud. In future the work can be enhanced by training dataset on multiple year's data and optimizing model with tuning.

References

- Dataset from CMS datasets: https://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/Medicare-Provider-Charge-Data/Part-D-Prescriber.html •
- LEIE Datasets: https://oig.hhs.gov/exclusions/exclusions_list.asp
- https://www.fda.gov/Drugs/InformationOnDrugs/ucm079750.htm#collapseOn
- Dataset Downloads. (n.d.). Retrieved June 23, 2020, from -the-Data/Dataset-Downloads
- ”A Tour of Machine Learning Algorithms” by Jason Brownlee from : https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms
- Django Tutorial from W3school https://www.w3schools.com/django/index.php
- Kafka spark integration guide https://spark.apache.org/docs/2.2.0/streaming-kafka-0-10-integration.html
- ”Alternative Hyperparameter Optimization Technique You need to Know” from analytics Vidhya on https://www.analyticsvidhya.com/blog/2020/09/alternativehyperparameter-optimization-technique-you-need-to-know-hyperopt/
- https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- Sampling techniques : https://imbalanced-learn.org/stable/references/index.html#api.
- Mongo powerbi https://www.mongodb.com/docs/bi-connector/current/connect/powerbi/
- Spark streaming https://spark.apache.org/docs/latest/streaming-programming-guide.html