# Abstract

Fraud is a continuous problem that affects more than 2.8 million consumers every year (FTC, 2022). The use of statistical tools and data analysis has been helping to ease some of these difficulties, but problems persist. In this analysis, we are looking at synthetic consumer fraud in Europe in 2021 and seeing if using Naïve Bayes techniques and using smaller sample sets calculated via student’s t-test. This technique will help improve the detection of fraud and be comparable to using a smaller dataset to help save time, processing power, and storage space.

# Research Question

The simple question that this analysis aims to answer is “Can we detect fraudulent charges in a transactional dataset?”. After narrowing that question down, we could also ask “How well does Naïve Bayes work at predicting fraudulent charges in a transactional dataset?” The hypothesis is that yes, we will be able to predict fraudulent charges using a Naïve Bayes analysis. The answer to the second question is yes, we will be able to select an adequate sample size to accurately predict fraudulent charges in the dataset.

# Data Collection

For our data collection method, I just found the dataset on Kaggle (Kaggle, 2022). The author of this dataset created this large dataset as a simulation of fraudulent data in order to allow open access to the information and for the sake of practice and development of techniques and analysis. This dataset contains 11 columns that each provide some information on the transaction taking place. The step represents a unit of time where 1 step is 1 hour. Type is the type of online transaction. Amount is the amount in the transaction. nameOrig is the customer starting the transaction. oldbalanceOrg is the balance before the transaction. newbalanceOrig is the balance after the transaction. nameDest is the recipient of the transaction. oldbalanceDest is the initial balance of recipient before the transaction. newbalanceDest is the new balance of the recipient after the transaction. isFraud is flagged if the transaction is fraudulent, and isFlaggedFraud is if the system detected the transaction as fraudulent. The main advantage of this is that it’s anonymous and provides a massive dataset that can test problems within the field of ‘Big Data’ such as managing such large datasets and finding meaningful patterns within the data. The one downside that occurs from this is that since its systemically created, all these calculations will be more theoretical and lacking any real tangible real world affect. While the analysis is real, the techniques are real, and the exploratory analysis is real, the underlying data is not. However, this shouldn’t discredit the analysis since the data could very easily be swapped with real data and perform just as well and would work in any type of fraud detection set.

# Data Extraction and Preparation

For this analysis we were able to pull the data directly from Kaggle and use it for preparation. We used Kaggle’s built in terminal in order to have direct access and fast use of the file since downloading and uploading the actual datafile took a lot of time and difficulties when using other notebooks like Google’s Colab or Jupyter. To pull the dataset we simply pulled the file directly from the Kaggle cloud and saved it as “df” and promptly displayed the head of the dataset to make sure it was worked correctly. First, we just want to import our libraries.

import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import GaussianNB

Our next step in our cleaning process will be to pull the data into a variable that we can work with to perform cleaning and analysis on

df = pd.read\_csv("../input/online-payments-fraud-detection-dataset/PS\_20174392719\_1491204439457\_log.csv")  
df.head()

Table

Description automatically generated

We observed the head to make sure that the data that was pulled by the file is correct and is reading data. We then want to check the shape to make sure that all columns and rows are being pulled over.

df.shape



After observing our shape, we can see that all rows and columns have been pulled over. Our next step is going to be to drop a couple columns that won't really be necessary for the analysis. We will be dropping the 'step', 'nameOrig', and 'nameDest' columns since those won't be needed. Once these columns are dropped we can start our analysis.

df = df.drop(['step', 'nameOrig', 'nameDest'], axis = 1)

# Analysis

The full purpose of this analysis is to see how accurate a smaller, statistically significant, sample size can be when compared to an entire big dataset. We will do a little visualization of the dataset and then we will begin to apply naive bayes theorem to these datasets to try and predict fraud. Naive bayes works by taking previous results and using it as a predictor for the next events. Naive bayes works well if all the components are independent of each other.

We are starting off our data cleaning by checking if there are any null values within the dataset.

df.isnull().sum()

A picture containing text

Description automatically generated

Luckily this dataset appears to contain no null values. Our next step will be to delete duplicate rows within the dataset.

df\_with\_dup = df.copy()  
df = df.groupby(df.columns.tolist(),as\_index=False).size()  
df.shape



After deleting the duplicate rows, we managed to maintain roughly 6.2 million recorded transactions to check. The next few steps are just a quick visualization of the dataset by seeing what kind of transaction occurred as well as creating a bar graph to visualize the amount of each type of transaction that's occurring since the numbers by themselves can be a little difficult to understand.

df.type.value\_counts()

Text

Description automatically generated

plt.figure(figsize = (15,8))  
plt.ticklabel\_format(style='plain', axis='x')  
ax = df.type.value\_counts().plot(kind = 'barh')

Chart, bar chart

Description automatically generated

Our next step in our analysis is just some quick checks to understand what we are looking at.

df.info

Table

Description automatically generated

df.columns



After getting an idea of our information and the columns in the dataset, we then need to do a little more cleaning to get it ready for analysis. We will be changing our type of transaction into a numerical column so that our machine learning system will be able to analyze it. While naive bayes will work with categorical data, in order to calculate, we need to convert it into numbers. For this I just created a simple dictionary that can go through the datafile and replace the categories with numbers.

di = {'CASH\_IN': 1, 'CASH\_OUT': 2, 'PAYMENT': 3, 'TRANSFER': 4, 'DEBIT': 5}  
  
df\_replace = df.replace({'type': di})  
df\_replace.head()

Table

Description automatically generated

After printing out the head, we can see that the replacement was a success. Now we are going to calculate the actual fraud that is happening in the dataset in order to see what kind of numbers we will be working with.

df\_replace.loc[df\_replace['isFraud'] == 1].amount.sum()  
# 11533226537.16 --- 11.5 bil $ of fraud (1.02%)



df\_replace.loc[df['isFraud'] == 0].amount.sum()  
# 1131576795552.79 --- 1.1 tril $ of no fraud



As we can see, there is roughly 1.1 trillion dollars in transactions in the dataset. The number of fraudulent charges comes out to around 11.5 billion dollars which is approximately 1.02% of the transactions are fraudulent. This already is not very bad, but 11.5 billion dollars is still a lot of money and the more that can get clawed back would be nothing but beneficial for an organization.

Our first test is going to be a correlation matrix. This is going to help us find relationships between the factors that are included within our dataset. After the correlation matrix, we will create a heatmap to visualize the correlations to help find patterns that could help us detect anything.

df\_replace.corr()

Table

Description automatically generated

samp\_corr = df\_replace.corr()  
plt.figure(figsize = (15,12))  
sns.heatmap(samp\_corr, xticklabels = samp\_corr.columns, yticklabels = samp\_corr.columns)

Chart

Description automatically generated

samp\_corr['isFraud']

A picture containing text

Description automatically generated

After observing the heatmap, we can see that there is a strong relationship between the type of charge and the original old and new balance. This makes a lot of sense because you’d expect the numbers moving between the 2 to be very similar if they're transferring funds. The other part on the heatmap that shows a little bit of correlation from the heatmap is the destination of the old and new balance. To get a better understanding of this, we decided to run a correlation analysis with all sections being correlated to the 'isFraud' column. We can see that when we check the correlations, it appears that the closest correlation to check if a charge is fraudulent, is the amount. Our amount to isFraud correlation is 0.073 which is still very small but is the closest to 1 in the bunch. Because of this we decided to pull out all the fraudulent charges and perform an analysis on those alone. We wanted to see if running a basic machine learning script on the factors that are correlated to fraud would give us a good result. The following few steps are what was done to prep the data for a correlation naive bayes analysis.

df\_replace.loc[df['isFraud'] == 1]

Table

Description automatically generated

df\_samp\_fraud = df\_replace.loc[df['isFraud'] == 1]  
df\_samp\_fraud.head()

Table

Description automatically generated

df\_replace['isFlaggedFraud'] = np.where(df\_replace['isFraud'] == 1, 1, 0)

After creating our datafile with the fraudulent data set from the correlation calculation, the next step was to build our model. the 'makeModel' function below was created to produce a train/test split on whatever datafile we choose to input. The model selected was the 'GaussianNB()' function from sci-kit which is using naive bayes to help build a prediction model. We also built a 'testValues' array which are values that we can input to see if our model works based on just a quick test.

model = GaussianNB()  
testValues = np.array([[3, 9839.64, 170136, 160296.36, 0, 0, 0]])   
  
def makeModel(dataFile, features):  
 X = dataFile[['type', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', 'isFlaggedFraud']]  
 y = dataFile[['isFraud']]  
 y = y.astype('int')  
   
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 42)  
 #print(X\_train, X\_test, y\_train, y\_test)  
 #model = GaussianNB()  
 x = model.fit(X\_train.values, y\_train['isFraud'].values)  
 print(model.score(X\_test.values, y\_test.values))  
 print(model.predict(features))  
 return x

makeModel(samp\_corr, testValues)

Text

Description automatically generated with medium confidence

makeModel(df\_replace, testValues)

Text

Description automatically generated

makeModel(df\_samp\_fraud, testValues)

Text

Description automatically generated with medium confidence

We decided to create 3 quick models to test our predictability and accuracy. Our first model used the correlation sample dataset and used our test values for validation. The results are interesting since the accuracy was 100%, which indicates an overfitting (Shraddha Goled, 2021). This does predict that the charge isn't fraudulent, which is correct, but since it was overfitted, it shouldn't be used since the over fitting doesn't leave room for the actual model to learn and make decisions based on previous data and won't achieve generalization and won't be good in more natural data. When we use the original dataset to train, we get a 99.34% accuracy with a correct prediction. The accuracy of this model shows that it's not quite overfitting so it can still work as a general-purpose model, plus it was able to accurately predict if the event was fraudulent or not. The only issue with this model though is that it uses all the data present as a test so it's not exactly clear if the prediction is just referencing a correct row in the dataset or if it actually learned something in its training. The final test was just a 'what if' scenario where we simply tested only the data that was fraudulent. This resulted in an overfitted training set and managed to predict that the charge was fraudulent when it wasn’t.

# seems like testing specifically the only fraud part doesn't work very well

# replaced all 'isFraud' and 'isFlaggedFraud' in order to try a new machine learning test

#df\_orig = pd.read\_csv("../input/online-payments-fraud-detection-dataset/PS\_20174392719\_1491204439457\_log.csv")  
df.head()

Table

Description automatically generated

With this dataset, the size makes it quite inhibitive to do quick analysis with. Having 6.2 million records to check adds a large amount of compute time and takes up a ton of space. One of the solutions for this analysis was to create a sample set to see if we could perform an analysis that is comparable to using the full dataset, but using far less memory and computing time.

In order to find a proper sample size, we decided to apply the Chochran formula (Glenn D. Israel, 2003) where is the sample size that we need. Z is our t score, which was looked up in a table (T-Table, 2021) for 99% confidence with 1% margin. This value came out to be 2.576. p is the variance within our sample. To maximize this, we will use 0.5. q is just 1-p, so this will also be 0.5 as well. our e variable will be our margin of error. This is going to be 1% which comes out to a value of 0.01.

After using Cochran's formula, we manage to find that we need 16590 random samples in order to create a sample set that is 99% confidence with 1% margin of error, which is a very far cry from the original 6.2 million samples and would allow us to compute our results faster and just as accurately. I decided to put that to the test and to create 2 new datasets. One of the datasets is a sample set which is 16590 randomly selected samples. The other dataset is a set that has the sample set removed so it still contains roughly 6.2 million records, but it's missing the ones that were randomly selected.

# resource: https://www.tarleton.edu/academicassessment/documents/samplesize.pdf  
df\_samp = df.sample(16590, random\_state = 42) # 99% confidence with 1% error --- 16534 sample size --- calculation from hand calc, 16546

# rips the sample data from original dataset for comparison of actual data  
df\_merge = pd.merge(df, df\_samp, indicator = True, how = 'outer').query('\_merge=="left\_only"').drop('\_merge', axis = 1)  
#df\_merge = pd.concat(df\_orig, df\_samp, indicator = True, how = 'outer').query('\_merge=="left\_only"').drop('\_merge', axis = 1)

print(df.shape)  
print(df\_merge.shape)  
print(df\_samp.shape)

Text

Description automatically generated

df\_merge = df\_merge.replace({'type': di})  
#df\_merge = df\_merge.drop(['step', 'nameOrig', 'nameDest'], axis = 1)  
df\_merge.head()

Table

Description automatically generated

df\_samp = df\_samp.replace({'type': di})  
#df\_samp = df\_samp.drop(['step', 'nameOrig', 'nameDest'], axis = 1)  
df\_samp.head()

Table

Description automatically generated

As we can see from above, we were able to rip our sample dataset and create another dataset with the remaining tables. using this we can compare the original dataset, the set without the samples, and the sample set. Below are the models being created and tested using a known test value array.

makeModel(df\_merge, testValues)

Text, letter

Description automatically generated

makeModel(df\_samp, testValues)

Text

Description automatically generated

Using the dataset with the sample ripped out, we have an accuracy of our naive bayes prediction model of 99.25% and was able to accurately predict whether the test case was fraud or not, which is very good. Now when we test our data sample the results aren't as good. We get an accuracy of 53.83%, which is slightly greater than a coin flip, however it still manages to predict the right outcome if a situation is fraudulent or not. Our next step was to test out some sample cases to see if we accurately predicted a few of the correct cases. We had to reshape the datafile to an array for this to work, as seen below, and then randomly selected rows in the dataset to test.

print(df\_samp.iloc[[12]])

A picture containing text

Description automatically generated

df\_samp\_array = df\_samp[['type', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', 'isFlaggedFraud']].to\_numpy()  
#df\_samp\_array.reshape(-1,1)

print(df\_samp\_array[0])



makeModel(df\_merge, df\_samp\_array[[11111]])  
print(df\_samp[['isFraud']].iloc[[11111]])

Text

Description automatically generated

df\_merge\_array = df\_merge[['type', 'amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest', 'isFlaggedFraud']].to\_numpy()

print(df\_merge\_array[0])



makeModel(df\_samp, df\_merge\_array[[1111]])  
print(df\_merge[['isFraud']].iloc[[1111]])

Text

Description automatically generated with medium confidence

model.predict(df\_merge\_array[[1111]])



# Conclusion

For this analysis we asked ourselves if we could accurately predict if a transaction was fraudulent or not by applying a Naive Bayes analysis to the dataset and train a sci-kit machine learning tool to detect that. We also decided to try seeing if we could speed up analysis time by pulling out a smaller sample set to apply our analysis to and see if it'd perform similarly to help save computation time and space. After performing our analysis that is demonstrated above, we were able to create a system that uses the bulk dataset to create a 99.25% accurate predictor of whether a transaction is fraudulent or not. This came at a cost to the size and time that it took to analyze almost 6.2 million records in order to create our model. We also created a system using a statistically selected sample set to see if we could speed up our time while also saving space. This sample model came back at 53.83% which isn't fantastic, but it was able to predict our fraudulent charges correctly for all our test cases.

With this analysis, it is advised that using a naive bayes prediction analysis does a fairly good job at acting as a prediction for if a charge is fraudulent or not. However, it would be advised that another type of test should be performed to act a redundancy check on our analysis. Another good way to analyze the dataset would be to perform a logistic regression on the fraudulent dataset. The logistic regression would help act as a classifier for the dataset to help find patterns that relate to fraudulent charges as opposed to using previous inputs to act as a predictor such as naive bayes (Saishruthi Swaminathan, 2018). Another possible course of action that could be performed with this dataset would be to perform a decision tree analysis on it. A decision tree would be able to create a set of decisions that could be easily traced in a graphical format to check if certain factors are the exact measures to use for predicting if a charge is fraudulent or not (Sci-Hit, 2022).

# Resources

<https://www.ftc.gov/news-events/news/press-releases/2022/02/new-data-shows-ftc-received-28-million-fraud-reports-consumers-2021-0> , FTC, 2022

<https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc> , Saishruthi Swaminathan, 2018

<https://scikit-learn.org/stable/modules/tree.html> , Sci-Kit, 2022

<https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset> , Kaggle, 2022

[https://analyticsindiamag.com/why-is-overfitting-so-demonized/#](https://analyticsindiamag.com/why-is-overfitting-so-demonized/):~:text=%E2%80%9CDue%20to%20overfitting%2C%20a%20model,unseen%20data%20in%20real%20situations , Shraddha Goled, 2021

<https://www.tarleton.edu/academicassessment/documents/samplesize.pdf> , Glenn D. Israel, 2003

<https://www.tdistributiontable.com/> , T-Table, 2021