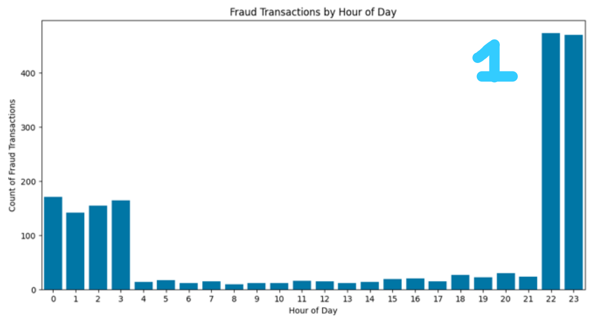
Kaggle Fraudulent Transactions Competition Report

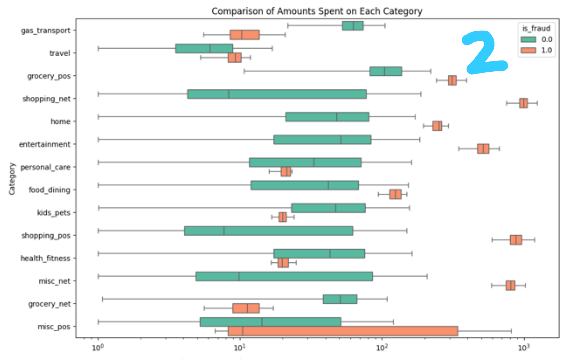
I approached this competition by thoroughly exploring the dataset, feature engineering, and targeted model selection. The dataset used in this project consists of many transactional records, including a binary target variable that indicates whether a transaction is fraudulent or not. In the feature engineering process, I focused on creating new variables that could capture the patterns and characteristics of fraudulent transactions, such as transaction variance, amount, and time.

Feature Selection Pt.1

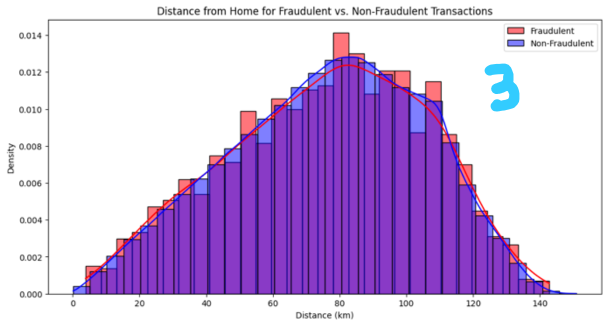
This process Is detailed heavily in the Data Exploration pt.1 notebook; here are the main points. My main goal was to explore the data to see what types of patterns I could see. I began by using simple prints to see the number of fraud counts, the times they happened, and how many were happening per user. While this was interesting, it didn't really tell me much. From here, I thought I could maybe create a tree for specific customers. However, I came to the conclusion that this wasn't a good idea because it was not scalable, the data was sparse (not every customer had many transactions), and It reduced the generalizability of the model to unseen customers. After that, I decided to look for trends in the data time with regard to the days and hours of transactions.



From the first plot **[1]**, "Fraud Transactions by Hour of Day," I could see that there's a significant spike in fraudulent transactions at certain hours, incredibly late at night. So, I created a feature based on this. I also took a look at the "Fraud Transactions by Day of Week," which showed some minor variability across the days, with Sunday having notably higher counts. While not the most conclusive graph, I decided to make a feature check to see if it was a weekend or not.



After that, I looked at the categories customers were spending in. I did this by plotting boxplots for the transaction amounts per category along with the fraudulent ones. Specific categories had clearly different fraud transaction ranges from the regular range, so I created a feature based on this.



I then decided to examine a geographical feature, plotting the distance of a transaction location from the customer's home address. From that graph **[3]**, the distribution of distances for fraudulent and non-fraudulent transactions seems to overlap significantly. This suggests that distance alone might not strongly predict fraud in the dataset. However, it's also possible that distance could still be a contributing factor when combined with other features.

My line of thinking was that a transaction that's atypical for a customer's pattern, such as one that's far from home in combination with an unusual transaction amount, might be more suspicious than any of those factors considered in isolation. So, I created features that keep track of the transaction amount/distance deviation, which is how much each customer's transaction/distance deviates from the average transaction amount(based on category) and distance(based on cc_num). From here, I wanted to try using a few models to see how well the features perform.

Fitting the Features to Models Pt.1

I wanted to use a pipeline similar to the one I used for the models in the Titanic Kaggle competition. This included trying KNN, Naïve Bayes, and Decision trees and then combining them together using a stacking classifier. Preparing the features was quite simple. I removed the non-numeric and identifier features, removed geolocation features(choosing to keep the derived ones instead), and kept behavioral features (like category_mean_amt and category_std_amt). More detailed explanations can be found in the notebook. After that, I had to use one-hot encoding to convert the categorical features into numerical ones that I could interpret.

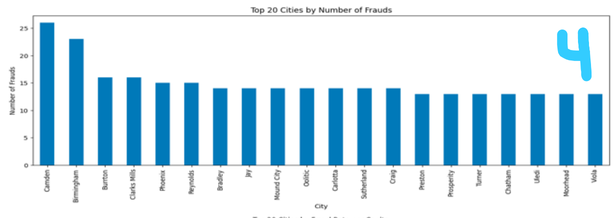
For KNN, I had to scale numerical features due to how the model uses distances to classify transactions. Then, I looped through the number of neighbors ranging from 1-21(arbitrary at the beginning) and plotted the accuracies. However, this took nearly 5 hours to run, leading to subpar results (an F-1 score of about 0.68). KNN tends to perform poorly when the features have a high multicollinearity coefficient, so I realized I needed features that cover more of the data. To confirm this suspicion, I ran PCA on my data set and found that about two features captured 95% of the variance. Which was quite disheartening, but it confirmed that I needed a broader range of features. However, before I moved on to find more features, I tried a decision tree classifier and got an F-1 score of about 0.71. I know naive Bayes works under the assumption that the variables are independent. However, from my work above, I could clearly see the variables are not, so I used the PCA'ed data and ran the model and got the worst F-1 of about 0.16, but that was expected. Finally, I tried a Stacking Ensemble Classifier and a Gradient Boosting Classifier. I used the stacking classifier because it combines diverse models by learning how to blend their strengths best using another model, and I used gradient Boosting because it builds a sequence of models in a way that each subsequent model focuses on correcting the mistakes of the previous ones. Even with this, I got a pretty bad F-1 score of about 0.45, so we can clearly see the problem seems to be a lack of variety in my features; I think while they were specific, they are not broad enough to capture the nuance in the data. This leads me to do some more data exploration.

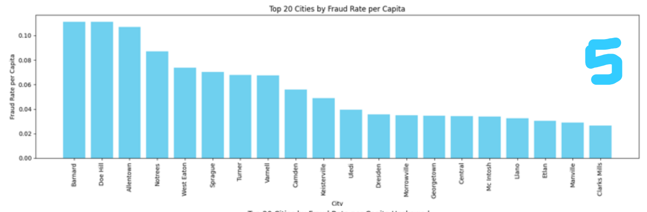
Feature Selection Pt.2

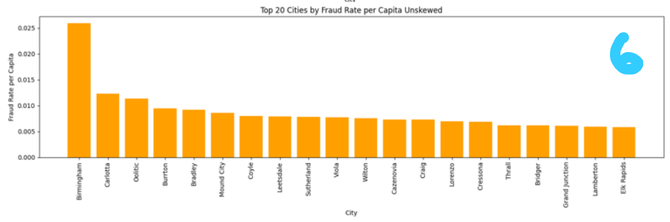
This process and all the associated graphs are detailed more in the Data Exploration Pt.2 Notebook. So now I want to capture risk scores based on state, city, job, and merchants. I started with states and cities. To start, I plotted the number of frauds occurring per state and city. However, the issue with this was that by doing this, it always seemed to be the case that bigger states/cities would have higher numbers of fraud just due to the population sizes. So, I decided to normalize the data, dividing fraud counts per state/city by their respective population sizes. But again, there was a problem here: cities with small population sizes would have much larger fraud rates just due to the nature of their size, which could lead to a misleading plot, so to combat this, I set minimum population sizes for cities, jobs, and merchants when calculating their fraud rates. However, I didn't have to do this for states because the populations were aggregated.

Here are the plots for cities: the first is un-normalized **[4]**, then normalized **[5]**, and then normalized with minimum population sizes **[6]**. Based on the graphs, I split the data into three quantiles to assign the risk scores. The top 33% were high risk, the middle 33% were medium risk, and the bottom 33% were low risk.

Finally, to ensure that my features were broad enough, I examined what categories were being spent on the most during at-risk hours. I noticed grocery_pos and shopping_net were more likely to be spent during these hours, so I made binary indicators that checked if these were being spent during high-risk hours.



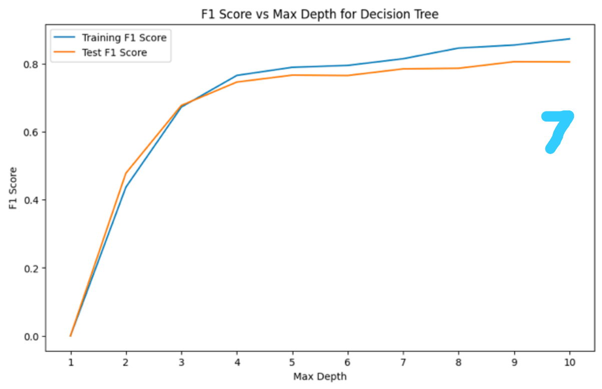




Fitting the Features to Models Pt.2

So, finally, I had a good set of features, which I think covered a good amount of the data. I spent the first half of the notebook pulling all the features together and applying them to the test set. For the risk scores, I applied the averages I calculated on the train set and applied them to the test set; for states/cities/jobs/merchants, I automatically applied a low risk to them as I didn't have any background information on them.

So, the pipeline I devised from the first fitting to models didn't seem to be working well, so I decided to restart. I noticed the decision trees seemed to be working the best. I believe this is because the data had a good number of indicator variables that the tree was able to make connections between. The main difference between how I ran it last time and this time is that I decided to keep more of the original features; I felt the tree was able to make more nuanced decisions with the more numerical information it had, so I decided to include zip, city population and the latitudes and longitudes of both the customer and the merchant.



Once the data was prepped, I looped through a depth range from 1 to 11(arbitrary) to see the best depth I could get. With this at a depth of 9 **[7]**, I got an F1 score of about 0.80, which was already better than before. This confirmed that the features weren't covering a large enough range of the data.

After that, I wanted to use a stacking classifier to combine this tree with another model. My research found that logistic regression captures variance in the data that the tree fails to capture. So, the following model I tried was logistic regression. The logistic regression model gave me an F-1 score of 0.74. I then used a stacking classifier to combine these models, but it performed quite poorly, so I removed it from the notebook. This led me to research methods that improve simple decision trees, leading me to an XGBoost algorithm. XGBoost builds upon the concept of decision trees by creating an ensemble of trees. It iteratively trains each tree to correct the errors made by the previous trees, resulting in a more accurate and robust model. Using XGBoost, I was able to get an F-1 Score of about 0.89 locally and 0.84 on Kaggle. At this point, I stopped working on the submission due to time constraints and other project deadlines. However, if I had more time, I would've tried other ensemble classifiers with different sets of features.

Conclusion and Struggles

I really enjoyed the entire process and found it quite fun! I struggled quite a bit at the beginning trying to come up with a feature set that captured the nuance of the data, but the more I looked into the data and thought about what exactly seemed to be causing fraudulent cases, the more interesting features I was able to come up with. At one point, which you can see in my Kaggle submission history, I was getting very low scores of around 0.0001, which was really weird. I spent quite a few hours trying to debug it and found that the cause was that I was scaling the training data for logistic regression and re-assigning it back to the training data variable, which then caused errors when I tried to run the XGBoost algorithm.

If I had more time, I would've loved to investigate the frequency of transactions between fraudulent transactions, as it seems quite reasonable to assume multiple fraudulent transactions may happen at the same time; beyond that, I wanted to look more into the geographical data to see if I could come up with any interesting features in that regard. I would've also liked to have explored different combinations of models and features to see if I could've gotten a better score.