G36: Detection of Fraud From Customer Transactions

Taaha Nazrul Islam Kazi, Nidhish Ganesh Kamath, Jonathan Adamic

University of Illinois

CS 412 Introduction to Data Mining

Arindam Banerjee

December 7, 2022

G36: Detection of Fraud From Customer Transactions

Abstract: In this report, we outline the work we've completed in exploratory data analysis, pattern mining, and classification of fraudulent transactions.

Introduction: Our project is inspired by the IEEE-CIS Fraud Detection Kaggle competition that originally took place in the Fall of 2019. The goal of the project is to explore data mining and classification techniques to create a transaction fraud detection system.

Motivation: Each year, millions of dollars are lost to fraudulent charges. Using data mining and classification techniques, patterns identified in fraudulent transactions can be used to detect and prevent fraud. We aim to build a model to detect such frauds. Additionally, our dataset has real world transactional & identity data. It is a heterogeneous mix of numerical & categorical data. The meaning of many attributes in the dataset are obscured (for privacy reasons); making it challenging to derive any inferences from domain knowledge.

Data: Our dataset contains fields for basic transactional information (such as amount and time), product code, multiple fields for card information, purchaser information (email domains, billing addresses, distance between billing and shipping address, device type used etc.). There are other attributes which are cryptically labeled as mentioned before. We only know[3] that they keep various counts (such as the number of addresses associated with a certain card), matching checks (phone area code with billing zip code, purchaser/recipient last name etc.), and certain "Vesta engineered rich features." The training data[2] consists of 590540 rows and 434 columns. Cursory analysis shows that some columns have empty rows and are sparsely populated (See Figure A below).

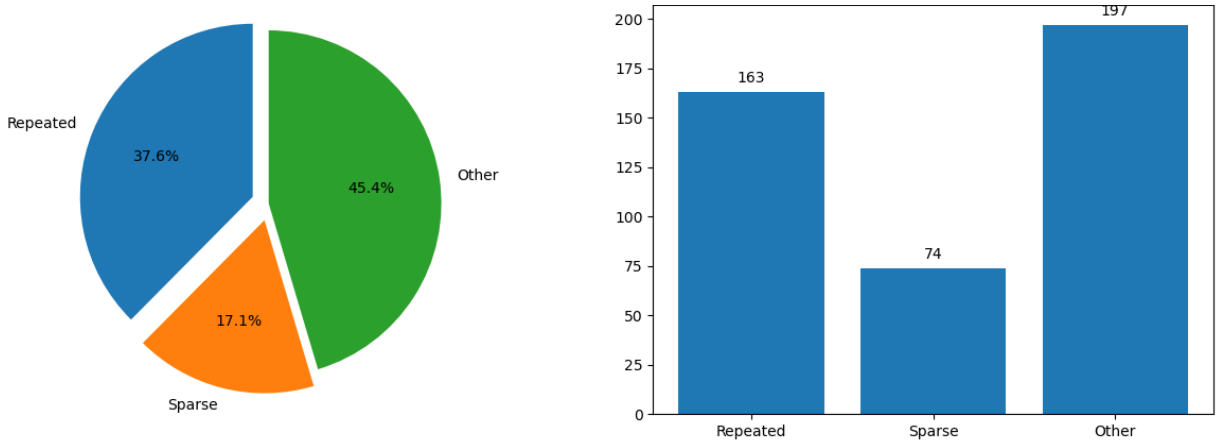


Figure A: Percentage & number of columns missing over 80% of values or containing 80% repeated values

Related work:

In this section, we review related work done in the field of fraud detection. Chen et. al.[6] used CatBoost for fraud detection. In this work, they applied feature engineering along with Catboost and achieved competitive scores. They also compared their approach to other approaches such as Naive Bayes and SVM and scored considerably higher on the ROC_AUC metric. Shaohui et. al.[7] proposed a transaction fraud detection model based on random forest. Their model outperformed benchmarks such as logistic regression and support vector machines, achieving an accuracy of 97.4% and an AUC ROC score of 92.7%. Also in a Kaggle notebook[10], the author used under-sampling and used a decision tree to classify

transactions. In another notebook[11], the author deployed Xgbm to classify the financial transactions as fraud or not.

Methodology: We have approached the problem by undertaking the following steps: First, we carried out Exploratory Data Analysis to get a deeper understanding of the data we are dealing with. This was followed by Pattern Mining, from which we tried to find out any interesting patterns in the dataset, which might help us in classification. Along with this, as our data set was huge, we carried out the task of Memory Compression to work with the data with minimum resources. After which we carried out PCA to explore if the data was easily separable. Then we ran a simplistic baseline to get a rough understanding of the data and the pipeline.
After completing the above tasks, we carried out a comparative study of different classification models and their performances. The detailed steps taken are mentioned in the Empirical results section.
Memory Compression: With a dataset this large, it was beneficial to compress the memory footprint by changing the data type of each column to be the smallest possible data type for numerical data. For example, if a column of number had a max value of 127, we could save significant memory space by converting data stored in that column to int8. To save us both time and memory, we used a public function written by Guillaume Martin[4], that carries out the process mentioned above on a given dataframe.

Exploratory Data Analysis: Once we loaded and optimized the dataset, we began our exploratory data analysis (EDA) by looking at the target label distribution. Across the entire dataset, approximately 96.5% of the transactions were non-fraudulent, leaving 3.5% that were fraudulent (see Figure B below). While sampling the dataset, we needed to take this wide class imbalance into consideration.

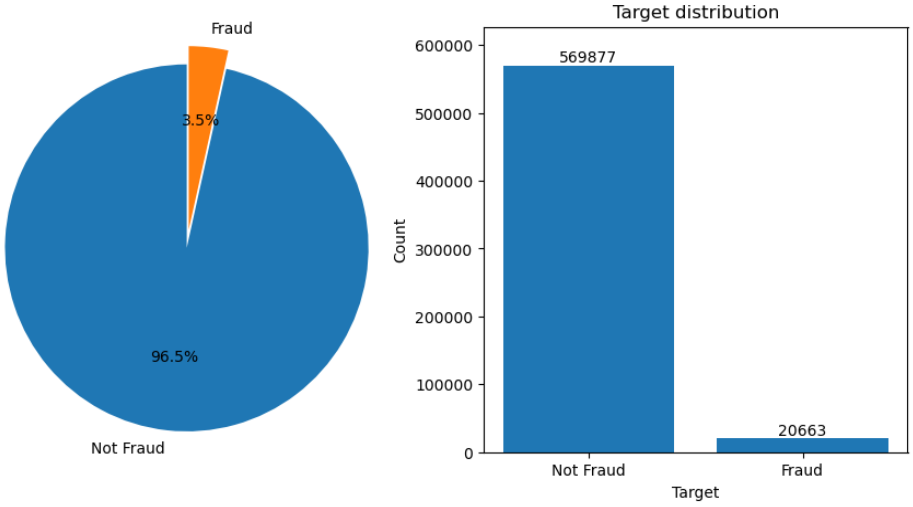


Figure B: Class Imbalance

Since most of the column meanings are obfuscated, the majority of column data distribution can only meaningfully be carried out on a small subset of the attributes. Despite this, there are still a number of interesting distributions we can analyze and visualize. Starting with the type of card used (ie, visa, mastercard, etc), we can see which cards are most popular, and how frequently fraud occurs in relation to each type–Discover cards had the highest fraud percentage at 8%, while Visa, Mastercard, and American Express tied with 3%. For fraud distributions based on the charge type–credit charges had 7% fraud, while debit had 2%. The transaction devices had little impact on fraud probability, with mobile being only 1% more likely than desktop. For further details refer to Appendix A figures 1,2 & 3.

Continuing our EDA we calculated the correlation between columns to see if we could prune some of the highly correlated attributes. We followed a method inspired by Omer Parlak[5]. By grouping attributes

based on their column category (ie, id columns, C columns, D columns, etc), we can efficiently compute and visualize correlations between attributes. Many of the C columns (values related with the payment card, whose real meaning are hidden) had the highest correlation with approximately 40% of the columns being more than 90% correlated. While id and D columns had the lowest correlations.

Pattern Mining: Conducting pattern mining on the dataset was rather tedious. The data was not structured in a format that is easily mineable. Paired with the fact that much of the data was missing, some simplifications had to be made. For our current approach, we run the FP Growth algorithm over a hand selected dataset of nine categorical attributes. Any missing values were replaced with the value of 'unknown{X}' where {X} is a one word description of the attribute. We ran FP Growth once for all fraudulent transactions, and once for all non-fraudulent transactions and then generated the association rules for each. By comparing which frequent itemsets and association rules are different between the two, we hoped to gain insight into which features are more likely to contribute meaningfully during classification.

itemset	Support (Fraud)	Support (Non-Fraud)
['W'] (a ProductCD)	0.434	0.643
['C'] (a ProductCD)	0.387	0.106 (not in top 15)
['M0'] (a Match attribute)	0.348	0.332 (not in top 15)
['M2'] (a Match attribute)	0.329	0.093 (not in top 15)

Table 1: Selected Data from Frequent Itemsets (Appendix B)

Looking at the results of our pattern mining (above and Appendix B), it's difficult to pick out specific itemsets that differ between the legitimate and fraudulent transactions. However, there are a few in particular that do stand out. The ProductCD (Product code–a letter identifier for which the real meaning has been hidden) of 'W' shows up in both, but with a lower support in the fraudulent transactions, while 'C' is frequent with a high support in only fraudulent transactions. Additionally, many of the fraudulent itemsets contain more itemsets with 'Matching' attribute tags. While 'matches' are described as appearing when values do match[3], it seems more likely that they actually occur when they don't match. Since the real meaning is hidden, it is possible this is the case.

Baseline Logistic Regression: We use logistic regression as our baseline model for this binary classification task. Logistic regression estimates the parameters in a linear combination and serves as a good first cut model.For our initial experiments, we have only considered the columns with numeric data. Then we trained a logistic regression model and measured the results using Area Under the Receiver Operating Characteristic Curve (AUC_ROC). Detailed results are mentioned in table 3.

Standardizing before Training: We standardize the features by subtracting the mean and scaling to unit variance. This led to an improvement of 5.2 points. (Increase from 78.61 to 83.81)

NaN value replacement technique: We experimented with replacing NaN with the mean of the column and with zero, The results were relatively similar. (83.81 for mean and 83.92 for 0)

Classes Balanced	Scaling Done	Value for replacing NaN	PCA applied	AUC_ROC
Yes	Yes	0	No	0.8392

Table 2: Best results for baseline Logistic Regression

Empirical Study:

We have carried out a comparative study of different classification models. Based on the insights we gained from the above sections, we followed a common data processing pipeline for training all the models. This included: Removing columns which had high correlation, and also removing columns that were very sparse. The pipeline also included steps for substituting NaN values and memory reduction for reduced resource usage. For training we used stratified K(10) fold Cross Validation across all models in the comparative study. We chose stratified sampling as our dataset is highly imbalanced. The seed for stratified k fold sampling was kept constant across all models as we intend to use the holdout validation scores for calculating the student's t test.We used Area Under the Receiver Operating Characteristic Curve(AUC_ROC) for scoring the models, as it is suitable for imbalanced datasets are used. For our empirical study, we examine Random forests, Xgboost and CatBoost classifiers.

Using Random Forests for classification:

We started with using a decision tree based on gini impurity reduction before grouping them, but noticed that though they had good accuracy (96.1%), they had low precision(44.5%) and recall (43.02%). A better way to use them is in random forests, which are ensemble methods that employ a collection of decision trees for classification. The performance of random forests comes from being able to aggregate the predictions of its constituent trees. When the trees are trained on different parts of the whole dataset using methods like bootstrapping, it adds diversity to the constituent trees. This is because each tree "learns" different aspects about the dataset and thus when combined give better predictions. Another factor that decreases the correlation in predictions between all the trees is that the candidate attributes/predictors chosen for optimally splitting a node in a given tree are randomly chosen as a subset of all the predictors available. They are also known to be robust to noise and outliers, owing to their diversity.

For our random forest algorithm, we adopt a CART based approach as described for a similar fraud detection application by Xuan et.al.[8] In their paper they have considered another approach where the data is split by comparing distance between records and two centers representing the class (isFraud). They found the CART based approach to work better on the basis of accuracy, precision, recall and F1-measure. For the implementation we used a set of libraries offered in the H2O Python module like the class H2ORandomForestEstimator. While employing the 0.632 bootstrapping method to sample the dataset, we followed iterations for the hyper-parameters as listed below with their performance metrics:

Iteration #	# of Trees (T)	# of candidates available at every node for split = m	Max depth of trees = d	Area under ROC curve (AUC)
1	100	100	20	0.87368
2	100	17	20	0.89885
3	100	14	20	0.89364
4	200	17	20	0.89694
5	200	17	60	0.93227

Table 3: Some hyperparameter tuning iterations for random forest (More iterations in appendix)

In iteration# 2 the value 17 was chosen for m based on a few sources referring to a relatively small[9] and possibly sqrt(total #of available attributes)[8] as the best possible m to provide discrete & thus better performing random forests. We had 270 attributes available for prediction after cleaning the data, so m

became sqrt(270)$^{0.5}$ which is ~17. Below 17, we observed a slight dip in the AUC metric. Higher number of trees in a random forest yielded better AUCs. Generally, the greater the maximum depth allowed for constituent trees, the better the performance of the random forest until a point when it starts to overfit.
Due to memory and runtime constraints, we restrict ourselves to a max depth of 60 and 200 trees. Based on the best value of AUC, we selected our hyperparameters for subsequent modeling.
Per our evaluation plan we ran a k-fold cross validation on our random forest. Following were the areas under the ROC curves for a 10-fold cross validation using the same partitions as our other classifiers (using the same random seed and stratified sampling): Table4

#	1	2	3	4	5	6	7	8	9	10
AUC	0.9330	0.9348	0.9335	0.9327	0.9340	0.9337	0.9340	0.9344	0.9327	0.9325

Table 4: Random Forests K fold validation results

Using XGBoostClassifier for classification:

Extreme Gradient Boosting is a boosting method which is a modified version of the GBM algorithm.
It is an ensemble learning algorithm that performs really well in classification tasks. For our task, we have initialized the xgboost classifier from the xgboost library. For our given project, we experimented with the following parameters: max_depth and sub sampling. Here max_depth defines the depth of the tree created, after which the model starts to prune the tree. We carried out hyper parameter tuning for the these two variables and the results are mentioned in the table below: Table 5

max_depth	sub_sample	Area under ROC curve (AUC)
12	0.5	0.9709
8	0.5	0.9672
12	0.9	0.9741

Table 5: Xgboost parameter selection results

Comparing rows 1 and 3, we see that increasing sub-sampling increases the ROC_AUC by 0.4%. And Comparing rows 1 and 2, we see that reducing max_depth leads to reduction in scores by 0.37%.
Note all the other parameters are kept the same for benchmarking from[10]As per our evaluation plan we ran a k-fold cross validation on our random forest. Following were the areas under the ROC curves for a 10-fold cross validation using the same partitions as our other classifiers (using the same random seed and stratified sampling): Table 6

#	1	2	3	4	5	6	7	8	9	10
AUC	0.9738	0.9791	0.9752	0.9766	0.9766	0.9763	0.9751	0.9768	0.9785	0.9759

Table 6: Xgboost K fold validation results

Using CatBoost Classifier:

Cat Boost is another boosting library like xgboost. It differs from it on the basis it forms symmetrical trees and it implements ordered boosting, because of which it prevents target leakage and overfitting. For our given project, we experimented with the following parameters: depth and bagging temperature. Here depth defines the depth of the tree created and bagging temperature. We carried out comparison for the these two variables and the results are mentioned in the table below: Table 7

depth	Bagging temperature	Area under ROC curve (AUC)
12	0.1	0.9650
8	0.1	0.9497
12	0.3	0.9663

Table 7: CatBoost parameter selection results

Comparing rows 1 and 3, we see that increasing bagging temperature increases the ROC_AUC by 0.13%. And Comparing rows 1 and 2, we see that reducing depth leads to reduction in scores by a 1.53%.
Note all the other parameters are kept the same for benchmarking from.[10]
As per our evaluation plan we ran a k-fold cross validation on our random forest. Following were the areas under the ROC curves for a 10-fold cross validation using the same partitions as our other classifiers (using the same random seed and stratified sampling):Table 8

#	1	2	3	4	5	6	7	8	9	10
AUC	0.9648	0.9648	0.9679	0.9676	0.9687	0.9673	0.9661	0.9680	0.9706	0.9670

Table 8: CatBoost K fold validation results

Classification Using a Neural Network: Our main focus was on the other classification methods, as such, our exploration of Neural Networks does not take into account many factors that should be considered for deeper analysis. We did not conduct k-fold cross validation for this classification method, so we will not be comparing it to other classification methods. The model is a simple 4 layer network using binary cross entropy and the Adam optimizer. We trained and tested a few combinations of the learning rate, and the batch size to see the effects they had on the model. We split the training data into three parts for training (80%) validation (10%) and testing (10%). We based each model on the resulting AUC ROC score. Learning rate had the most impact on the model. Values too large (>.05) did not converge, while values much lower (<.0005) converged too slow. We found .005 was optimal for our testing. Testing with batch sizes of 500, 1000, and 1500. We found the smaller batch size of 500 provided marginally better results (+/- 1%) than the other two. Our best model had an AUC ROC score of .8616. While our exploration of NN was rather shallow, it offers a vast potential for future research and analysis.

Model Selection:

We used the Student's t-test to check if the distributions are the same for the k-fold cross validation roc_auc values. As we are using a 10 fold cross validation strategy, the degree of freedoms for our calculation will be 9. Hence, at alpha= 0.05, for a symmetric distribution, the value for t0= 2.262. That is if our distributions are different then the t statistic value should be either > than 2.262 or < than -2.262.
We calculated the t-statistic between all the three models.
The pairwise t-statistics are: (RandomForest - xgboost) = -82.837, (xgboost - catboost)=41.411 , (catboost- Random Forest) =57.372
Hence we can confidently say that Xgboost is the best performing model followed by CatBoost and then Random forest.

Conclusion: For the given transaction dataset we have analyzed the data, carried out pattern mining and compared different classification methods.

References:

[1] Phua, C., Lee, V., Smith, K., & Gayler, R. (2010, September 30). "A comprehensive survey of data mining-based Fraud Detection Research". arXiv.org. Retrieved October 3, 2022, from https://arxiv.org/abs/1009.6119

[2] The link to the training data:
IEEE-CIS Fraud Detection. Kaggle. (n.d.). Retrieved October 3, 2022, from https://www.kaggle.com/competitions/ieee-fraud-detection/data

[3] Data Description Discussion:
https://www.kaggle.com/c/ieee-fraud-detection/discussion/101203

[4] Guillaume Martin - Memory saving function:
https://www.kaggle.com/code/gemartin/load-data-reduce-memory-usage/notebook

[5] Omer Parlak - 2-Correlation Analysis and Feature Importance
https://www.kaggle.com/code/omerparlak/2-correlation-analysis-and-feature-importance

[6] Y. Chen and X. Han, "CatBoost for Fraud Detection in Financial Transactions," 2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE), 2021, pp. 176-179, doi: 10.1109/ICCECE51280.2021.9342475.

[7] D. Shaohui, G. Qiu, H. Mai and H. Yu, "Customer Transaction Fraud Detection Using Random Forest," 2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE), 2021, pp. 144-147, doi: 10.1109/ICCECE51280.2021.9342259.

[8] S. Xuan, G. Liu, Z. Li, L. Zheng, S. Wang et.al.,"Random forest for credit card fraud detection", IEEE 2018.

[9] H Ishawaran, "The effect of splitting on random forests", Machine Learning 99, 2015 (Springer)

[10] https://www.kaggle.com/code/kirankunapuli/ieee-fraud-stacknet-on-gpu-lgb-xgb-cb

[11] https://www.kaggle.com/code/dataraj/eda-and-decision-tree-with-data-imbalancetackling

Appendix A

Exploratory Data Analysis Visualizations

Figure 1: Target distribution in Card type

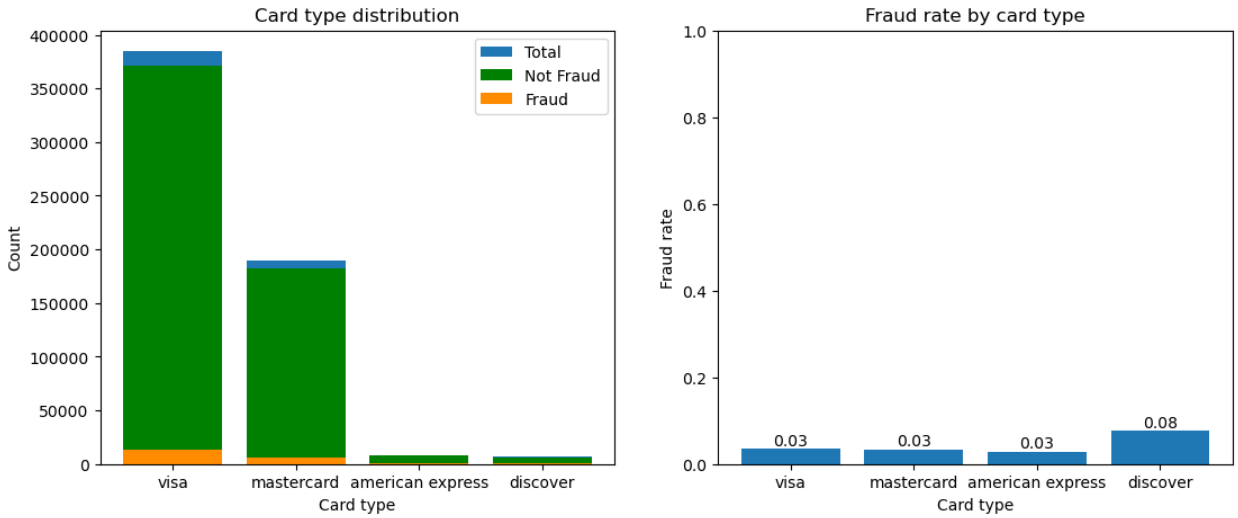


Figure 2: Target Distribution in Charge type

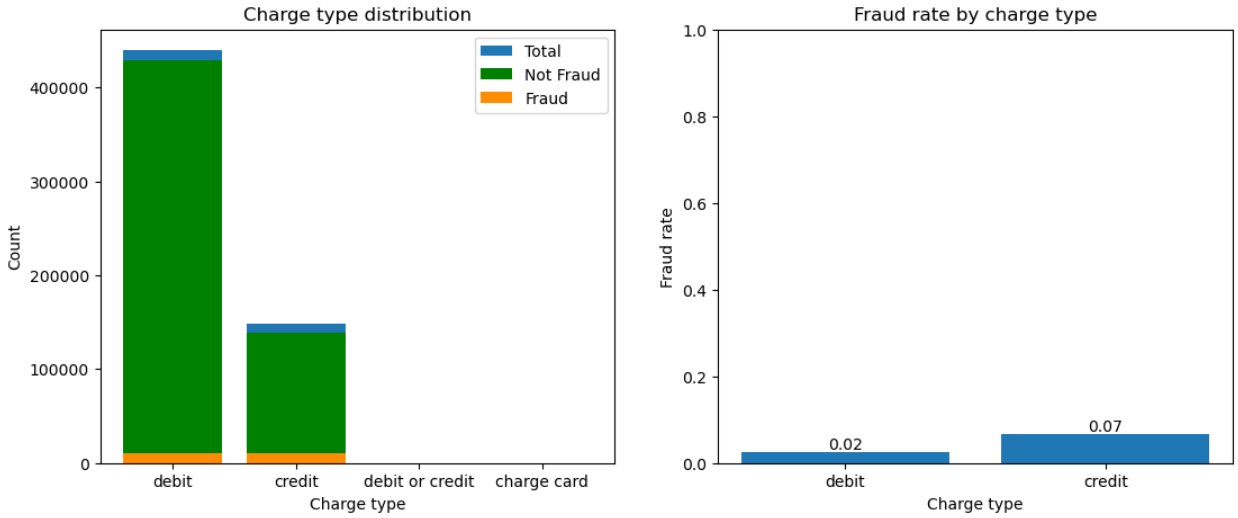


Figure 3: Target Distribution in Transaction Device

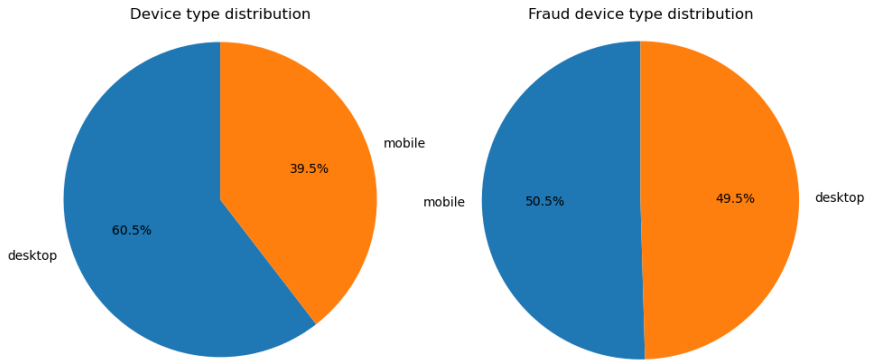


Figure 4: Transaction amounts for fraudulent transactions and their frequencies

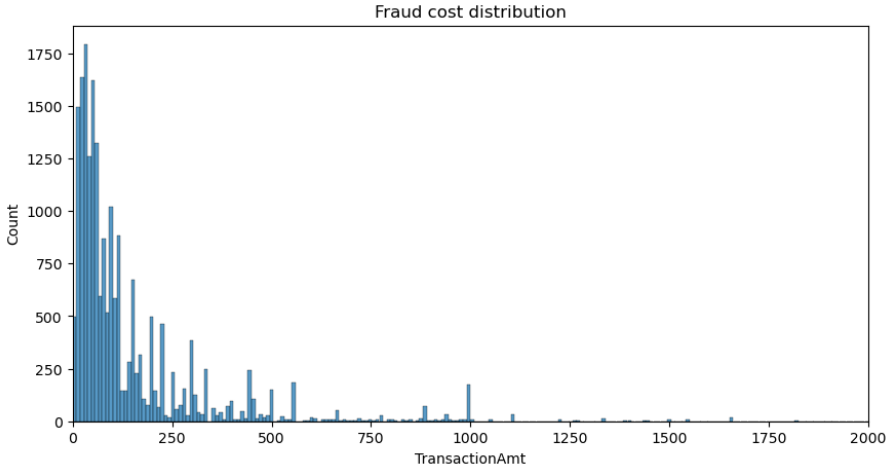


Figure 5: Count of fraudulent transactions in the range of $0 - $260

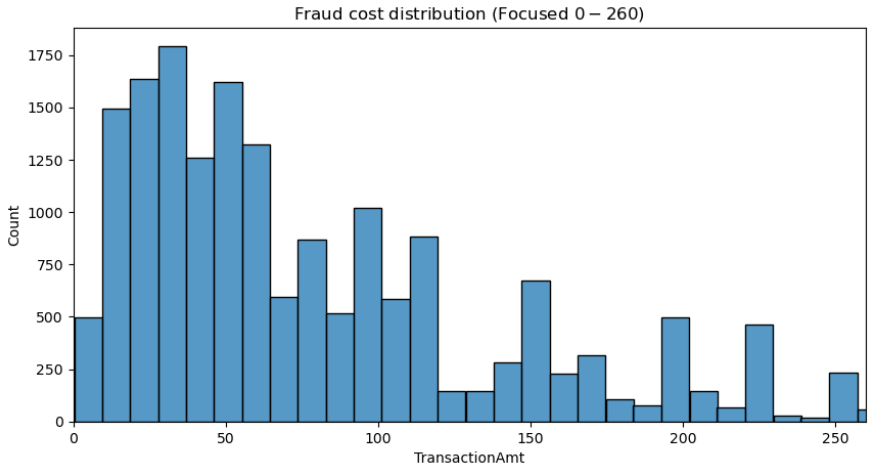


Figure 6: Target Distribution Against Transaction Delta

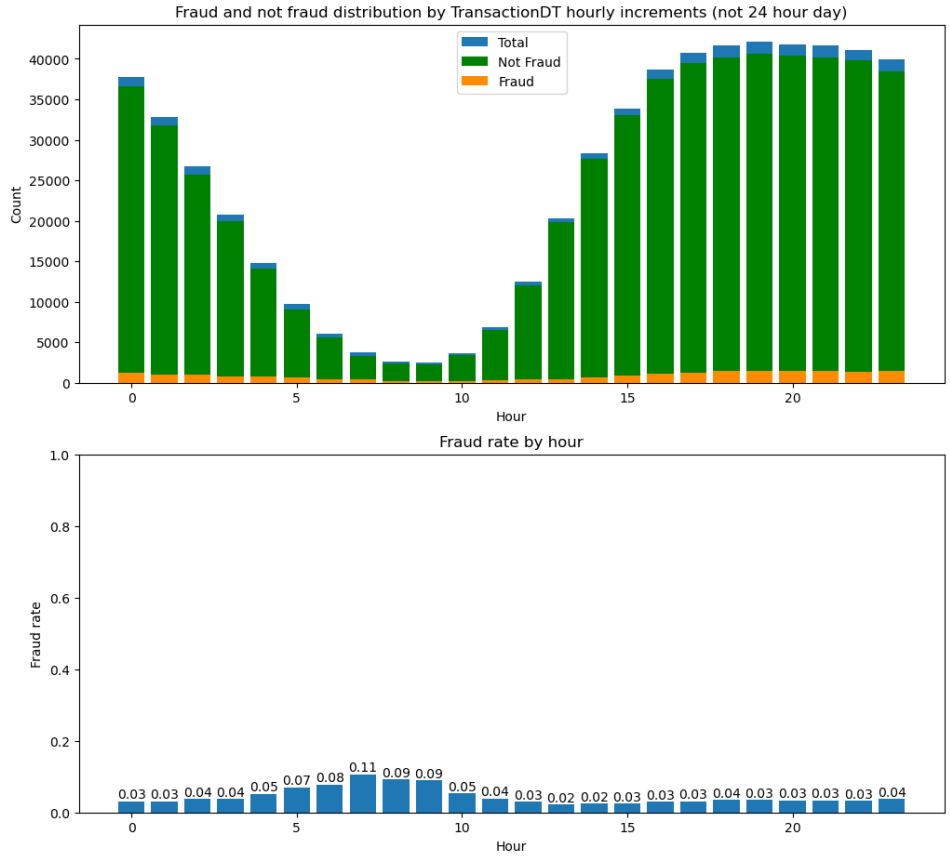


Figure 7: Percentage and Number of Columns With Sparse or Repeating Values

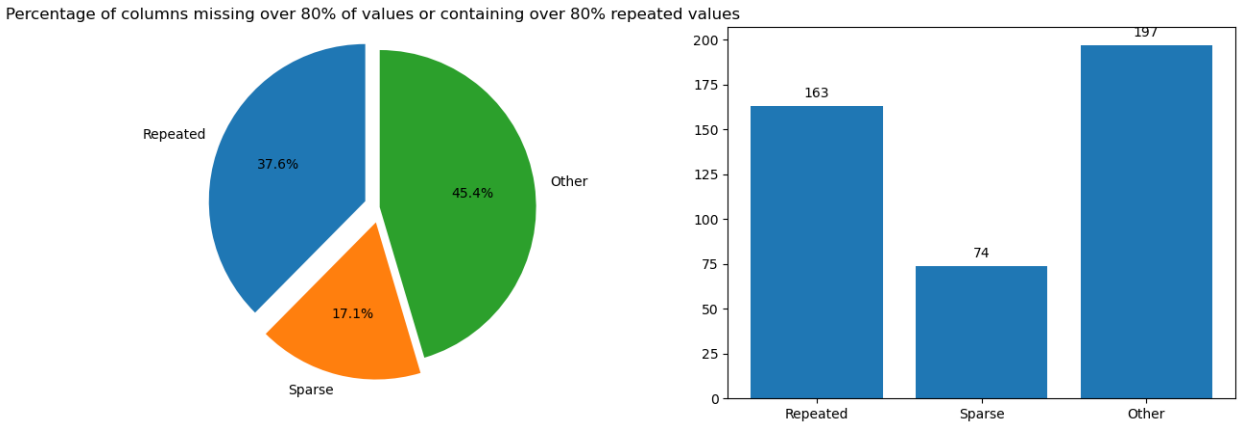


Figure 8: C-Column Correlation

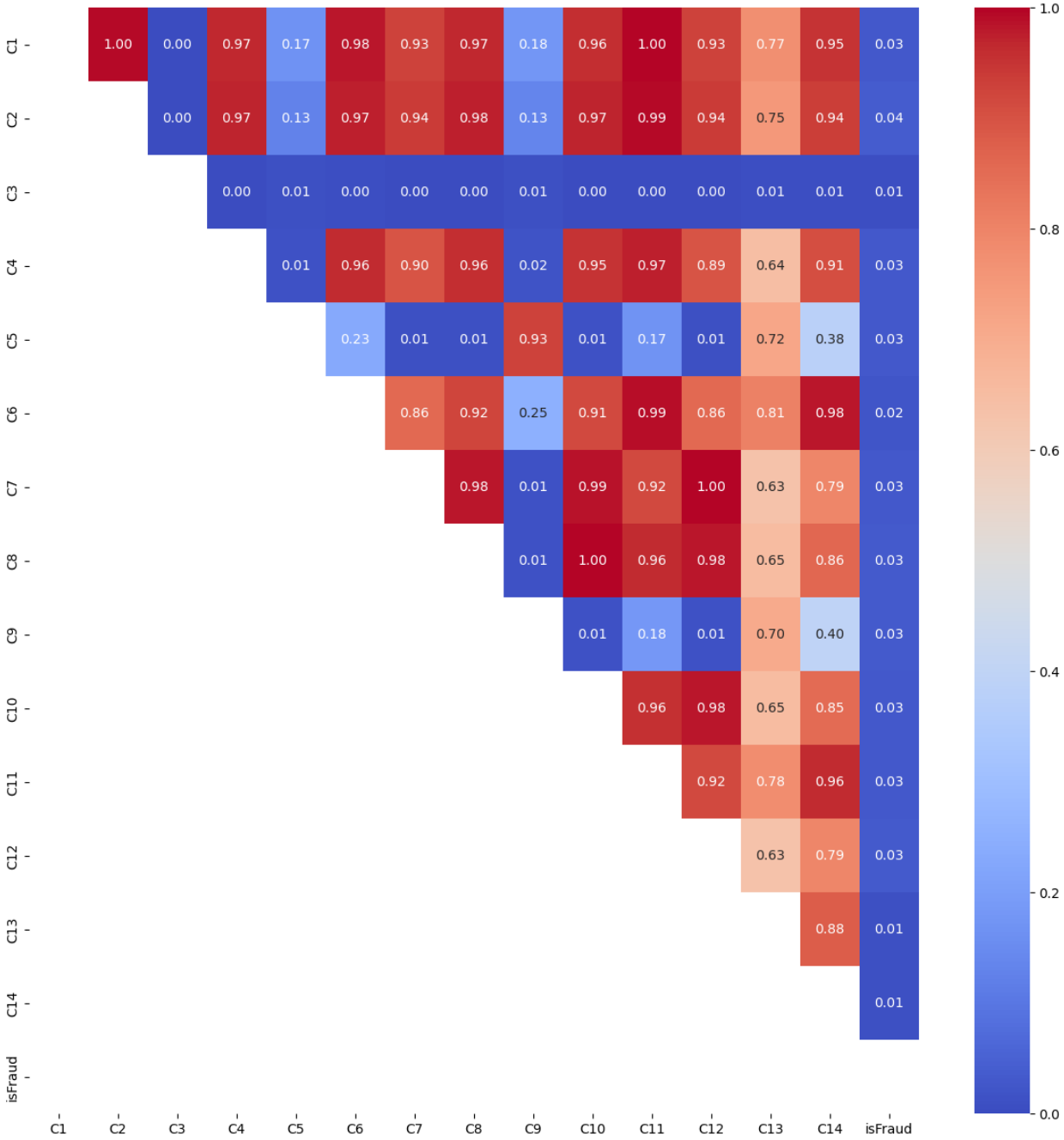


Figure 9: D-Column Correlation

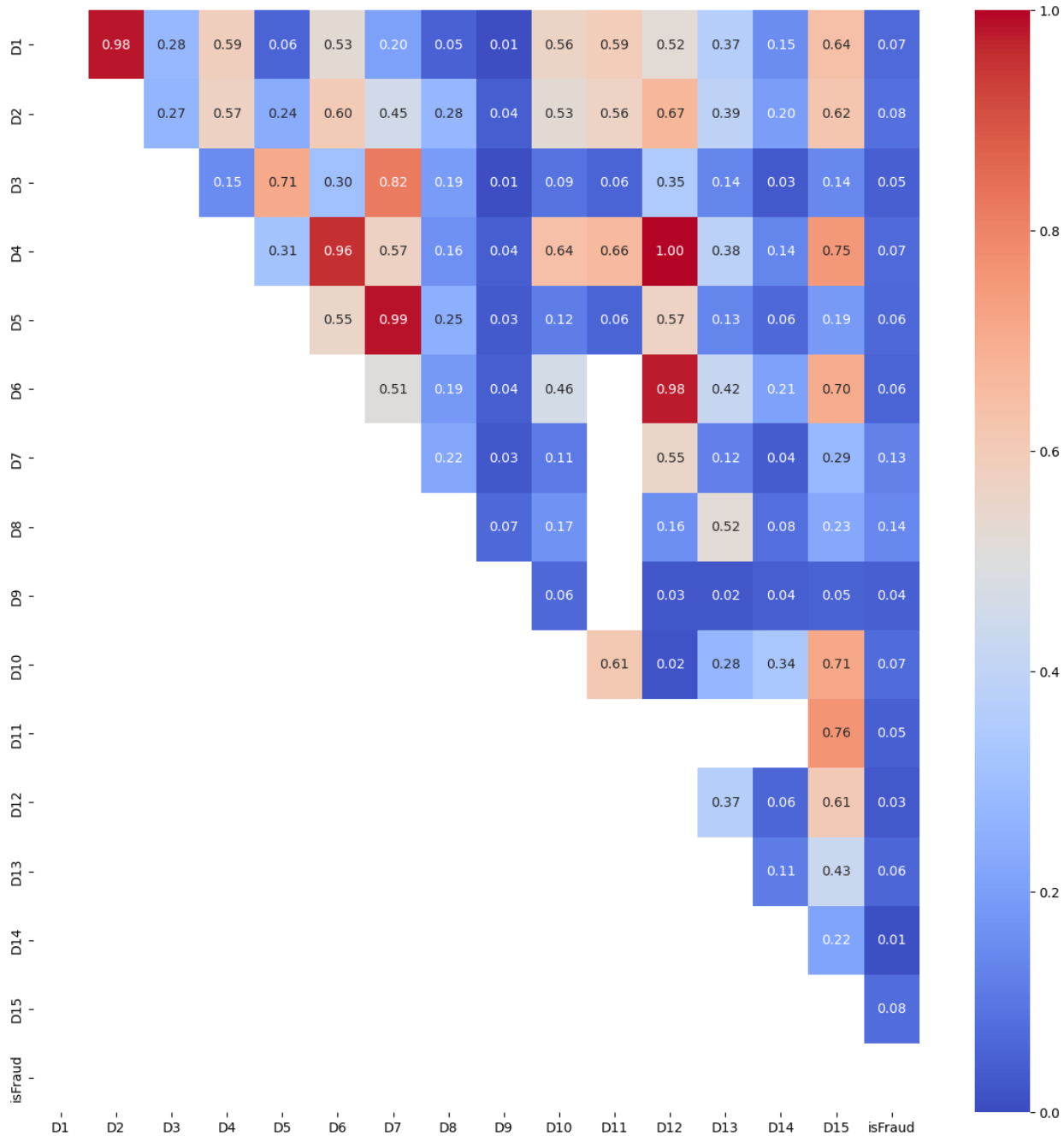


Figure 10: ID-Column Correlation

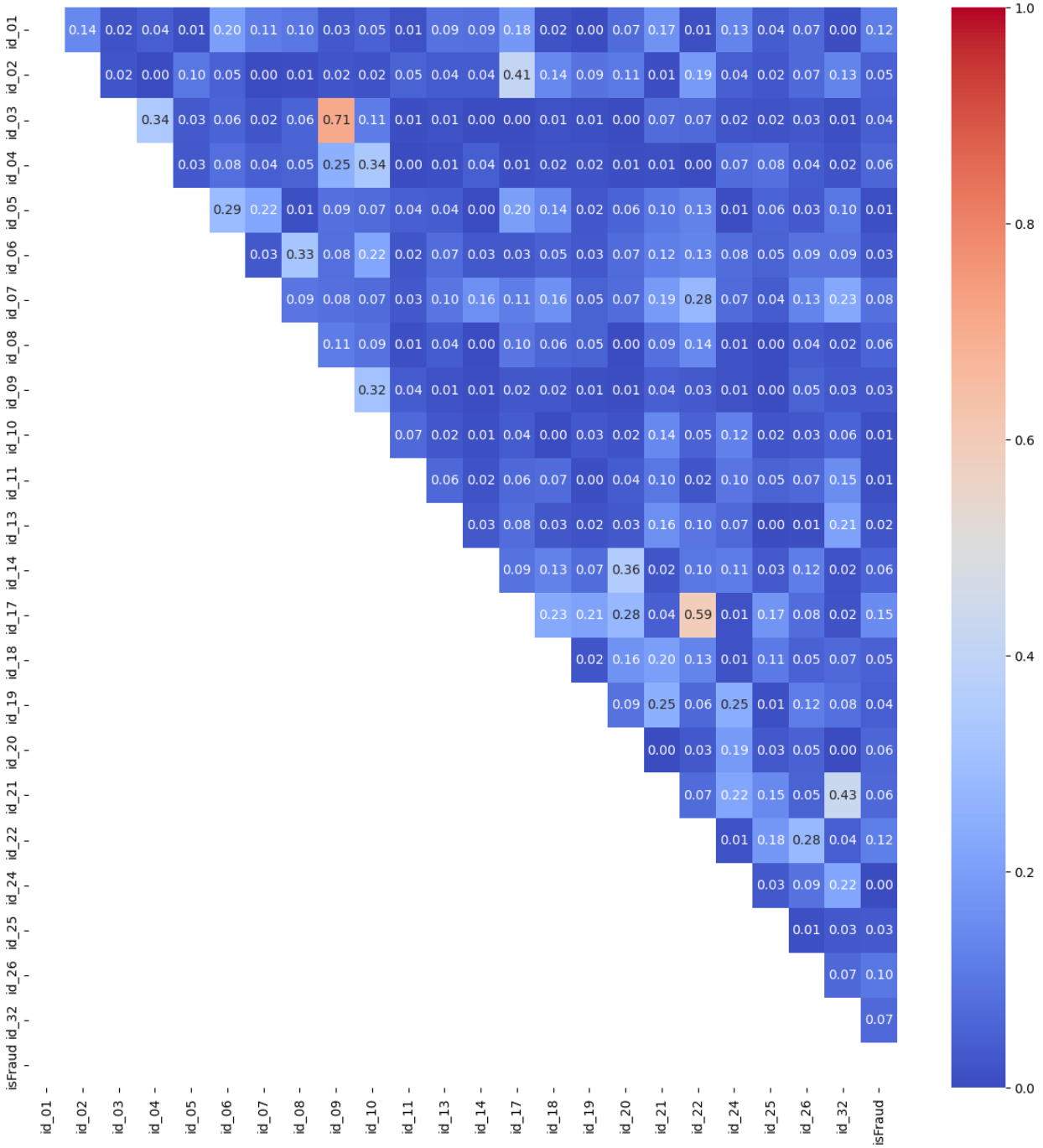


Figure 11: Explained Variance vs Number of Principal components

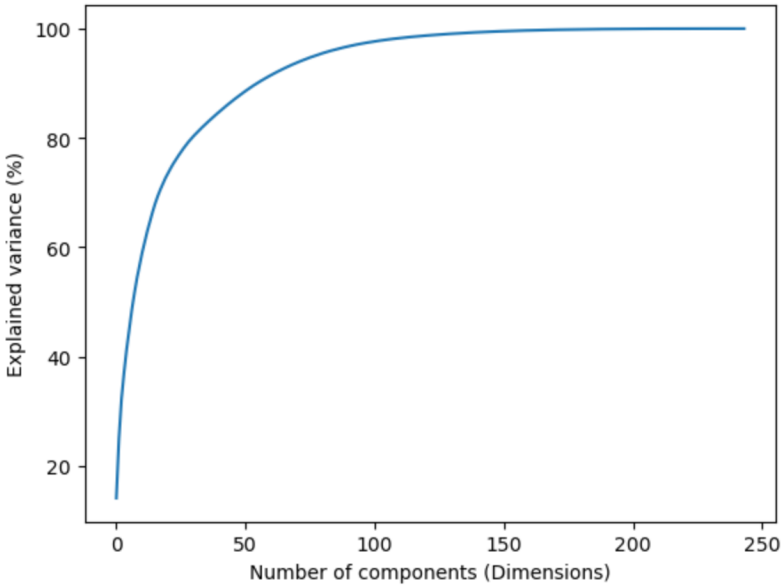
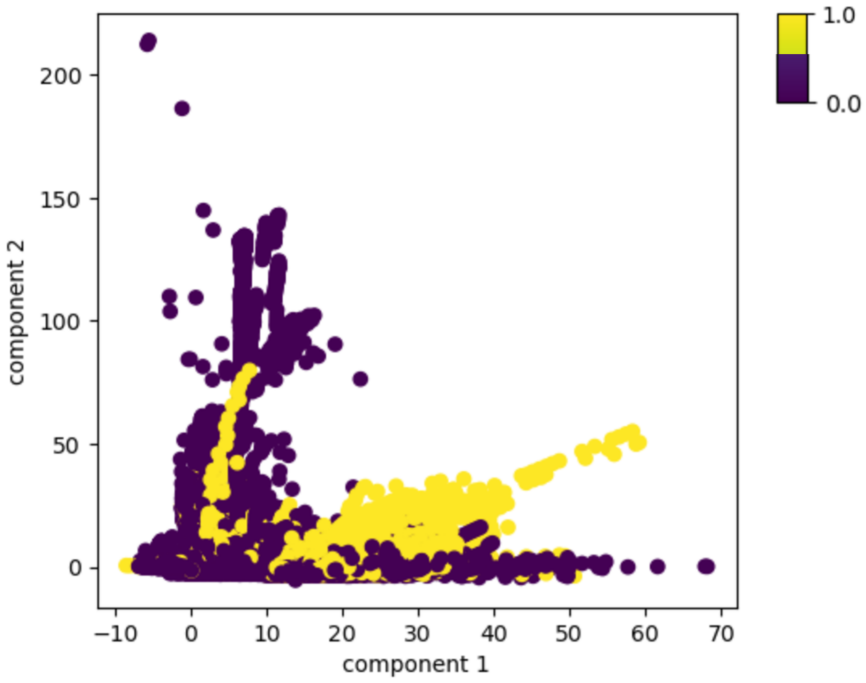


Figure 12: Visualizing all data points in first two principal components as per class (isFraud = 1 and isFraud = 0)

```
Visualizing if target variable is separated in first 2 principal components:
```



Appendix B

Pattern Mining Results

Table 1: Frequent Itemsets

Fraudulent Transactions		Non-Fraudulent Transactions	
support	itemsets	support	itemsets
0.647	['visa']	0.773	['unknownBrowser']
0.516	['debit']	0.772	['unknownDevice']
0.481	['credit']	0.772	['unknownDevice', 'unknownBrowser']
0.481	['gmail.com']	0.755	['W']
0.459	['unknownBrowser']	0.755	['unknownDevice', 'W', 'unknownBrowser']
0.457	['unknownDevice']	0.755	['W', 'unknownBrowser']
0.457	['unknownDevice', 'unknownBrowser']	0.755	['unknownDevice', 'W']
0.434	['unknownDevice', 'W', 'unknownBrowser']	0.753	['debit']
0.434	['W', 'unknownBrowser']	0.651	['visa']
0.434	['W']	0.643	['debit', 'unknownBrowser']
0.434	['unknownDevice', 'W']	0.642	['unknownDevice', 'debit']
0.387	['C']	0.642	['unknownDevice', 'debit', 'unknownBrowser']
0.371	['visa', 'debit']	0.634	['unknownDevice', 'W', 'debit', 'unknownBrowser']
0.348	['M0']	0.634	['debit', 'W', 'unknownBrowser']
0.329	['M2']	0.634	['unknownDevice', 'W', 'debit']
0.319	['M2', 'C']	0.634	['debit', 'W']
0.314	['mastercard']	0.514	['visa', 'debit']

Table 2: Association Rules (Fraudulent Transactions)

antecedents	consequents	Antecedent support	consequent support	support	confidence	lift	leverage	conviction
unknown Device	unknown Browser	0.457	0.459	0.457	0.999	2.175	0.247	851.62
W	unknown Device	0.434	0.457	0.434	1.0	2.186	0.235	inf
W	unknown Browser	0.434	0.459	0.434	1.0	2.177	0.234	inf
visa	debit	0.647	0.516	0.371	0.573	1.110	0.036	1.133
M2	C	0.329	0.387	0.319	0.968	2.499	0.191	19.577

Table 3: Association Rules (Non-Fraudulent Transactions)

antecedents	consequents	Antecedent support	consequent support	support	confidence	lift	leverage	conviction
unknown Device	unknown Browser	0.772	0.773	0.772	0.999	1.292	0.174	2267.02
W	unknown Device	0.755	0.772	0.755	1.0	1.294	0.171	inf
W	unknown Browser	0.755	0.773	0.755	1.0	1.292	0.171	inf
debit	unknown Browser	0.753	0.773	0.643	0.853	1.103	0.060	1.548
W	debit	0.755	0.753	0.634	0.839	1.114	0.065	1.537

Table 4: AUC_ROC results for logistic regression

Index	Classes Balanced	Scaling Done	Value for replacing NaN	PCA applied	AUC_ROC
i	No	No	Mean	No	0.7779
ii	Yes	No	Mean	No	0.7861
iii	Yes	Yes	Mean	No	0.8381
iv	Yes	Yes	0	No	**0.8392**
v	Yes	Yes	Mean	Yes	0.8191
vi	No	Yes	Mean	Yes	0.8151

Fig 13 (i-vi) :ROC graph for above experiments (images labeled with index number)

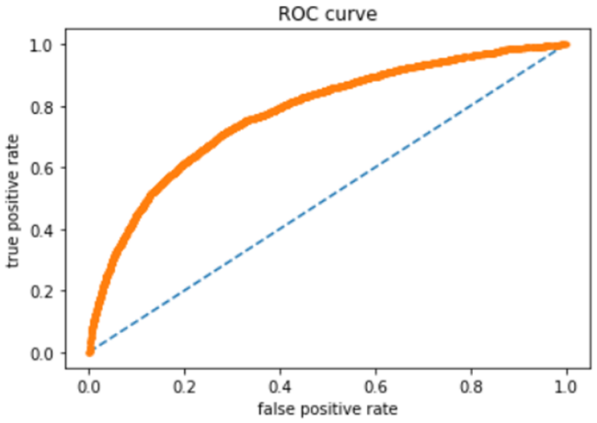


Fig i

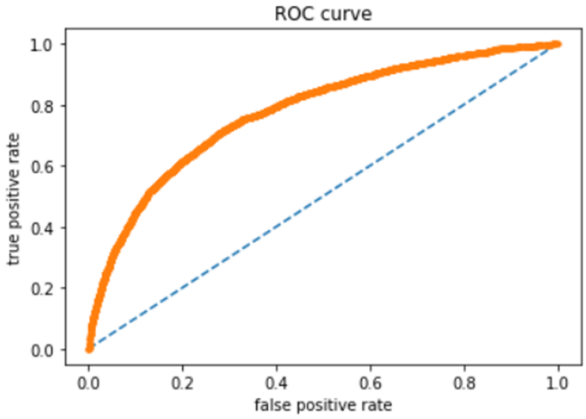


Fig ii

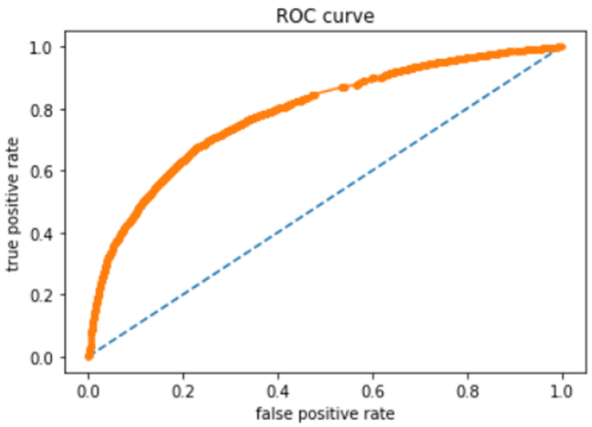


Fig iii

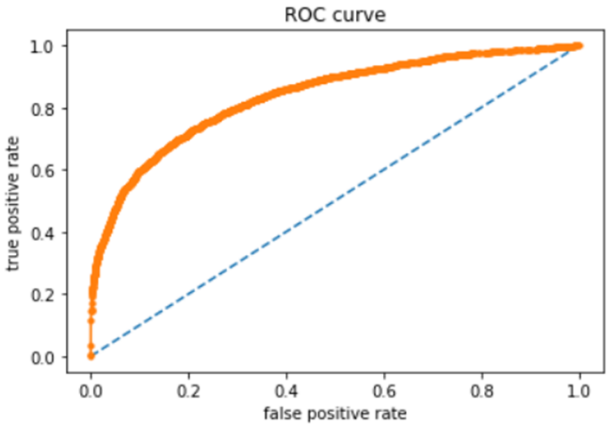


Fig iv

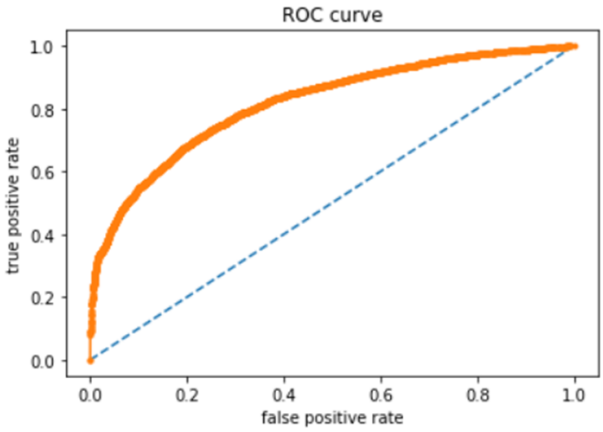


Fig v

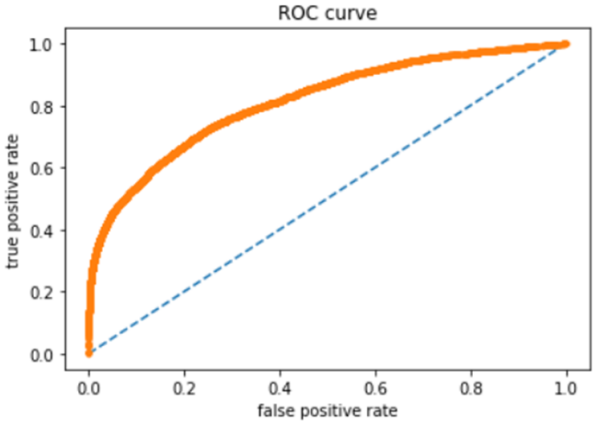


Fig vi

Table 5 : Hyperparameter tuning for random forest

Iteration #	# of Trees (T)	# of candidates available at every node for split = m	Max depth of trees = d	AUC
1	100	100	20	0.87368
2	100	17	20	0.89885
3	100	14	20	0.89364
4	100	34	20	0.88864
5	200	34	20	0.89057
6	200	17	20	0.89694
7	100	17	40	0.91926
8	100	17	60	0.92068
9	200	17	60	0.93227

Fig 14: Sample ROC curve & PR curve for Random Forest (from one iteration of k-fold CV):