5-2020

Financial Fraud Detection using Machine Learning Techniques

Matar Al Marri
mka8033@rit.edu

Ahmad AlAli
aaa4476@rit.edu

Financial Fraud Detection using Machine Learning Techniques

by

Matar Al Marri

Ahmad AlAli

A Capstone Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Science in Professional Studies: Data Analytics

Department of Graduate Programs & Research

Rochester Institute of Technology

RIT Dubai

May 2020

RIT

Master of Science in Professional Studies:

Data Analytics

Graduate Capstone Approval

Student Name**:**

Graduate Capstone Title**: Financial Fraud Detection using Machine Learning Techniques**

Graduate Capstone Committee:

Name: Dr. Sanjay Modak **Date:**

Chair of committee

Name: Dr. Ioannis Karamitsos **Date:**

Member of committee

Acknowledgments

We would like to express sincere gratitude to our committee chair Dr. Sanjay Modak and our supervisor Dr. Ioannis Karamitsos for providing their invaluable guidance, comments and suggestions throughout the course of this project. We offer our appreciation for the learning opportunities provided by the committee. We should also thank all our course instructors throughout the program without whose guidance and support it would not be possible to undertake solving of a complex analytical problem like fraud detection. We would like to specially thank Dr. Erik Golen for the very interesting Intro to Data Mining course where we studied the basic concepts related to data exploration and machine learning and learnt the fundamentals of python. This project wouldn't have been possible without getting a good understanding of the fundamental concepts that was taught in this course.

Abstract

Payments related fraud is a key aspect of cyber-crime agencies and recent research has shown that machine learning techniques can be applied successfully to detect fraudulent transactions in large amounts of payments data. Such techniques have the ability to detect fraudulent transactions that human auditors may not be able to catch and also do this on a real time basis.

In this project, we apply multiple supervised machine learning techniques to the problem of fraud detection using a publicly available simulated payment transactions data. We aim to demonstrate how supervised ML techniques can be used to classify data with high class imbalance with high accuracy.

We demonstrate that exploratory analysis can be used to separate fraudulent and non-fraudulent transactions. We also demonstrate that for a well separated dataset, tree-based algorithms like Random Forest work much better than Logistic Regression.

Table of Contents

List of Figures

List of Tables

Chapter 1

1.1 Introduction

Digital payments of various forms are rapidly increasing across the world. Payments companies are experiencing rapid growth in their transactions volume. For example, PayPal processed ~$578 billion in total payments in 2018. Along with this transformation, there is also a rapid increase in financial fraud that happens in these payment systems.

Preventing online financial fraud is a vital part of the work done by cybersecurity and cyber-crime teams. Most banks and financial institutions have dedicated teams of dozens of analysts building automated systems to analyze transactions taking place through their products and flag potentially fraudulent ones. Therefore, it is essential to explore the approach to solving the problem of detecting fraudulent entries/transactions in large amounts of data in order to be better prepared to solve cyber-crime cases.

1.2 Aims and Objectives

This project was a few month's efforts to develop a framework of fraud detection in financial transactions. We hope the outcome of the project will help streamline the analysis and detection of fraudulent transactions.

Overall, there are three main objectives of the project –

- To study the literature on financial fraud detection and understand the different aspects of the problem.
- To solve the problem of financial fraud detection on a publicly available sample dataset using supervised machine learning techniques.
- To compare different classification techniques to understand which is best suitable for this application.

Ultimately, the creation of a framework and codes that incorporate analytics and machine learning concepts studied in the program is the goal. The success of the project is predicated on the accuracy of the classification results and the extent of analysis conducted. We hope the final report will serve as a benchmark for further development

on this topic and as a knowledge base for students to understand the nuances of fraud detection.

1.3 Research Methodology

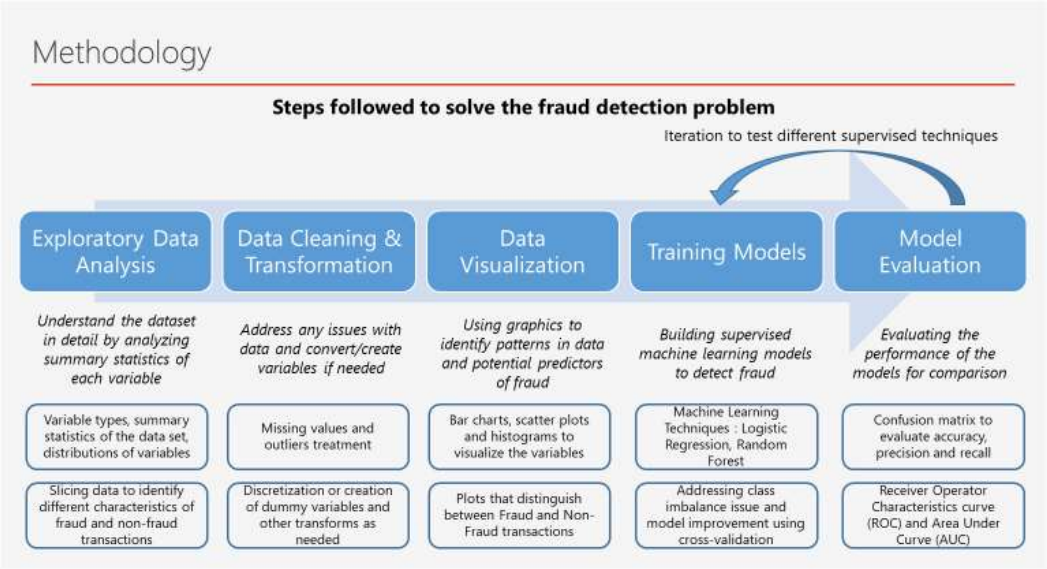
The typical machine learning approach was followed in this project. The identified dataset has labelled class variable, which was used as the prediction variable in machine learning models.

- Through exploratory analysis, we analyzed the data set in detail and identified possible predictors of fraud.
- Through various visualization techniques, we observed the separation between fraud and non-fraud transactions.
- To solve the fraud detection problem, we experimented with two supervised machine learning techniques – Logistic Regression and Random Forest.
- Additionally, we also tried under-sampling to address the class imbalance in the dataset.
- The models were developed with cross-validation to avoid overfitting and obtain consist of performance.
- Performance measures, like Confusion Matrix and Area Under Curve (AUC), was used to compare the performance of the models.

This analysis was conducted using Python through Jupyter notebook. In-built libraries and methods were used to run the machine learning models. When needed, functions were defined to simplify specific analyses or visualizations.

The below diagram shows in detail the full process that was followed in the project.

Figure 1: Project Methodology



1.4 Limitations of the Study

In this study, we evaluated the effectiveness of using specific supervised machine learning techniques to solve the problem of fraud detection in financial transactions. The limitations of the methods applied in this study are as follows:

- We used a pre-labeled dataset to train the algorithms. However, usually, it is difficult to find labeled data and thus applying supervised machine learning techniques may not be feasible. In such cases, we should evaluate unsupervised techniques which were beyond the scope of this study.
- This study considers digital transactions data that includes amount transacted, the balance of recipient and originator, and time of transaction. These variables that helped in detecting fraud may not apply to other types of financial transactions, such as credit card fraud.
- We evaluated two machine learning algorithm – Logistic Regression and Random Forest. Although the result of the study using these algorithms is good, it is necessary to evaluate other techniques to determine which algorithm works best for this application.
- Due to the large size of data, we were limited by computation capacity to explore different techniques such as grid search for parameter tuning, SMOTE sampling technique. These techniques may help in further improving the results of this study.

Chapter 2

2.1 Literature Review

Considerable literature is available on financial fraud detection due to its high importance in reducing cyber crimes and also from a business point of view. A few researchers have also conducted literature reviews of articles published in the 2000s and 2010s.

To detect financial fraud, researchers typically use outlier detection techniques (Jayakumar et.al., 2013) with highly imbalanced datasets. Different types of financial frauds are also possible. One article suggests four categories of financial fraud – financial statement fraud, transaction fraud, insurance fraud and credit fraud (Jans et al., 2011). In this project, the focus is on transaction fraud specifically as it applies to mobile payments.

A variety of techniques have been tested to detect financial fraud.

- Phua et al., (2004) used Neural Networks, Naïve Bayes and Decision Trees to detect automobile insurance fraud.
- Ravisankar et al., (2011) detect financial statement fraud in Chinese companies, another article used SVM, Genetic Programming, Logistic Regression and Neural Networks.
- Density-based clustering (Dharwa et al., 2011) and cost-sensitive Decision Trees (Sahin et al., 2013) have been used for credit card fraud.
- Sorournejad et al., (2016) discusses both supervised and unsupervised machine learning-based approaches involving ANN (Artificial Neural Networks), SVM, HMM (Hidden Markov Models), clustering.
- Wedge et al., (2018) address the problem of imbalanced data that result in a very high number of false positives, and some papers propose techniques to alleviate this problem.

However, there is very little literature available on detecting fraudulent transactions in mobile payments, probably due to relatively recent advancements in the technology.

Albashrawi et al., (2016) present a systematic review of the most used methods in financial fraud detection. The top 5 techniques are shown in the table below:

Table 1: Frequency of use of machine learning techniques in fraud detection problems

Technique	Frequency of use
Logistic Regression	13% (17 articles)
Neural Networks	11% (15 articles)
Decision Trees	11% (15 articles)
Support Vector Machines	9% (12 articles)
Naïve Bayes	6% (8 articles)

Chapter 3

3.1 Methodology

This methodology served as the deliverables of the project. It describes the results of each phase that was tried out and do a comparison between them to identify which is the best technique to address the fraud detection problem.

Each phase of the project has an output that describes the findings in that phase. These deliverables were used in this final project are explained below –

Table 2: Project Deliverables

Methodology Phases	Project Deliverables
Understanding the data set	• Report on the summary of the data set and each variable it contains along with necessary visualizations
Exploratory Data Analysis	• Report on analysis conducted and critical findings with a full description of data slices considered • Hypothesis about the separation between fraud and non-fraud transactions • Visualizations and charts that show the differences between fraud and non-fraud transactions • Python code of the analysis performed
Modeling	• Report on the results of the different techniques tried out, iterations that were experimented with, data transformations and the detailed modeling approach • Python code used to build machine learning models
Final Project Report	• Final report summarizing the work done over the course of the project, highlighting the key findings, comparing different models and identifying best model for financial fraud detection

3.2 Tools Used

This project was entirely done using Python, and the analysis was documented in a Jupyter notebook. Standard python libraries were used to conduct different analyses. These libraries are described below –

- ***sklearn*** – used for machine learning tasks
- ***seaborn*** – used to generate charts and visualizations

- ***pandas*** – used for reading and transforming the data

3.3 Data Sources

Due to the private nature of financial data, there is a lack of publicly available datasets that can be used for analysis. In this project, a synthetic dataset, publicly available on Kaggle, generated using a simulator called PaySim is used. The dataset was generated using aggregated metrics from the private dataset of a multinational mobile financial services company, and then malicious entries were injected. (TESTIMON @ NTNU, Kaggle).

The dataset contains 11 columns of information for ~6 million rows of data. The key columns available are –

- Type of transactions
- Amount transacted
- Customer ID and Recipient ID
- Old and New balance of Customer and Recipient
- Time step of the transaction
- Whether the transaction was fraudulent or not

In the following figure, a snapshot of the first few lines of the data set is presented.

Figure 2: Snapshot of the raw dataset

	# step	A type	# amount	A nameO...	# oldbala...	# newbal...	A nameD...	# oldbala...	# newbal...	# isFraud
1	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0
2	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0
3	1	TRANSFER	181.0	C1305486145	181.0	0.0	C553264065	0.0	0.0	1
4	1	CASH_OUT	181.0	C840083671	181.0	0.0	C38997010	21182.0	0.0	1
5	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0
6	1	PAYMENT	7817.71	C90045638	53860.0	46042.29	M573487274	0.0	0.0	0
7	1	PAYMENT	7107.77	C154988899	183195.0	176087.23	M408069119	0.0	0.0	0
8	1	PAYMENT	7861.64	C1912850431	176087.23	168225.59	M633326333	0.0	0.0	0
9	1	PAYMENT	4024.36	C1265012928	2671.0	0.0	M1176932104	0.0	0.0	0
10	1	DEBIT	5337.77	C712410124	41720.0	36382.23	C195600860	41898.0	40348.79	0

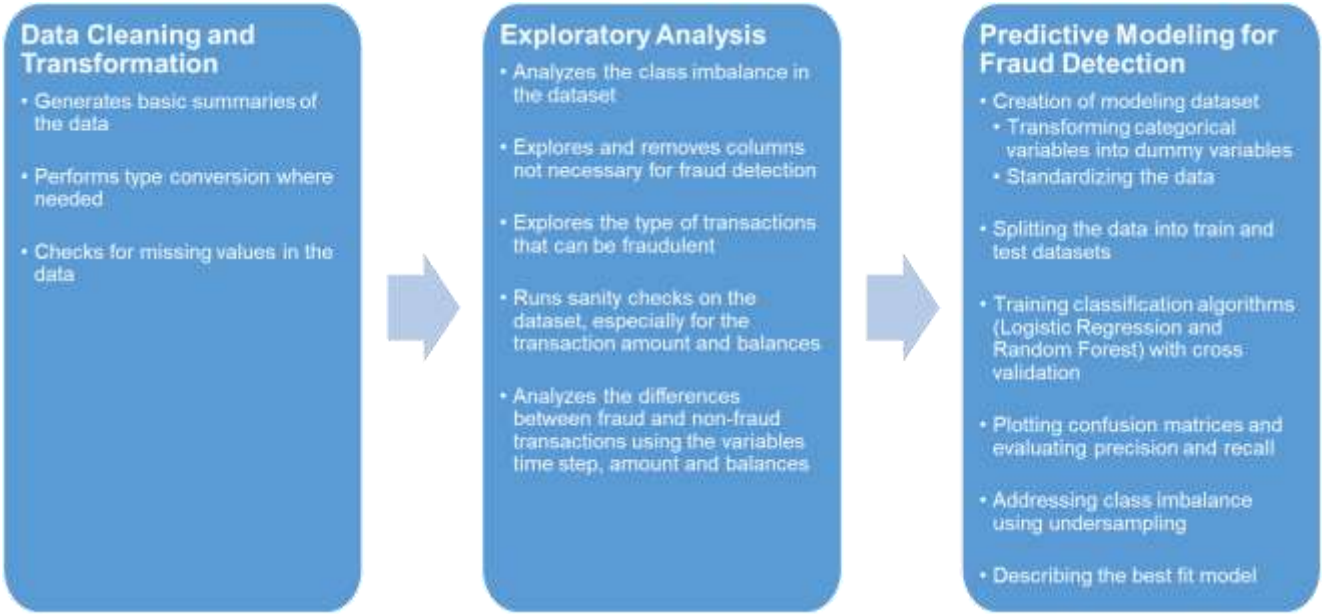
Chapter 4

4.1 Data Analysis

This section describes each step of the analysis conducted in detail. All analysis is documented in Jupyter notebook format, and the code is presented along with the outputs.

The analysis is split into three main sections. These are described in the diagram below.

Figure 3: Structure of the analysis



4.2 Detailed Analysis

The following pages show the step by step process followed in executing the mentioned analysis structure. Relevant code snippets and graphics included are based on Python programming language.

4.2.1 Data Cleaning

This section describes the data exploration conducted to understand the data and the differences between fraudulent and non-fraudulent transactions.

4.2.1.1 Data Description

The data used for this analysis is a synthetically generated digital transactions dataset using a simulator called PaySim. PaySim simulates mobile money transactions based on a sample of real transactions extracted from one month of financial logs from a mobile money service implemented in an African country. It aggregates anonymized data from the private dataset to generate a synthetic dataset and then injects fraudulent transactions.

The dataset has over 6 million transactions and 11 variables. There is a variable named 'isFraud' that indicates actual fraud status of the transaction. This is the class variable for our analysis.

The columns in the dataset are described as follows:

Table 3: Variables in the Dataset

Name of the variable	Description
step	Maps a unit of time in the real world. 1 step is 1 hour of time.
type	Indicates the type of transaction. This can be CASH-IN, CASH-OUT, DEBIT, PAYMENT or TRANSFER
amount	amount of the transaction in local currency
nameOrig	identifier of the customer who started the transaction
oldbalanceOrg	initial balance of the originator before the transaction
newbalanceOrg	originator's balance after the transaction
nameDest	identifier of the recipient who received the transaction
oldbalanceDest	initial balance of the recipient before the transaction
newbalanceDest	recipient's balance after the transaction
isFraud	indicates whether the transaction is actually fraudulent or not. The value 1 indicates fraud and 0 indicates non-fraud

4.2.1.2 Type Conversion

Since it is necessary that all columns in the data are of appropriate type for analysis, we check if there is any need for type conversion. Here are the initial types of the columns read by python.

Figure 4: Initial data types of columns

```
Out[10]:  step              int64
          type             object
          amount          float64
          nameOrig         object
          oldbalanceOrg   float64
          newbalanceOrig  float64
          nameDest         object
          oldbalanceDest  float64
          newbalanceDest  float64
          isFraud           int64
          isFlaggedFraud    int64
          dtype: object
```

The isFraud variable is read as an integer. Since this is the class variable, we convert it to object type. The following python code is used to perform this conversion.

Figure 5: [Code snippet] Type Conversion

```
# Convert class variables type to object
data['isFraud'] = data['isFraud'].astype('object')
```

4.2.1.3 Summary Statistics

Before proceeding with the analysis, we present the summary statistics of the variables. In case of numeric variables, we evaluate the mean, standard deviation and the range of values at different percentiles. In case of categorical variables, we evaluate only the number of unique categories, the most frequent category and its frequency.

Figure 6: Summary of Statistics of Numeric Variables

	step	*amount*	*oldbalanceOrg*	*newbalanceOrig*	*oldbalanceDest*	*newbalanceDest*
count	6362620	6362620	6362620	6362620	6362620	6362620
mean	243.40	179861.90	833883.10	855113.67	1100701.67	1224996.4
std	142.33	603858.23	2888242.67	2924048.50	3399180.11	3674128.9
min	1.00	0.00	0.00	0.00	0.00	0.0
25%	156.00	13389.57	0.00	0.00	0.00	0.0
50%	239.00	74871.94	14208.00	0.00	132705.66	214661.4
75%	335.00	208721.48	107315.18	144258.41	943036.71	1111909.2
max	743.00	92445516.64	59585040.37	49585040.37	356015889.35	356179278.9

Figure 7: Summary of Statistics of Categorical Variables

	type	nameOrig	nameDest	isFraud	isFlaggedFraud
count	6362620	6362620	6362620	6362620	6362620
unique	5	6353307	2722362	2	2
top	CASH_OUT	C1976208114	C1286084959	0	0
freq	2237500	3	113	6354407	6362604

4.2.1.4 Missing Values Check

In this phase, we also check if there are any missing values in the dataset. The following code and output indicate the total number of missing / NA values in all columns, which is zero.

Figure 8: [Code snippet] Missing Values Check

```
# Missing Values Check

print('Maximum number of missing values in any column: ' +
str(data.isnull().sum().max()))
```

```
Maximum number of missing values in any column: 0
```

4.2.2 Exploratory Analysis

4.2.2.1 Class Imbalance

In this exploratory analysis, we assess the class imbalance in the dataset. The class imbalance is defined as a percentage of the total number of transactions presented in the *isFraud* column.

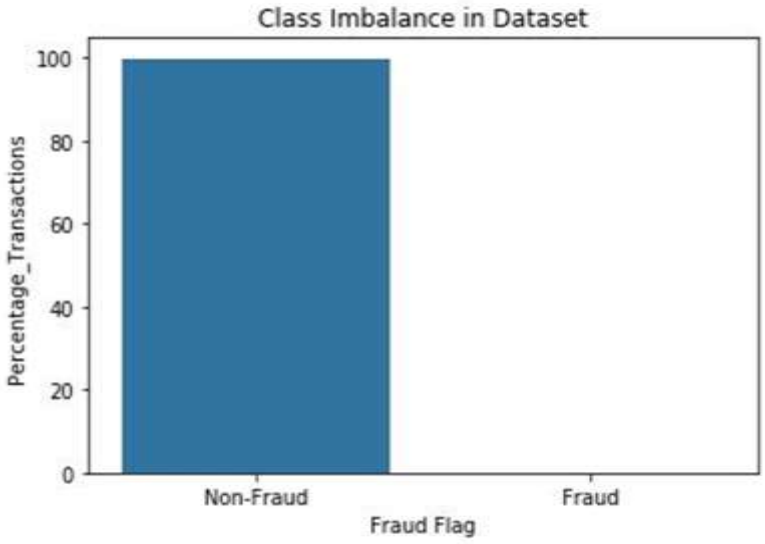
The percentage frequency output for the isFraud class variable is shown below:

Figure 9: Class Imbalance

	Fraud Flag	Percentage_Transactions
0	Non-Fraud	99.87
1	Fraud	0.13

As we can see from the figure.10 there is an enormous difference between the percentage_transactions.

Figure 10: Class Imbalance Visualization



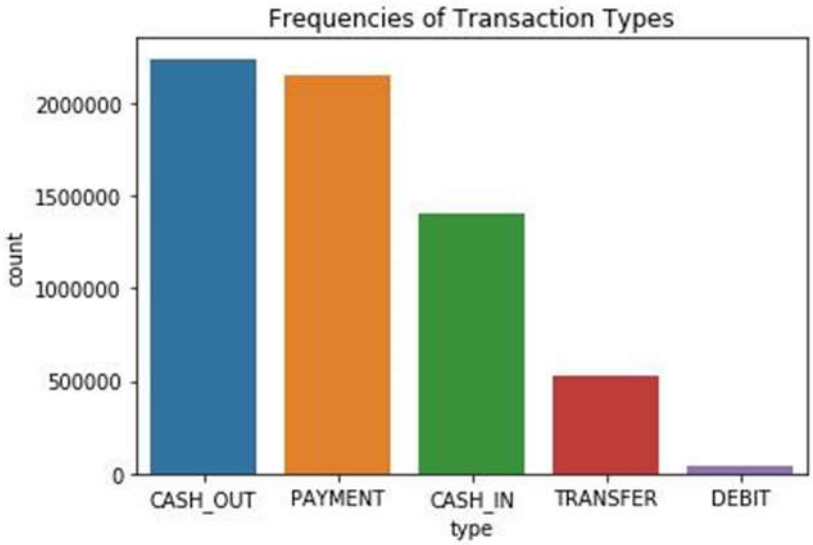
Only 0.13% (8,213) transactions in the dataset are fraudulent indicating high-class imbalance in the dataset. This is important because if we build a machine learning model on this highly skewed data, the non-fraudulent transactions will influence the training of the model almost entirely, thus affecting the results.

4.2.2.2 Types of Transactions

In this section, we are exploring the dataset by examining the 'type' variable. We present what the different 'types' of transactions are and which of these types can be fraudulent.

The following plot shows the frequencies of the different transaction types:

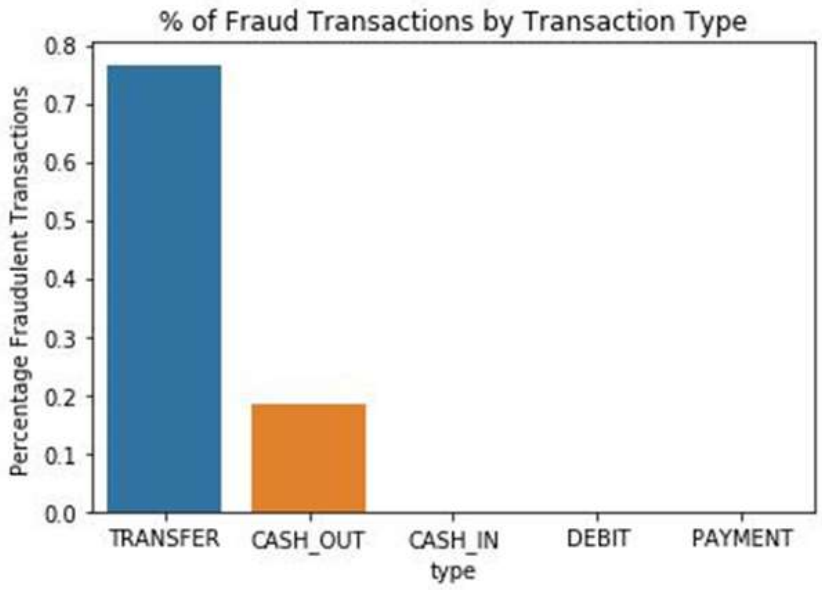
Figure 11: Frequencies of Transaction Types



The most frequent transaction types are **CASH-OUT** and **PAYMENT**.

From the above possible types of transactions, only cash-out and transfer are considered as fraudulent transactions.

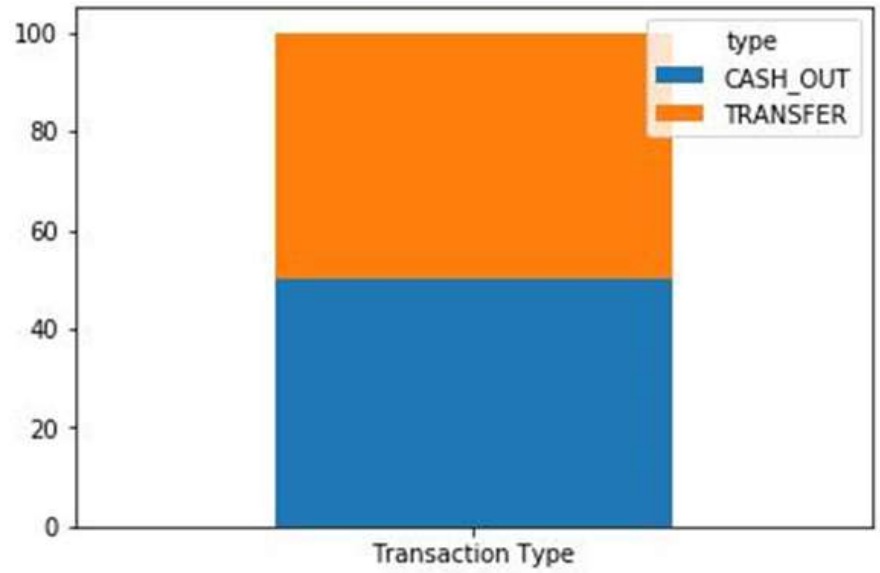
Figure 12: Fraud Transactions by Transaction Type



Only **CASH-OUT** and **TRANSFER** transactions can be fraudulent. So, it makes sense to retain only these two types of transactions in our dataset.

From figure.13 the fraudulent transactions are splitted in an equal percentage.

Figure 13: Split of Fraud Transactions by Transaction Type



Therefore, there is an almost equal likelihood that a fraudulent transaction can be **CASH_OUT** or **TRANSFER**.

Since only **CASH-OUT** and **TRANSFER** transactions can be fraudulent, we reduce the size of the dataset by retaining only these transaction types and removing PAYMENT, CASH-IN and DEBIT.

The following code performs and prints the number of new rows in the simplified data.

Figure 14: [Code snippet] Retaining only CASH-OUT and TRANSFER transactions

```
# Retaining only CASH-OUT and TRANSFER transactions

data = data.loc[data['type'].isin(['CASH_OUT', 'TRANSFER']),:]
print('The new data now has ', len(data), ' transactions.')
```

```
The new data now has 2770393 transactions.
```

Therefore, we managed to reduce the data from over 6 million transactions to ~2.8 million transactions.

4.2.2.3 Data Sanity Checks

4.2.2.3.1 Negative or Zero Transaction Amounts

First, we check if the amount column is always positive. The following two code snippets break this into the number of transactions where the amount is negative and those where the amount is 0.

Figure 15: [Code snippet] Negative or Zero Transaction Amount

```
# Check that there are no negative amounts

print('Number of transactions where the transaction amount is negative: ' +
str(sum(data['amount'] < 0)))
```

```
Number of transactions where the transaction amount is negative: 0
```

```
# Check instances where transacted amount is 0

print('Number of transactions where the transaction amount is negative: ' +
str(sum(data['amount'] == 0)))
```

```
Number of transactions where the transaction amount is negative: 16
```

There are only a few cases in which transacted amount is 0. We observe by exploring the data of these transactions that they are all fraudulent transactions. So, we can assume that if the transaction amount is 0, the transaction is fraudulent.

We remove these transactions from the data and include this condition while making the final predictions.

Figure 16: [Code snippet] Removing transactions where the amount is 0

```
# Remove 0 amount values

data = data.loc[data['amount'] > 0,:]
```

4.2.2.3.2 Originator's balance and recipient's balance

In this section, we check if there are any ambiguities in the originator's balance or recipient's balance. The following output identifies instances where originator's initial balance or recipient's final balance is 0.

Figure 17: [Code Output] Zero Balance Check

```
Percentage of transactions where originators initial balance is 0: 47.23%

Percentage of transactions where destination's final balance is 0: 0.6%
```

Therefore, in almost half of the transactions, the originator's initial balance was recorded as 0. However, in less than 1% of cases, the recipient's final balance was recorded as 0.

Ideally, the recipient's final balance should be equal to the recipient's initial balance plus the transaction amount. Similarly, the originator's final balance should be equal to originator's initial balance minus the transaction amount.

Then, we check these conditions to see whether the old balance and new balance variables are captured accurately for both originator and recipient.

Figure 18: [Code output] Incorrect Balance Check

```
% transactions where originator balances are not accurately captured: 93.72

% transactions where destination balances are not accurately captured: 42.09
```

Therefore, in most transactions, the originator's final balance is not accurately captured, and in almost half the cases, the recipient's final balance is not accurately captured.

It could be interesting to see if any of the above discrepancies identified vary between fraudulent transactions and non-fraudulent transactions. This will be done in subsequent sections.

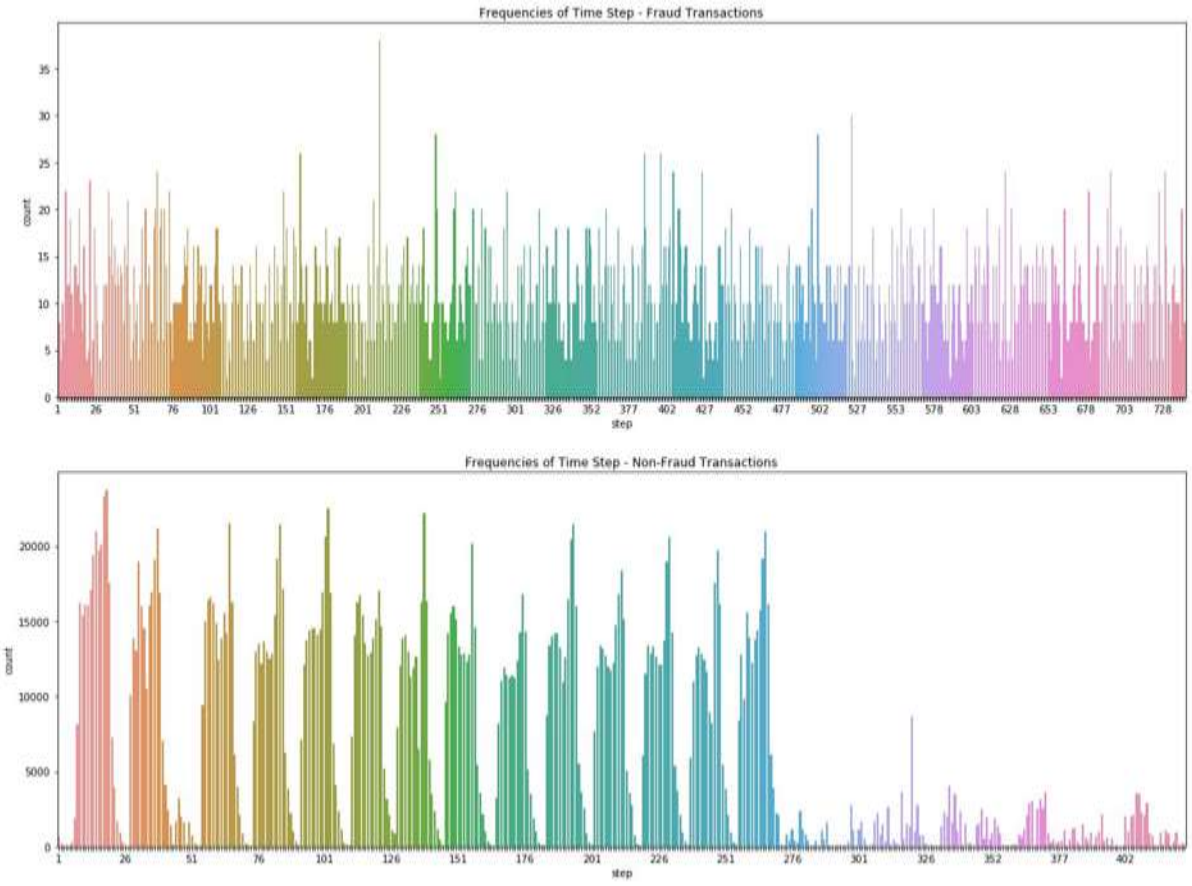
4.2.2.3.3 Fraud Transactions Analysis

In this section, an additional exploratory analysis is performed to identify if any of the variables can predict a fraud.

Time Step:

We start by analyzing the time step variable. The number of transactions in each time step by fraud status was measured in order to identify if there are any particular time steps where fraudulent transactions are more common than others. From the data description, we know that each time step is an hour.

Figure 19: Fraud and Non-Fraud Transactions Count by Time Step

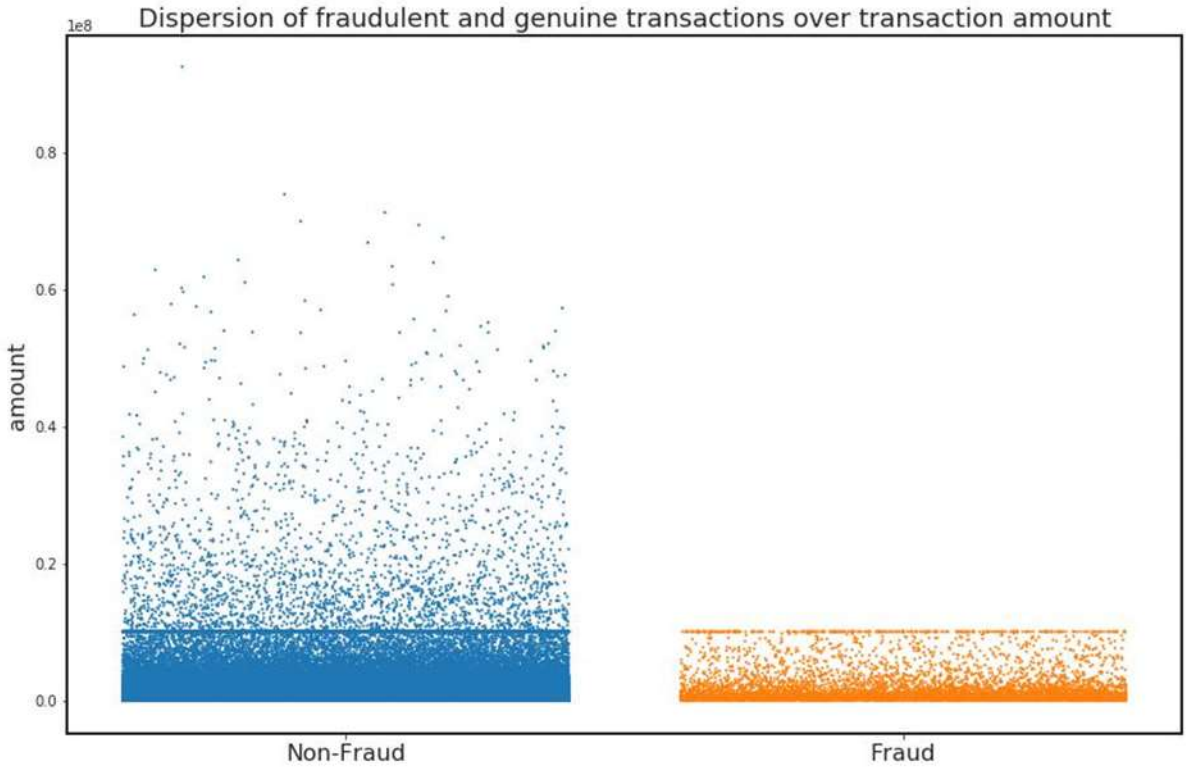


From Figure.19 show that the fraud transactions are almost uniformly spread out across time steps, whereas non-fraudulent transactions are more concentrated in specific time steps. This could be a differentiator between the two categories and can help in the training of the classification models.

Transaction Amount:

We now check if there are any differences between fraud and non-fraud transactions in terms of the transaction amount.

Figure 20: Transaction Amount of Fraud and Non-Fraud Transactions



The distribution of the transaction amount suggests that the amount can be slightly higher for Non-Fraud transactions, but nothing can be said conclusively about differences Fraud and Non-Fraud in terms of the transaction amount.

Balances:

In the previous section on Sanity Checks, we noticed that there are inaccuracies in how the 'balance' variable is captured for both originator and recipient. We also observed that in almost half the cases, the originator's initial balance is recorded as 0.

In the below code, we compare the percentage of cases where originator's initial balance is 0.

Figure 21: [Code Output] Comparison of fraud and non-fraud transactions where originator's initial balance is 0

```
% of fraudulent transactions where initial balance of originator is 0: 0.31%

% of genuine transactions where initial balance of originator is 0: 47.37%
```

In fraudulent transactions, originator's initial balance is 0 only 0.3% of the time as compared to 47% in case of non-fraudulent transactions. This could be another potential differentiator between the two categories.

We check the inaccuracy in the balance variable and compare between fraud and non-fraud. The inaccuracy is defined as the difference between what the balance should be accounting for the transaction amount and what it is recorded as balance.

We calculate the balance inaccuracies for both the originator and destination as follows:

Figure 22: [Code snippet] Defining balance inaccuracies feature

```
# Defining inaccuracies in originator and recipient balances

data['origBalance_inacc'] =   (data['oldbalanceOrg'] - data['amount']) -
data['newbalanceOrig']

data['destBalance_inacc'] = (data['oldbalanceDest'] + data['amount']) -
data['newbalanceDest']
```

In the following figures, we depicted the distribution of the balance inaccuracy feature of originator and destination balances for fraud and non-fraud transactions as below:

Figure 23: Originator Balance Inaccuracy of Fraud and Non-Fraud Transactions

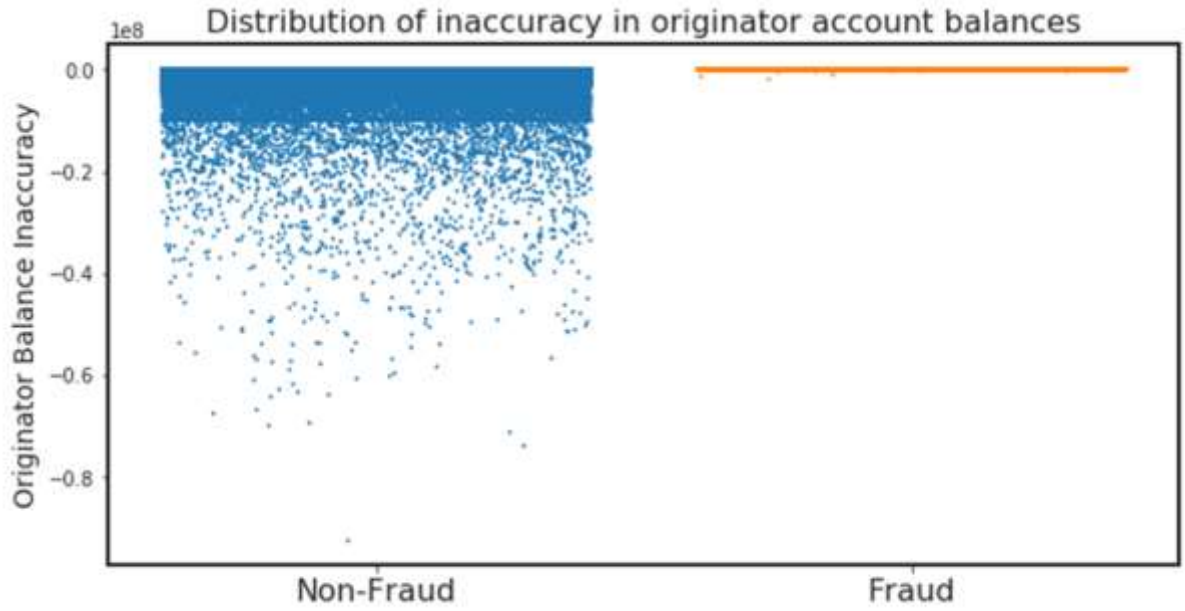
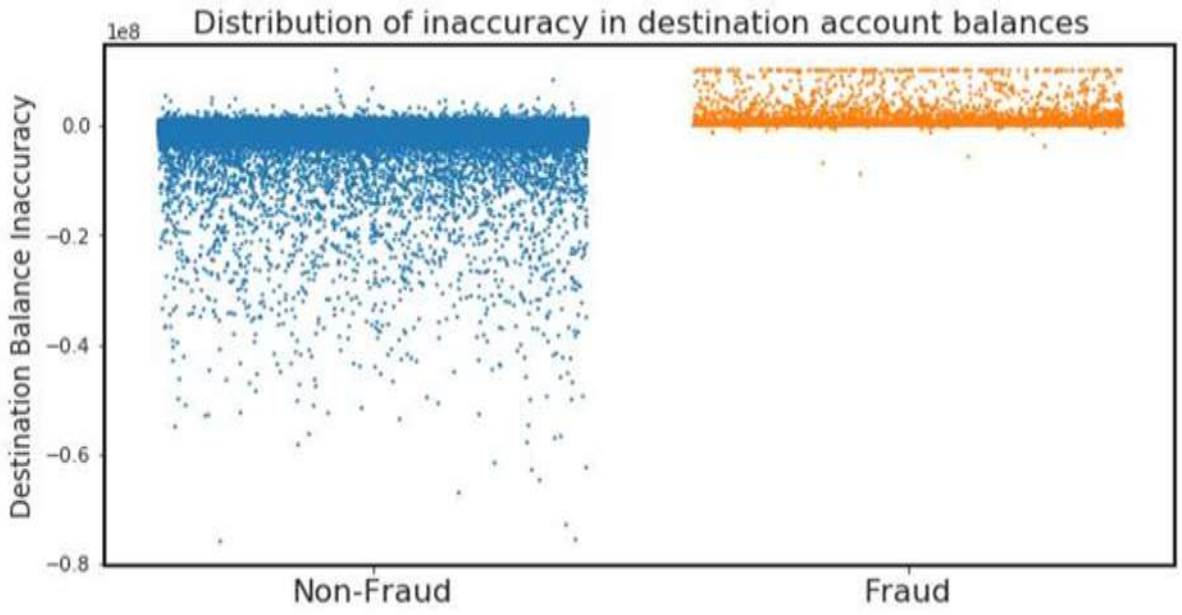


Figure 24: Destination Balance Inaccuracies of Fraud and Non-Fraud Transactions



There are differences between fraud and non-fraud in the inaccuracy measures we analyzed above. In particular, it appears that the inaccuracy in destination balance is almost always negative for non-fraud transactions, whereas it is almost always positive for fraud transactions. This could also be potential predictors of fraud.

Overall, we identified a few dimensions along which fraudulent transactions can be distinguished from non-fraudulent transactions. These are as follows:

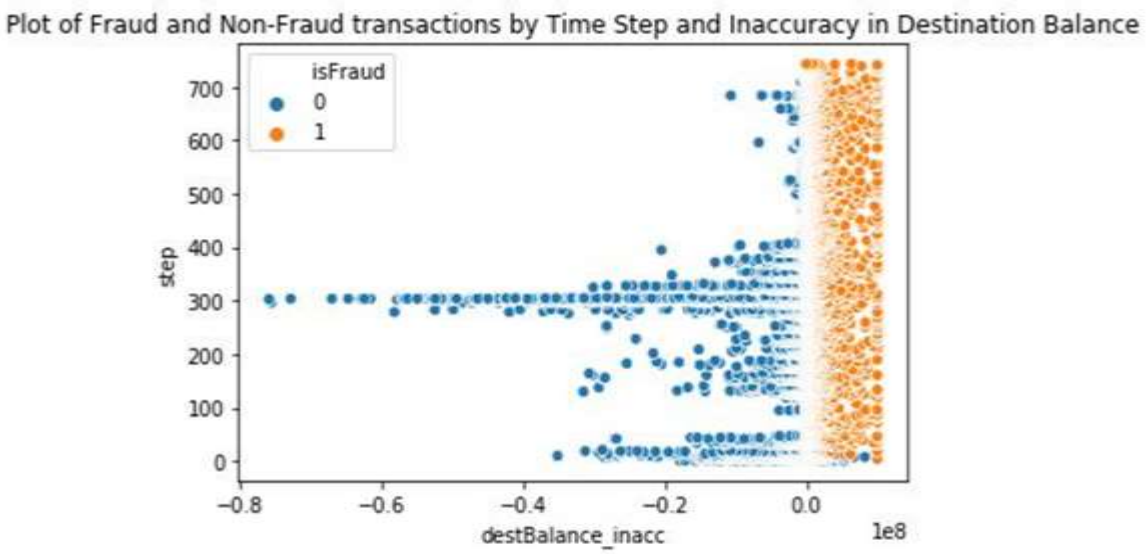
- **time step** - fraudulent transactions have are equally likely to occur in all time steps, but genuine transactions peak in specific time steps

- **balances** - initial balance of originator is much more likely to be 0 in case of genuine transactions than fraud transactions

- **inaccuracies in balance** - inaccuracy in destination balance is likely to be negative in case of genuine transactions but positive in case of fraud transactions

The below scatter plot shows a clear differentiation between fraudulent and non-fraudulent transactions along time step and destination balance inaccuracy dimensions.

Figure 25: Separation between Fraud and Non-Fraud Transactions



4.2.3 Predictive Modeling for Fraud Detection

In the previous sections, we identified dimensions that make fraudulent transactions detectable. Based on these results, we build supervised classification models.

4.2.3.1 Modeling Dataset Creation

In this section, we choose the variables needed for the ML model, encode categorical variables as numeric and standardize the data.

Let us recall columns in the dataset

```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOri g',
'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud', 'origBalance_inacc',
'destBalance_inacc'],dtype='object')
```

The name (or ID) of the originator and destination are not needed for classification. So, we remove them.

Figure 26: [Code snippet] Removing name columns

```
# Removing name columns

data = data.drop(['nameOrig', 'nameDest'], axis=1)
```

4.2.3.1.1 Creating dummy variables

We have one categorical variable in the dataset – the transaction type. This feature needs to be encoded as binary variables, and dummy variables need to be created. The following code snippet is used to perform this.

Figure 27: [Code snippet] Encoding categorical 'type' variable

```
# Creating dummy variables through one hot encoding for 'type' column
data = pd.get_dummies(data, columns=['type'], prefix=['type'])
```

This creates two binary dummy variables – **type_CASH_OUT** and **type_TRANSFER**.

4.2.3.1.2 Standardizing the data

In this transformation, we convert all columns in the data to have the same range. This is done through the standard scaler feature available in python. The following code snippet is used to perform this transformation.

Figure 28: [Code snippet] Data standardization

```
# Normalization of the dataset
std_scaler = StandardScaler()
data_scaled =
pd.DataFrame(std_scaler.fit_transform(data.loc[:,~data.columns.isin(['isFraud'])]))
data_scaled.columns = data.columns[:-1]
data_scaled['isFraud'] = data['isFraud']
```

4.2.3.1.3 Create train and test datasets

We split the scaled dataset into training and testing datasets. We decide to use 70% of the original data for training and the remaining 30% for testing.

The following code snippet is used to create training and testing datasets.

Figure 29: [Code snippet] Train and test dataset creation

```
X = data_scaled.loc[:, data_scaled.columns != 'isFraud']
y = data_scaled.loc[:, data_scaled.columns == 'isFraud']

X_train_original, X_test_original, y_train_original, y_test_original =
train_test_split(X,y,test_size = 0.3, random_state = 0)

label_encoder = LabelEncoder()
y_train_original = label_encoder.fit_transform(y_train_original.values.ravel())
y_test_original = label_encoder.fit_transform(y_test_original.values.ravel())
```

Then we check whether the class imbalance in train and test datasets are similar. The following code output indicates the % of transactions that are fraud in the two datasets –

Figure 30: [Code output] Class imbalance in train and test datasets

```
Class imbalance in train dataset: 0.297%

Class imbalance in test dataset 0.291%
```

Therefore, the class imbalance is similar, and we can proceed with the training of the algorithms.

4.2.3.2 Classification Models for Fraud detection

We define two models to perform the classification: **Logistic Regression** and **Random Forest**.

To measure the performance of the models, Recall is a useful metric. High-class imbalance datasets typically result in poor Recall, although accuracy may be high. Precision will also be a consideration because reduced precision implies that the company that is trying to detect fraud will incur more cost in screening the transactions. In fraud detection problems, though, accurately identifying fraudulent transactions is more critical than incorrectly classifying legitimate transactions as fraudulent.

Alternatively, we could also go with Area Under Curve (AUC) of the ROC curve. However, this will not adequately capture if the model is correctly identifying most of the fraudulent transactions. Therefore, we use this as a validation of the model performance.

The following code snippet is used to define the accuracy of the two models.

Figure 31: [Code snippet] Defining Logistic Regression and Random Forest Models

```
scr = 'recall'
accuracy_dict = {}
model_lr = LogisticRegression()
model_rf = RandomForestClassifier()
```

We also need to do cross-validation to ensure the models do not overfit the training data. For this, we use Stratified 5-fold since we need to ensure that the class imbalance is retained in the validation sets.

Figure 32: [Code snippet] Defining stratified 5-fold cross-validation

```
skf = StratifiedKFold(5)
```

4.2.3.2.1 Logistic Regression Model

In this section, we train the logistic regression model and calculate the mean recall score. This parameter will serve as a benchmark for further experiments.

Figure 33: [Code snippet] Logistic Regression model training

```
sc_lr = cross_val_score(model_lr, X_train_original, y_train_original, cv=skf,
scoring=scr)
```

The following output indicates how the Logistic Regression model performs on the training dataset.

Figure 34: [Code output] Logistic Regression model training performance

```
Logistic Regression's average recall score across validation sets is: 50.67%
```

Therefore, the default Logistic Regression model is able to capture only half of the actual Fraud cases.
We plot the confusion matrixes for the train and test datasets of the logistic regression model, and we check the precision and recall in each case.

Figure 35: Logistic Regression - Train Confusion Matrix

```
Precision: 91.03%
Recall: 50.88%
```

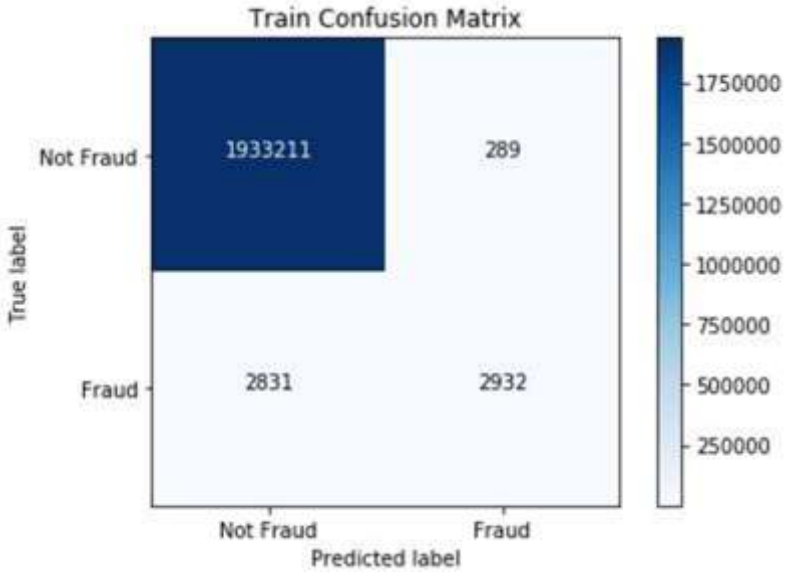
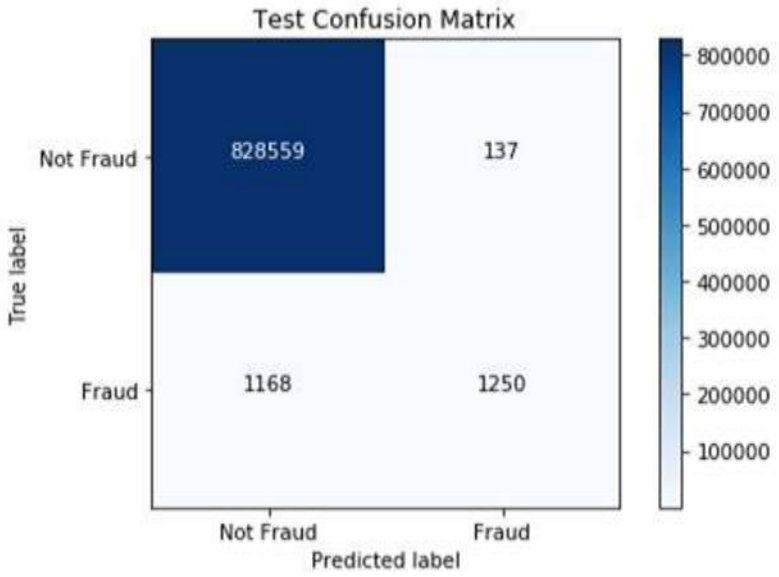


Figure 36: Logistic Regression - Test Confusion Matrix

```
Precision: 90.12%
Recall: 51.7%
```



From the above results, there are two results:

- The training and testing datasets are consistent, and there is no overfitting.
- High precision and low Recall indicate that running the algorithm on the data with high-class imbalance will not provide excellent results.

4.2.3.2.2 Random Forest Model

In this section, we repeat the same steps using a different classification algorithm such as Random Forest, and we calculate the mean recall score. We can compare with the Logistic Regression model to evaluate which is to perform better.

Figure 37: [Code snippet] Random Forest model training

```
sc_rf = cross_val_score(model_rf, X_train_original, y_train_original, cv=skf,
scoring=scr)
```

The following output indicates how the Random Forest model performs on the training dataset.

Figure 38: [Code output] Random Forest model training performance

```
Random Forest's average recall score across validation sets is: 99.48%
```

The Random Forest model seems to produce excellent results on the training dataset. Again, we plot the confusion matrices for the training and testing datasets and we check the precision and recall in each case.

Figure 39: Random Forest - Train Confusion Matrix

```
Precision: 100.0%
Recall: 99.84%
```

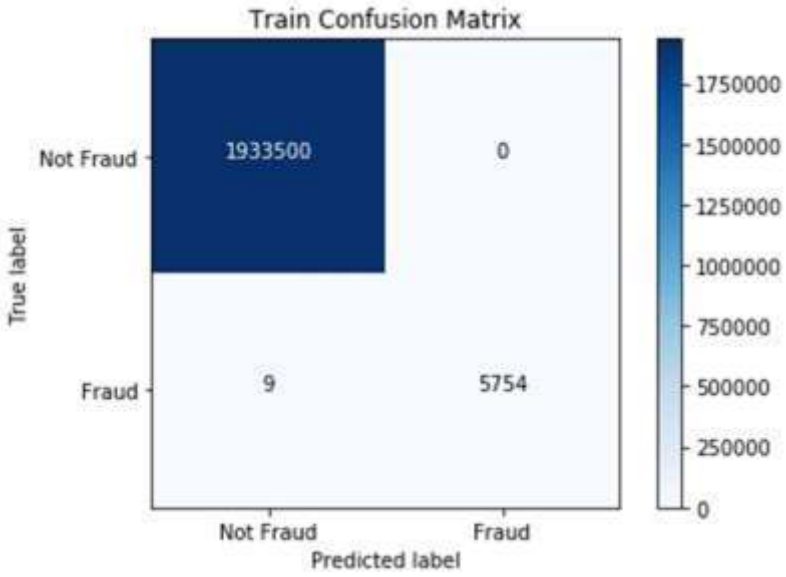
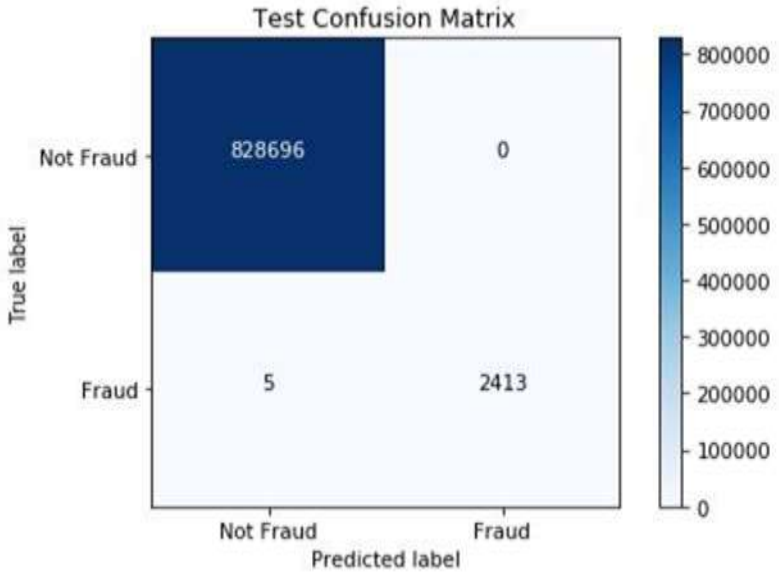


Figure 40: Random Forest - Test Confusion Matrix

```
Precision: 100.0%
Recall: 99.79%
```



The Random Forest algorithm gives almost perfect results. Comparing the recall scores with Logistic Regression, Random Forest performs much better in detecting fraud.

Also, the performance of the Random Forest model is consistent between the training and testing datasets. So, there is no overfitting.

The following table compares the results of the two models:

Table 4: Comparison of Results of Logistic Regression and Random Forest

Model	Train Precision	Train Recall	Test Precision	Test Recall
Logistic Regression	91.03%	50.88%	90.12%	51.7%
Random Forest	100%	99.84%	100%	99.79%

Regardless of the positive results from the Random Forest model, we should try to improve the results of Logistic Regression through parameter tuning and by addressing the class imbalance. In the following section, we present these techniques.

4.2.3.2.3 Addressing Class Imbalance

There are many techniques to address high-class imbalanced datasets. A few examples are as follows –

- **Undersampling:** In this method, random samples from the majority class are deleted so that the class imbalance is more manageable.
- **Oversampling:** In this method, observations of the minority class are resampled with repetition to increase their presence in the data
- **SMOTE:** This is a type of oversampling, but instead of repeating the observations, it synthesizes new plausible observations of the minority class

We use undersampling as it is less computation-intensive. We also do this only for the logistic regression model as the random forest model is already giving excellent results. The aim is to check if it is possible to get better performance than what we observed with the Random Forest model.

We train the Logistic Regression model on a subset of the original training dataset. We retain all the fraud cases and randomly select an equal number of non-fraud cases to create an undersampled training dataset.

The following code snippet is used to do this –

Figure 41: [Code snippet] undersampling the training dataset

```
# Undersampling the training dataset

fraud_indices_train = np.where(y_train_original == 1)[0]
non_fraud_indices_train = np.where(y_train_original == 0)[0]

undersample_non_fraud_indices_train =
np.random.choice(non_fraud_indices_train,len(fraud_indices_train), replace = False)
undersample_non_fraud_indices_train =
np.array(undersample_non_fraud_indices_train)

undersample_indices_train =
np.concatenate([fraud_indices_train, undersample_no n_fraud_indices_train])

X_train_undersample =
X_train_original.loc[X_train_original.reset_index(drop=True).index.isin(undersample_i
ndices_train),:]
y_train_undersample = y_train_original[undersample_indices_train.tolist()]
```

Following code, the output indicates the number of transactions in the undersampled data –

Figure 42: [Code output] Rows in the undersampled training data

```
There are 11526 rows in the undersampled training data.
```

Logistic Regression Parameter Tuning:

We now identify the best Logistic Regression model for the undersampled dataset by tuning the 'Cost function' and 'Regularization factor' parameters. The following output describes the recall scores for different combinations of the penalty and cost function.

Figure 43: [Code output] Logistic Regression Parameter Tuning - Undersampling

```
Recall of Logistic Regression for l1 penalty and C = 0.001 is: 0.0%
Recall of Logistic Regression for l1 penalty and C = 0.01 is: 22.22%
Recall of Logistic Regression for l1 penalty and C = 0.1 is: 41.02%
Recall of Logistic Regression for l1 penalty and C = 1 is: 43.83%
Recall of Logistic Regression for l1 penalty and C = 10 is: 44.15%
Recall of Logistic Regression for l1 penalty and C = 100 is: 44.16%
Recall of Logistic Regression for l1 penalty and C = 1000 is: 44.16%
Recall of Logistic Regression for l2 penalty and C = 0.001 is: 43.21%
Recall of Logistic Regression for l2 penalty and C = 0.01 is: 44.13%
Recall of Logistic Regression for l2 penalty and C = 0.1 is: 44.16%
Recall of Logistic Regression for l2 penalty and C = 1 is: 44.16%
Recall of Logistic Regression for l2 penalty and C = 10 is: 44.16%
Recall of Logistic Regression for l2 penalty and C = 100 is: 44.16%
Recall of Logistic Regression for l2 penalty and C = 1000 is: 44.16%
```

Therefore, the best Logistic Regression model with undersampling (l1 penalty and C of 100) has a recall of <50%.

The default random forest model performs better than logistic regression model.

4.2.3.2.4 Best Fit Model Details

The Random Forest model gave the best results above. The parameters of this model are presented in the following code.

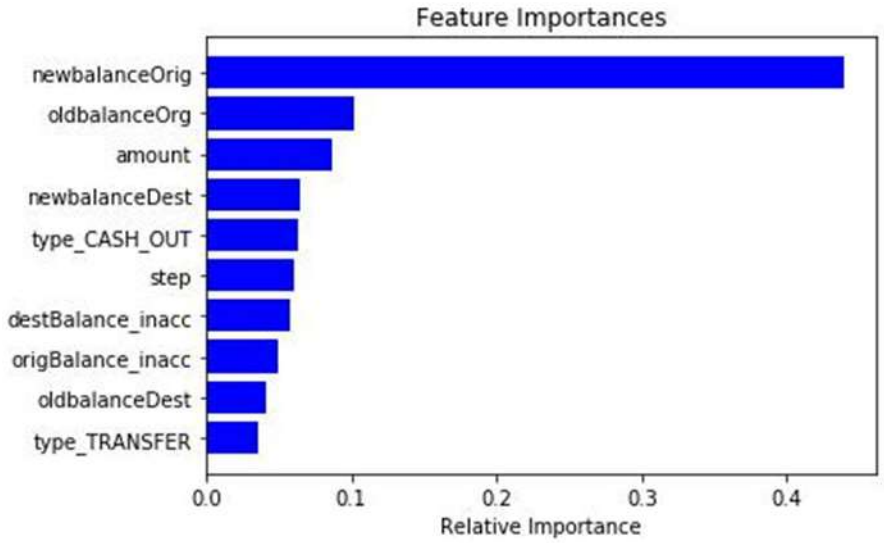
Figure 44: [Code output] Parameters of the best fit Random Forest Model

```
<bound method BaseEstimator.get_params of RandomForestClassifier(bootstrap=Tr
ue, class_weight=None, criterion='gini', max_depth=None, max_features='auto',
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2, min_weight_fraction_leaf=0.0,
n_estimators=10, n_jobs=None, oob_score=False, random_state=None, verbose=0,
warm_start=False)>
```

The model uses 10 trees in the forest (n_estimators) and has an infinite max depth. Positive cross-validation results remove the possibility of overfitting.

In the following figure, we present the relative feature importance of the random forest model. The following plot shows which variables are contributing more to make the fraud prediction.

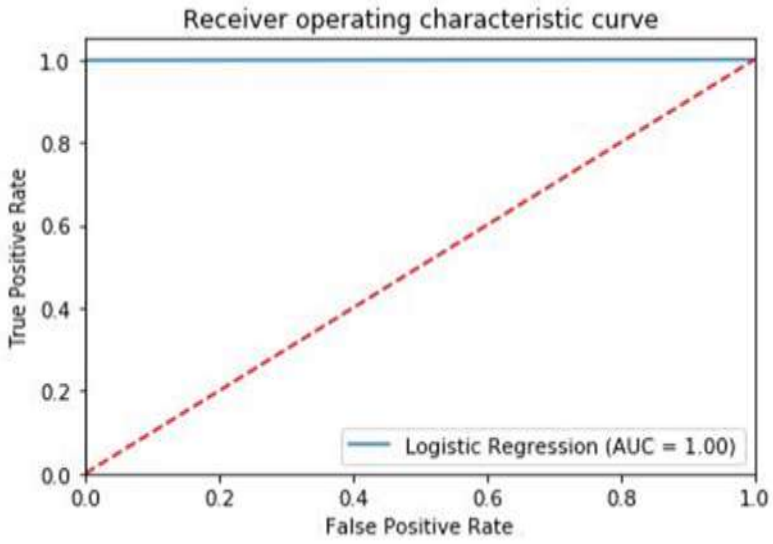
Figure 45: Random Forest Model Feature Importance



Therefore, the balance of the originator (“newbalanceOrig”) feature is critical to making the prediction as compared to all other variables.

For the Receiver-Operator Characteristics (ROC) curve and calculate Area-Under-Curve (AUC) for this model is depicted in the following figure.

Figure 46: ROC curve of Random Forest Model



4.2.4 Analysis Summary

We analyzed the financial transactions data and developed a machine learning model to detect fraud. The analysis included data cleaning, exploratory analysis and predictive modeling.

In the data cleaning, we checked for missing values, converted data types and summarized the variables in the data. In an exploratory analysis, we looked at the class imbalance, and deep-dived into each of the variables, in particular transaction type, transaction amount, balance and time step. We identified derived variables that can help with fraud detection. We also plotted various graphs to better visualize the data and come up with insights.

In predictive modeling, we experimented with Logistic Regression and Random Forest algorithms. We observed that Random Forest performs best for this application with almost 100% precision and recall scores. We tried to improve the logistic regression results by undersampling, but the results were the same because of a lot of the data is excluded. We ensured that there is no overfitting in the models through cross-validation.

We can conclude that fraud detection in financial transactions is successful in this labeled dataset, and the best algorithm for this purpose is Random Forest.

Chapter 5

5.1 Conclusion

In conclusion, we successfully developed a framework for detecting fraudulent transactions in financial data. This framework will help understand the nuances of fraud detection such as the creation of derived variables that may help separate the classes, addressing class imbalance and choosing the right machine learning algorithm.

We experimented with two machine learning algorithms – Logistic Regression and Random Forest. The Random Forest algorithm gave far better results than Logistic Regression indicating tree-based algorithms work well for transactions data with well-differentiated classes. This also emphasizes the usefulness of conducting rigorous exploratory analysis to understand the data in detail before developing machine learning models. Through this exploratory analysis, we derived a few features that differentiated the classes better than the raw data.

5.2 Recommendations

Through this project, we demonstrated that it is possible to identify fraudulent transactions in financial transactions data with very high accuracy despite the high-class imbalance. We provide the following recommendations from this exercise -

- Fraud detection in transactions data where transaction amount and balances of the recipient and originator are available can be best performed using tree-based algorithms like Random Forest
- Using dispersion and scatter plots to visualize the separation between fraud and non-fraud transactions is essential to choose the right features
- To address the high-class imbalance typical in fraud detection problems, sampling techniques like undersampling, oversampling, SMOTE can be used. However, there are limitations in terms of computation requirements with these approaches, especially when dealing with big data sets.
- To measure the performance of fraud detection systems, we need to be careful about choosing the right measure. The recall parameter is a good measure as it captures whether a good number of fraudulent transactions are correctly classified or not. We should not rely only on accuracy as it can be misleading.

References

1. E. Ngai et.al., *The Application of Data Mining Techniques in Financial Fraud Detection: A Classification Framework and an Academic Review of Literature*, Decision Support Systems. 50, 2011, 559–569
2. Albashrawi et.al., *Detecting Financial Fraud Using Data Mining Techniques: A Decade Review from 2004 to 2015*, Journal of Data Science 14(2016), 553-570
3. TESTIMON @ NTNU, *Synthetic Financial Datasets for Fraud Detection*, Kaggle, retrieved from https://www.kaggle.com/ntnu-testimon/paysim1
4. Jayakumar et.al., *A New Procedure of Clustering based on Multivariate Outlier Detection*. Journal of Data Science 2013; 11: 69-84
5. Jans et.al, *A Business Process Mining Application for Internal Transaction Fraud Mitigation*, Expert Systems with Applications 2011; 38: 13351–13359
6. Phua et.al., *Minority Report in Fraud Detection: Classification of Skewed Data*. ACM SIGKDD Explorations Newsletter 2004; 6: 50-59.
7. Dharwa et.al., *A Data Mining with Hybrid Approach Based Transaction Risk Score Generation Model (TRSGM) for Fraud Detection of Online Financial Transaction*, International Journal of Computer Applications 2011; 16: 18-25.
8. Sahin et.al., *A Cost-Sensitive Decision Tree Approach for Fraud Detection*, Expert Systems with Applications 2013; 40: 5916–5923.
9. Sorournejad et.al., *A Survey of Credit Card Fraud Detection Techniques: Data and Technique Oriented Perspective,* 2016
10. Wedge et.al., *Solving the False Positives Problem in Fraud Prediction Using Automated Feature Engineering*, Machine Learning and Knowledge Discovery in Databases, pp 372-388, 2018