Table of Contents

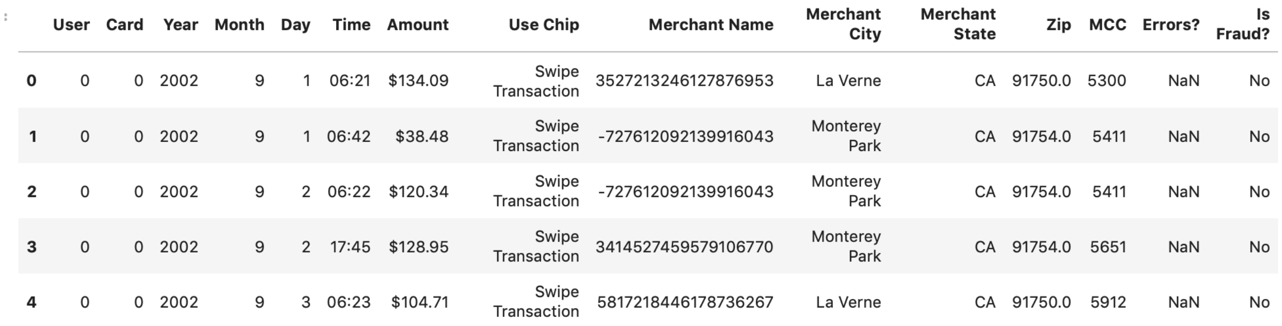
1 Fraud detection model

In the following sections we'll show how to tackle the problem of building a machine learning model for detecting fraudulent transactions, given a dataset with fraudulent and legitimate transactions. This dataset contains over 20 million synthetically-generated transactions and can be downloaded from Kaggle.

1.1 Data exploration

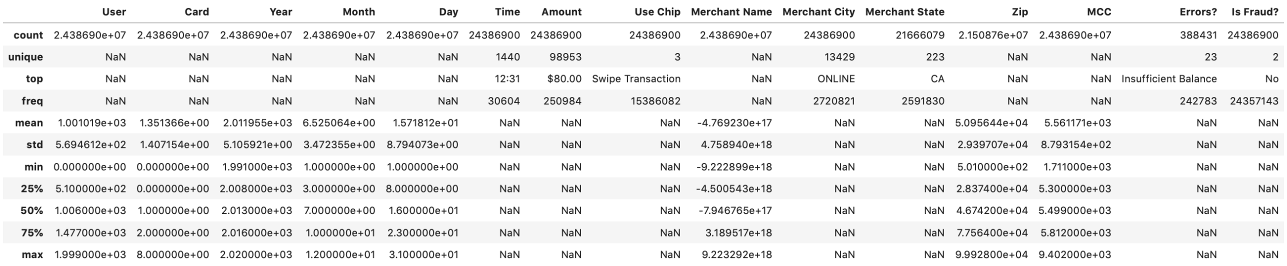
We'll start by exploring the different variables of the dataset in order to get familiar with the different types, ranges and potentially missing values.

The shape of the dataset is 24,386,900 rows by 15 columns. Below is a small sample from it:



	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?
0	0	0	2002	9	1	06:21	$134.09	Swipe Transaction	3527213246127876953	La Verne	CA	91750.0	5300	NaN	No
1	0	0	2002	9	1	06:42	$38.48	Swipe Transaction	-727612092139916043	Monterey Park	CA	91754.0	5411	NaN	No
2	0	0	2002	9	2	06:22	$120.34	Swipe Transaction	-727612092139916043	Monterey Park	CA	91754.0	5411	NaN	No
3	0	0	2002	9	2	17:45	$128.95	Swipe Transaction	3414527459579106770	Monterey Park	CA	91754.0	5651	NaN	No
4	0	0	2002	9	3	06:23	$104.71	Swipe Transaction	5817218446178736267	La Verne	CA	91750.0	5912	NaN	No

We can also have a statistical summary of the dataset:



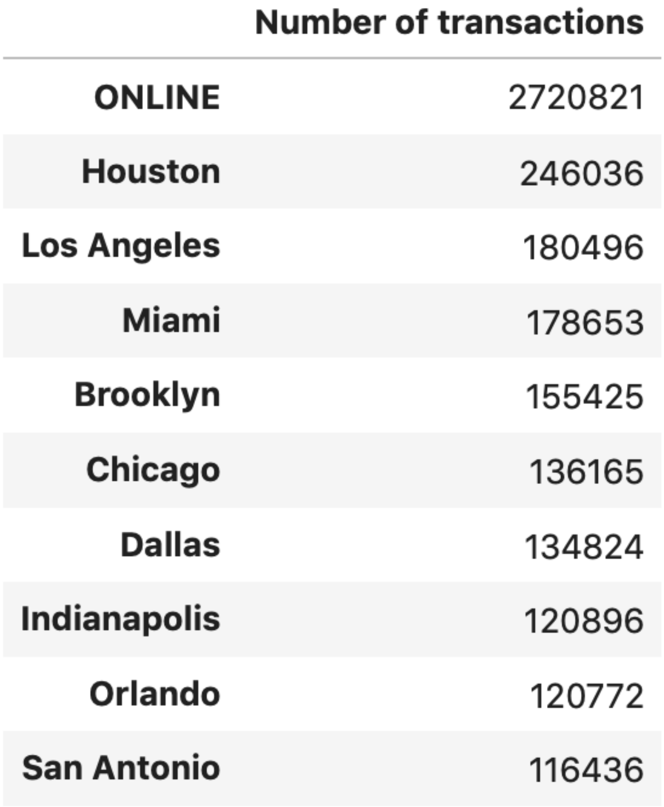
	User	Card	Year	Month	Day	Time	Amount	Use Chip	Merchant Name	Merchant City	Merchant State	Zip	MCC	Errors?	Is Fraud?
count	2.438690e+07	2.438690e+07	2.438690e+07	2.438690e+07	2.438690e+07	24386900	24386900	24386900	2.438690e+07	24386900	21666079	2.150876e+07	2.438690e+07	388431	24386900
unique	NaN	NaN	NaN	NaN	NaN	1440	98953	3	NaN	13429	223	NaN	NaN	23	2
top	NaN	NaN	NaN	NaN	NaN	12:31	$80.00	Swipe Transaction	NaN	ONLINE	CA	NaN	NaN	Insufficient Balance	No
freq	NaN	NaN	NaN	NaN	NaN	30604	250984	15386082	NaN	2720821	2591830	NaN	NaN	242783	24357143
mean	1.001019e+03	1.351366e+00	2.011955e+03	6.525064e+00	1.571812e+01	NaN	NaN	NaN	-4.769230e+17	NaN	NaN	5.095644e+04	5.561171e+03	NaN	NaN
std	5.694612e+02	1.407154e+00	5.105921e+00	3.472355e+00	8.794073e+00	NaN	NaN	NaN	4.758940e+18	NaN	NaN	2.939707e+04	8.793154e+02	NaN	NaN
min	0.000000e+00	0.000000e+00	1.991000e+03	1.000000e+00	1.000000e+00	NaN	NaN	NaN	-9.222899e+18	NaN	NaN	5.010000e+02	1.711000e+03	NaN	NaN
25%	5.100000e+02	0.000000e+00	2.008000e+03	3.000000e+00	8.000000e+00	NaN	NaN	NaN	-4.500543e+18	NaN	NaN	2.837400e+04	5.300000e+03	NaN	NaN
50%	1.006000e+03	1.000000e+00	2.013000e+03	7.000000e+00	1.600000e+01	NaN	NaN	NaN	-7.946765e+17	NaN	NaN	4.674200e+04	5.499000e+03	NaN	NaN
75%	1.477000e+03	2.000000e+00	2.016000e+03	1.000000e+01	2.300000e+01	NaN	NaN	NaN	3.189517e+18	NaN	NaN	7.756400e+04	5.812000e+03	NaN	NaN
max	1.999000e+03	8.000000e+00	2.020000e+03	1.200000e+01	3.100000e+01	NaN	NaN	NaN	9.223292e+18	NaN	NaN	9.992800e+04	9.402000e+03	NaN	NaN

As we can see above, some of the features contain null variables (represented as NaN) which will be handled in the following section. From the previous table, it is important to note some facts from certain features:

- There are 2,000 different users and each of them can use up to 9 cards.

- The transactions were recorded from the year 1991 to 2020.
- There are 3 types of transactions: Swipe Transaction, Online Transaction and Chip Transaction.
- The transactions were paid in 223 different states including US states and world countries.
- There are 23 different errors that occurred during the transactions.

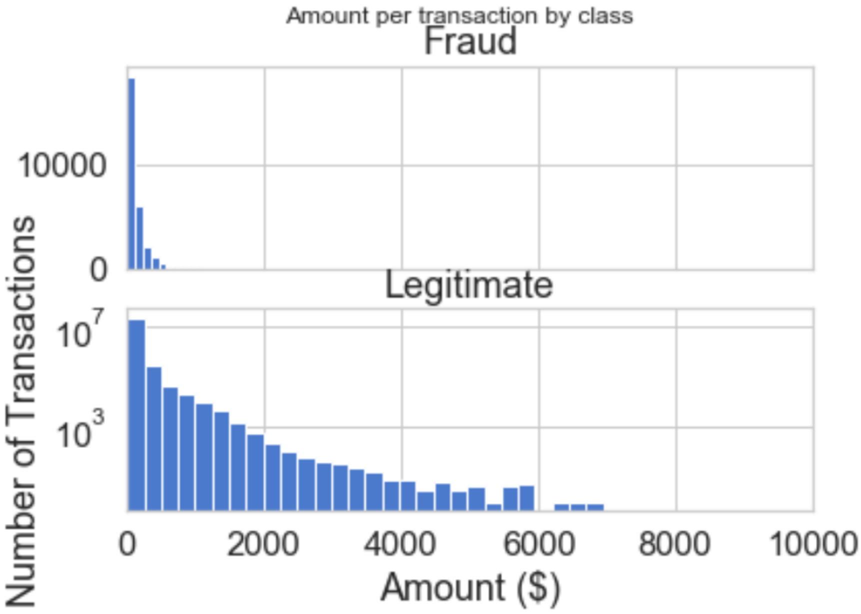
Below are the cities with the highest number of transactions. As we can see all of them are cities in the US except for the most frequent value which are online transactions:



	Number of transactions
ONLINE	2720821
Houston	246036
Los Angeles	180496
Miami	178653
Brooklyn	155425
Chicago	136165
Dallas	134824
Indianapolis	120896
Orlando	120772
San Antonio	116436

The categorical features that will have to be transformed into numerical values (since the machine learning algorithms are math-based) are: Time, Amount (removing $ character only), Use Chip, Merchant City, Merchant State, Errors?, Is Fraud?

Concerning the payment amounts, we can see below how different the amount of money are for each transaction class. All the fraudulent transactions have a very slow amount, while legitimate payments reach higher amounts:



1.2 Feature engineering

The following sub-sections cover the different steps needed to convert our data into a format that's useable by machine learning algorithms.

1.2.1 Handling of missing values

As we discovered in our descriptive analysis earlier, our dataset contains null values that have to be treated. These are the features that contain null values: Merchant State, Zip, Errors?

It turns out that when a transaction is online, the merchant state and zip are filled with null values. In order for the machine learning algorithms to work correctly we should fill these values, and one option is to use the same "ONLINE" string as for the Merchant City.

We found there are over 157,000 in-person transactions without a Zip code. We are going to fill them following this approach:

1. Get the most frequent Zip from the transactions for the same merchant name and city.
2. Get the most frequent Zip from the transactions for the same merchant city only.
3. Get the most frequent Zip from the transactions for the same merchant state.

4. Get the most frequent Zip from all the transactions.

Finally, when a transaction is legitimate, the field "Errors?" is null. We'll fill these values with an "OK" string.

1.2.2 Handling of categorical values

As mentioned earlier there is a set of features that are considered categorical instead of numeric:

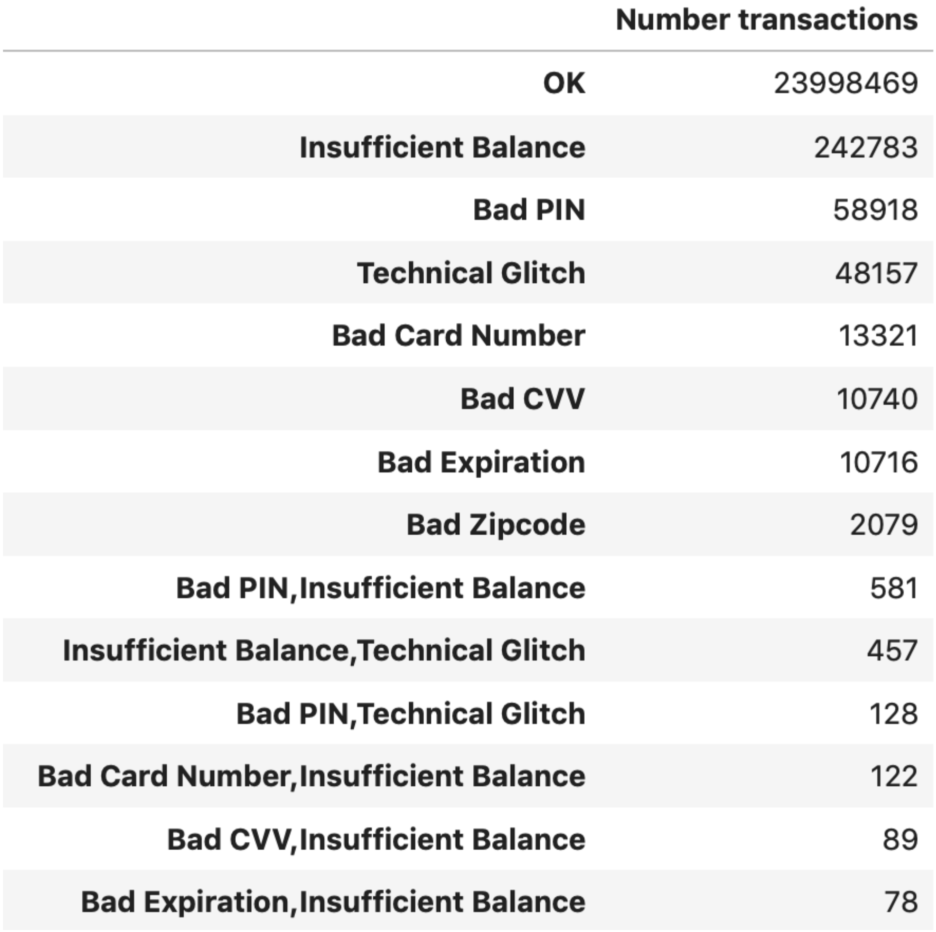
- Nominal: Use Chip, Merchant City, Merchant State, Errors?, Is Fraud?
- Ordinal: Time, Amount

It is clear that Amount should be numerical and the only reason it's been detected as categorical is due to the currency symbol so we'll drop this character since all the amounts are in USD and convert them into a float numerical field. Once this feature is numerical, we can see there is a group of transactions wih a negative amount which looks like an error so we’ll drop this rows with Amount below 0.

The time can be converted into an hour-only integer feature since minutes will have low or no impact on the fraud detection.

The "Use Chip" variable can be easily converted with one-hot encoding since there are only 3 types of transaction: swipe, online and chip.

We are going to apply the same technique on the "Errors?" feature since there is a limited amount of error types. However, as shown below, some transactions have multiple errors combined into one single error string. This would lead to a loss of potentially valuable information if we apply one-hot encoding directly without splitting the errors into single-error boolean features. Therefore we will have to apply it manually.

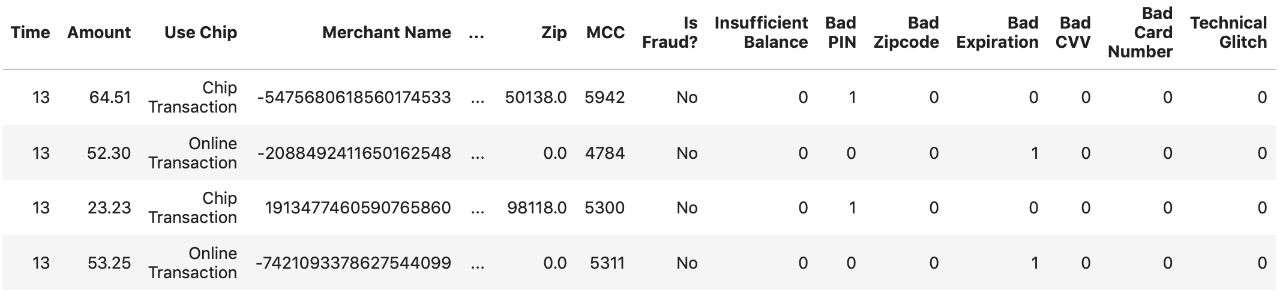


	Number transactions
OK	23998469
Insufficient Balance	242783
Bad PIN	58918
Technical Glitch	48157
Bad Card Number	13321
Bad CVV	10740
Bad Expiration	10716
Bad Zipcode	2079
Bad PIN,Insufficient Balance	581
Insufficient Balance,Technical Glitch	457
Bad PIN,Technical Glitch	128
Bad Card Number,Insufficient Balance	122
Bad CVV,Insufficient Balance	89
Bad Expiration,Insufficient Balance	78

It turns out that there are only 7 different errors that can occur during a transaction. We'll create a different dummy column for each of them:

- Insufficient Balance
- Bad PIN
- Bad Zipcode
- Bad Expiration
- Bad CVV
- Bad Card Number
- Technical Glitch

Below we can see some transactions with the new error columns:

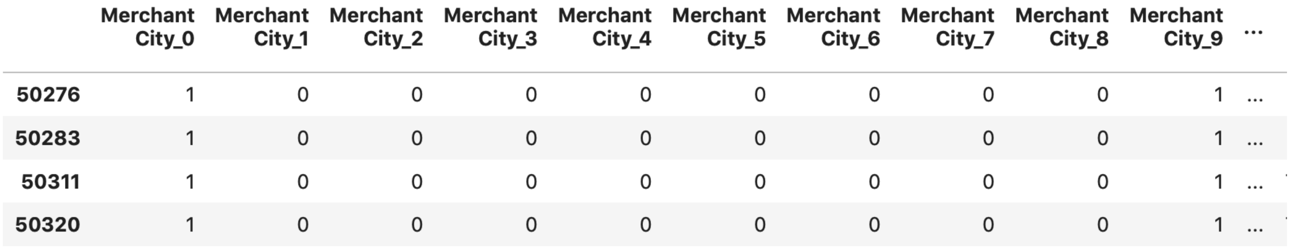


Time	Amount	Use Chip	Merchant Name	...	Zip	MCC	Is Fraud?	Insufficient Balance	Bad PIN	Bad Zipcode	Bad Expiration	Bad CVV	Bad Card Number	Technical Glitch
13	64.51	Chip Transaction	-5475680618560174533	...	50138.0	5942	No	0	1	0	0	0	0	0
13	52.30	Online Transaction	-2088492411650162548	...	0.0	4784	No	0	0	0	1	0	0	0
13	23.23	Chip Transaction	1913477460590765860	...	98118.0	5300	No	0	1	0	0	0	0	0
13	53.25	Online Transaction	-7421093378627544099	...	0.0	5311	No	0	0	0	1	0	0	0

The next two categorical variables are going to be treated in the same way. Both Merchant City and Merchant State have a high cardinality (13,429 and 224, respectively), so applying one-hot encoding on them would create far too many columns which might lead to overfitting of the future tree-based model. Therefore we are going to convert them into numeric features via binary encoding, by which each category is converted into binary digits and each digit creates one feature column. The benefit of binary ecoding is that only *lnX* columns are needed, where X is the cardinality of the feature.

We can make use of the library category_encoders for the binary encoding.

Below we can see some of the new city columns resulting from the binary encoding:



	Merchant City_0	Merchant City_1	Merchant City_2	Merchant City_3	Merchant City_4	Merchant City_5	Merchant City_6	Merchant City_7	Merchant City_8	Merchant City_9	...
50276	1	0	0	0	0	0	0	0	0	1	...
50283	1	0	0	0	0	0	0	0	0	1	...
50311	1	0	0	0	0	0	0	0	0	1	...
50320	1	0	0	0	0	0	0	0	0	1	...

Finally, for the target variable "Is Fraud?", we can just map it to an integer where 0 represents a legitimate transaction and 1, a fraudulent transaction.

With the above transformations, all our features are now numerical and can be used to train a machine learning model.

1.3 Feature selection

Most likely not all of our features contribute to the determination of fraud. Even though one might think increasing the size of the feature vector is expected to provide more discriminating power, in practice, excessively large feature vectors significantly slow down the learning process as well as cause the fraud detection classifier to overfit the training data.

With feature selection we would try to select the minimum subset of variables with which the classification accuracy will not significantly decrease and we'll predict a fraud detection distribution that is as close as possible to the original class distribution given all features.

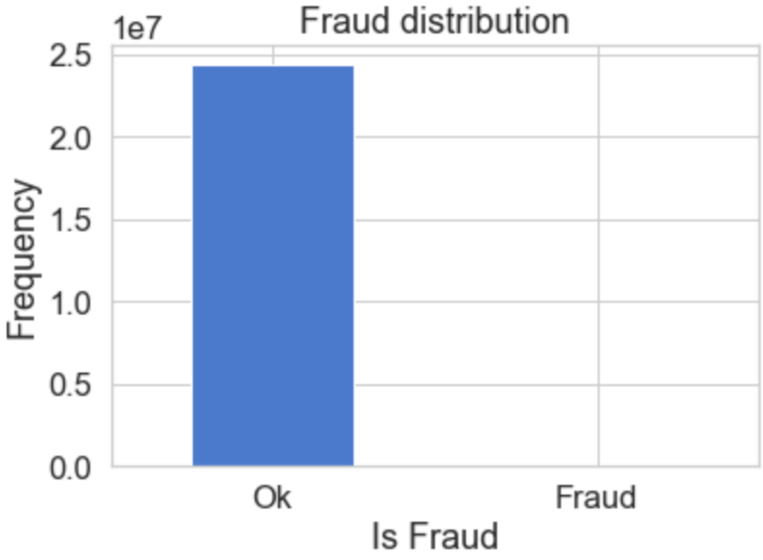
Even though I have not applied it here, below are some feature selection methods that we could use for the fraud detection problem:

- Correlation between variables, where we keep the features that have the highest correlation with the fraud target.
- Lasso (L1) regularisation, which drives features towards zero making the model less complex, the higher the λ parameter the more features it will drop.
- Wrapper methods (very expensive), which in practice are search methods that iteratively use a subset of features, training and evaluating the models after each subset.

1.4 Data balancing

When dealing with a classification problem it is important to make sure the dataset is balanced to a minimum extent and that some target classes are not extremely larger in proportion to others. Otherwise, the model could be biased towards systematically detecting the majority class and forgetting about the minority classes. Therefore we should first check the distribution of fraudulent and legitimate transactions in our dataset:

Transaction type	**Number of transactions**	**Ratio**
Legitimate	24357143	99.88%
Fraudulent	29757	0.12%

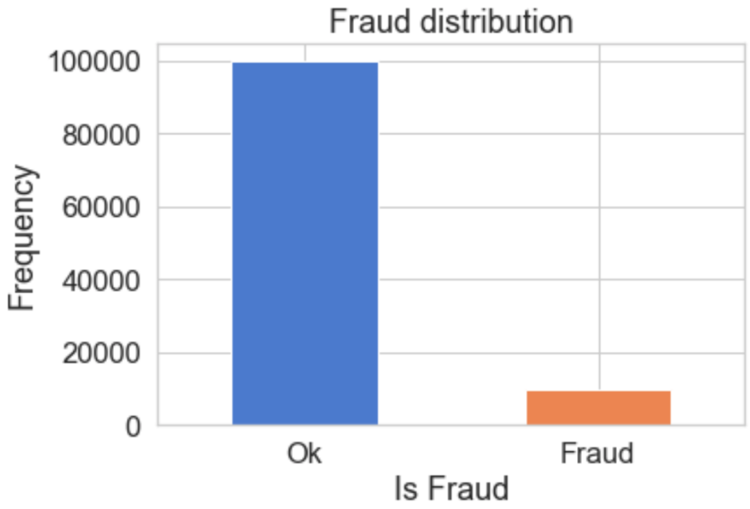


It seems that we have a highly imbalanced dataset since legitimate transactions overwhelm the fraudulent ones by a large margin. This imbalance of the target class might decrease the performance of the classification algorithm (the model might fail to identify fraud) so we'll need to create more samples of the minority class i.e. the fraudulent transactions.

In particular, we are going to apply an oversampling method called Synthetic Minority Oversampling Technique (SMOTE) which takes into account characteristics of the fraudulent class to create synthetic duplicates. The objective is to increase the ratio of the fraudulent class from 0.12% to 10% of the total transactions.

We can use the library imbalanced-learn for this task.

Below we can see that the distribution of fraudulent and legimitate transactions has now changed to have a ratio of 90-to-10:



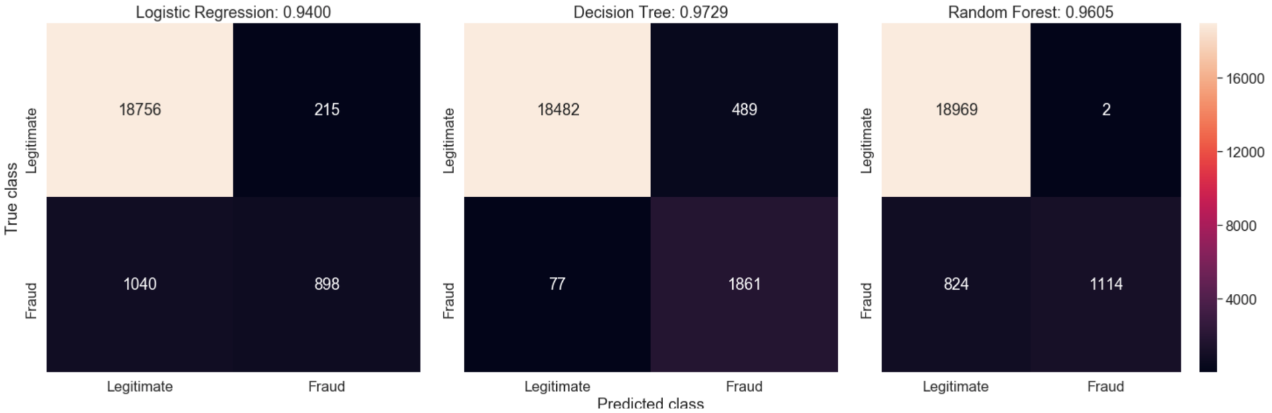
1.5 Modelling and evaluation

We are going to try 3 different classification algorithms (logistic regression, decision tree and random forest) but fefore that, we need to split our data into train (80%) and test (20%), and normalize it with the standard scaler.

It is good practice to evaluate different algorithms in order to find the one that best predicts the test class so we'll be trying the following types:

Algorithm	**Hyperparameters**
Logistic Regression	-
Decision Tree	Max depth = 5
Random Forest	Max depth = 5, Number estimators = 50

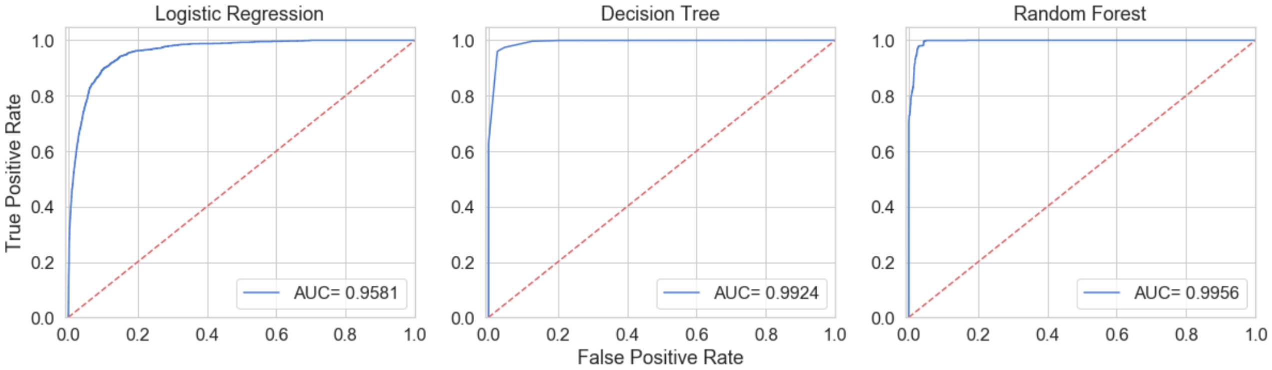
Now let's run the different algorithms and generate their confusion matrices. Confusion matrix is an essential evaluation method for classification problems such as our fraud detection system. With this method we can easily visualise the false/true negatives and positives, which will help us understand the precision and recall of our model.



Note: a smaller dataset has been used for this part due to lack of computer resources.

As we can see above, our model managed to correctly detect most of the trully legitimate transactions (majority class), while performing more poorly on the fraudulent transactions (minority class).

Even though the plots above can give us a good summary of our models performance, we should dive deeper specially taking into account how imbalanced our dataset was originally. One way to investigate further is by using ROC curves and AUC score to understand the performance of our binary classifier.



The charts above play an important role at explaining how our model is able to distinguish between the two classes (fraudulent and legitimate), the higher the AUC, the better our model is at predicting these classes. We can clearly see a difference between the logistic regression model (AUC=0.958) and the two tree-based models (AUC>0.992).

As for the ROC curves, we can find that the area above the curve for the random forest model is clearly smaller than for the other algorithms, and far away from the diagonal line that indicates a random guess.

We can state that random forest is the algorithm that performs best for our problem. A possible enhancement would be using grid search for tuning the hyperparameters such as number of trees and maximum depth of them.

2 Region analysis

An important issue to be considered when deciding in which regions to extend our new business is how safe the payment transactions generally are in those countries. We can solve this question by grouping all fraudulent transactions from our dataset by their merchant state.

Since we will be performing an analysis at country level, we are going to treat all the transactions from an US state as part of the United States of America. Therefore we will convert all 2-letter US state names into the string "United States of America". This conversion reduces the list of our countries from 223 “states” to 172 countries. Furthermore, online transactions are out of the scope of this analysis since they didn’t take place in a physical country.

There are two ways to look at payment fraud at country level: one way is to find the percentage of fraudulent transactions over all transactions done in each country, and the other way is to consider the full amount that was defrauded in each country.

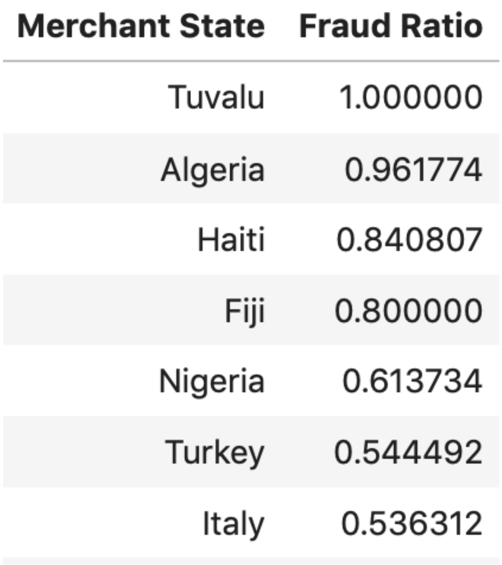
These two different analysis are assisted below by colour-coded world maps that have been plotted using the Python library Folium.

2.1 Fraud ratio by country

We can easily calculate the fraud ratio of each country by dividing the number of fraudulent transactions by the number of total transactions. After a quick analysis, we can find that there is a group of countries where the majority of their payment transactions were considered fraudulent. From my point of view, there are two surprises here:

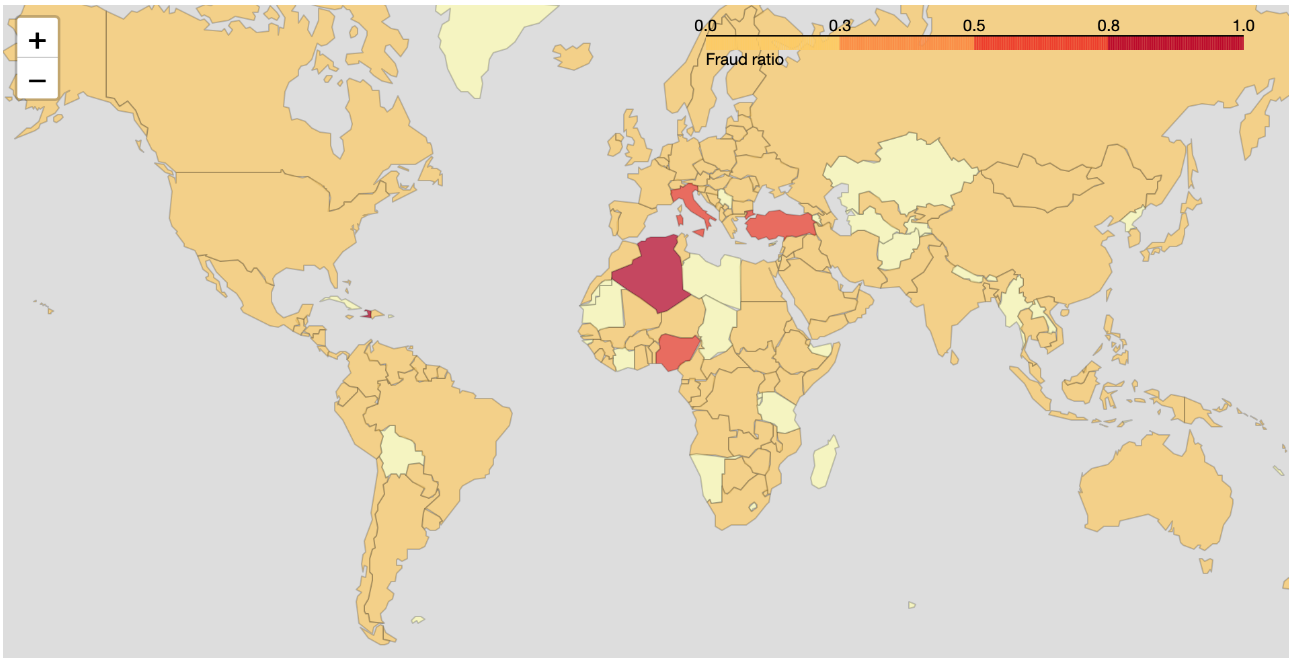
- The island country of Tuvalu, in the Pacific ocean, had all their 59 payment transactions detected as fraudulent, and not a single transaction was legitimate.
- Italy is the only developed western country where fraud was commited in the majority of their payment transactions. It doesn’t seem to be a problem of low amount of transactions since almost 9,000 payment transactions were registered in Italy.

Below we can see a more detailed view of the most fraudulent countries:



Merchant State	Fraud Ratio
Tuvalu	1.000000
Algeria	0.961774
Haiti	0.840807
Fiji	0.800000
Nigeria	0.613734
Turkey	0.544492
Italy	0.536312

In order to get a wider picture of the fraudulent transactions in the world, we can see that generally African and mediterranean countries, as well as some remote islands, tend to be more affected by this type of fraud. It is also interesting to note that some important countries such as Bolivia or Cuba did not register any payment transaction.

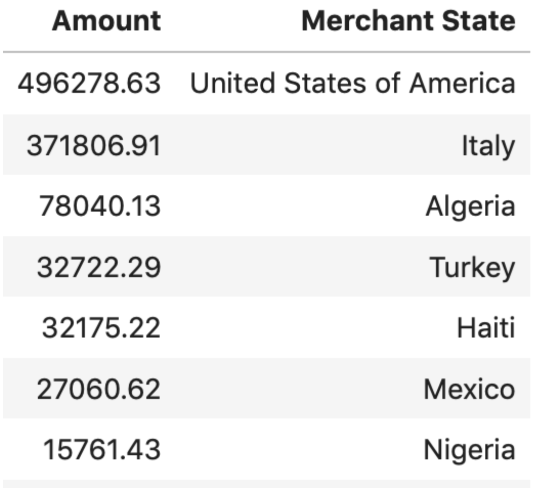


In general, apart from the mentioned countries, the ratio of fraudulent transactions is similar in most parts of the world. However, in the next section we'll change the perspective and look at this problem considering the defrauded amount instead of number of transactions.

2.2 Fraud amount by country

It might also be interesting to find out what are the amounts (in USD) for which fraud has been commited. Perhaps only a few fraudulent transactions have been commited in a country but if those transactions included large amounts of money then it will be relevant for our analysis.

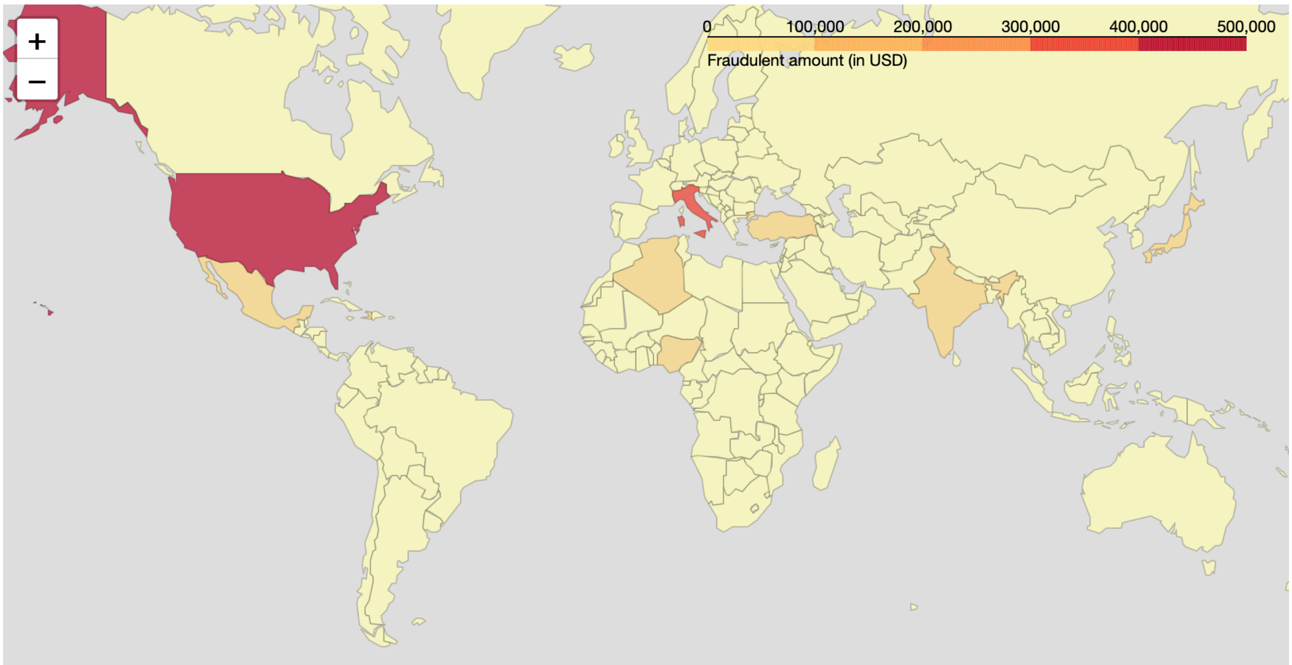
As the following table points out, two new countries appear now that, even though their ratio of fraudulent transactions was not large, their defrauded amount is among the highest in the world. Particularly, USA registered almost $0.5Mill of payment fraud, while Mexico registered just over $27,000. Again, Italy is in the top of the list so that means not only the fact that most of their payment transactions were fraudulent but also that the defrauded amount is large.



Amount	Merchant State
496278.63	United States of America
371806.91	Italy
78040.13	Algeria
32722.29	Turkey
32175.22	Haiti
27060.62	Mexico
15761.43	Nigeria

Plotting a map of defrauded money in the world, we can now see new regions being hit by large amount of fraud such as north/central America with USA and Mexico, India and Japan.

A region of the world that doesn’t seem to be hugely affected by payment fraud is South America.

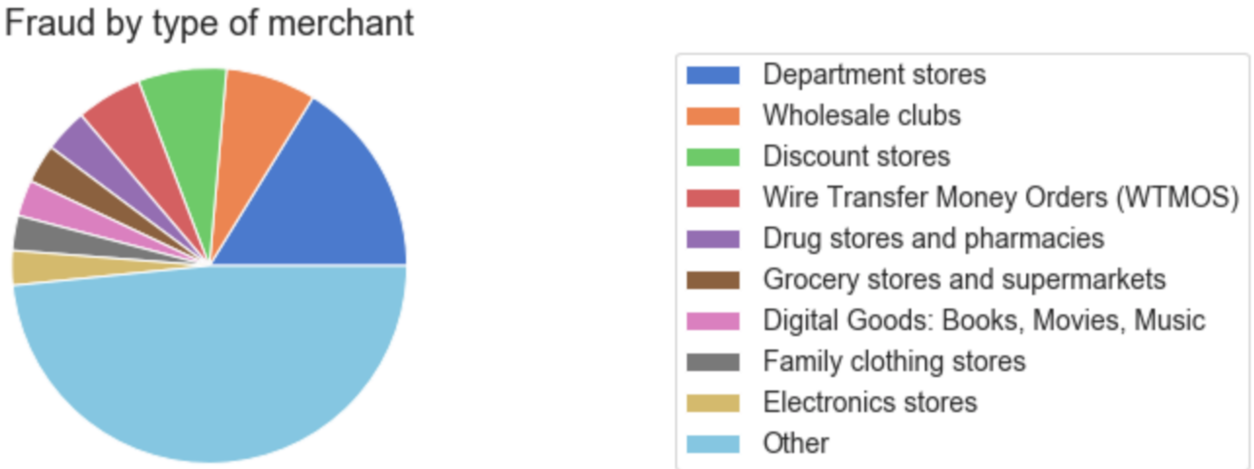

+
−
0
100,000
200,000
300,000
400,000
500,000
Fraudulent amount (in USD)

3 Merchant categories

If we think that financial fraud affects equally to all types of sectors we are probably wrong. Either because of the nature of the business or because of the operations of different merchants, we'll see below how certain categories of merchant are more likely to suffer fraud.

Since our dataset only contains a variable named "MCC" (Merchant Category Code) which is a 4-digit code we'll need to convert that into a proper description of the merchant categories. We can achieve this by using a MCC table from this website. Instead of downloading this table in a file, we're going to use a technique called web scrapping for reading and processing the HTML code from this table, via the Python library BeautifulSoup4. After processing this table, we find out there exist 365 different categories of merchant.

The next step is to group our transactions by MCC and find out how many of those 365 codes are actually present in our dataset, which turns out to be 109 categories. We'll sort them by number of fraudulent transactions and display only the top most categories:



As we can see above, there is clearly a type of merchant which is extremely hit by fraud. We can state that it is highly recommended to install a robust security system before investing in a department store since these merchants cover around 20% of world fraudulent transactions.

Wholesale clubs, discount stores and wire transfer money orders are other types of merchants that are likely to be hit by payment fraud.

As for the rest of merchant categories, it seems there is no major difference between them when it comes to financial fraud.

4 Architecture

One way to deploy our model would be to containerise it in Docker together with a REST API service. The API would receive payment transaction one at a time and would generate a boolean output with the fraudulent/legitimate value.

We could deploy this Docker container in a public cloud provider such as AWS or GCP in order for our model to make predictions on real-time data. It is important that this container includes the pre-processing pipeline to select the right features of the incoming transactions, convert categorical features into numeric and normalise data, just as we would have done when training our model. We can create this pipeline using the Spark framework in order to parellelise the data processing.

Once the model is deployed, we should continue monitoring its performance. One way to achieve this is by configuring alarms in our cloud so that we are notified when:

- Distribution of fraudulent vs legitimate transactions goes below or above certain thresholds. For instance, if we know that the normal distribution is to have 0.1 % of fraudulent transactions and out of sudden, during the last 24 hours, our model has predicted over 1% of fraud then there is something to investigate.
- Defrauded amount for a certain country or city in the last 24 hours has gone down close to zero, when maybe the expected average of defrauded amount is 100 USD. In this case, we should investigate if our model is processing the payment amounts correctly.

Most likely the data evolves over time and we need to adapt our model to the new type of data, either currencies, new merchant types, new error types due to other payment tools, etc. In the case that we have to re-train our model, we should use recent data to make sure our model predictions are not based on obsolete data. For this purpose, we should store the payment transactions that have been processed by our system for a certain duration.

We could make use of Elasticsearch to index the transactions once they have been processed. A great benefit of this tool is that it’s natively synced with Kibana which will let us create real-time dashboards with the current distribution (fraud vs legitimate), transaction counts and more, it will even let us create alarms for monitoring our model.