Capital One Data Science Challenge - Q&A

Question 1: Load

The data is provided as a .txt file containing line-delimited JSON. Each line represents a credit card transaction and is structured as a JSON object.
There are 786,363 records and 29 fields in each record.

Some statistics for each of the fields are below, split by whether numerical or categorical. It is worth noting that while there are no null values in any of the columns, there are some columns in the categorical data which have empty strings, with some of them being entirely empty. It might be beneficial to drop them entirely during later analysis and feature engineering.

Numerical Features:

	Null Count	Unique Values	Min	Max	Mean	Median
creditLimit	0.0	10.0	250.0	50000.0	10759.464459	7500.0
availableMoney	0.0	521916.0	-1005.63	50000.0	6250.725369	3184.86
transactionDateTime	0	776637	2016-01-01 00:01:02	2016-12-30 23:59:45	2016-07-06 01:58:58.395681536	2016-07-08 05:03:57
transactionAmount	0.0	66038.0	0.0	2011.54	136.985791	87.9
accountOpenDate	0	1820	1989-08-22 00:00:00	2015-12-31 00:00:00	2014-02-03 01:11:17.352825856	2014-09-05 00:00:00

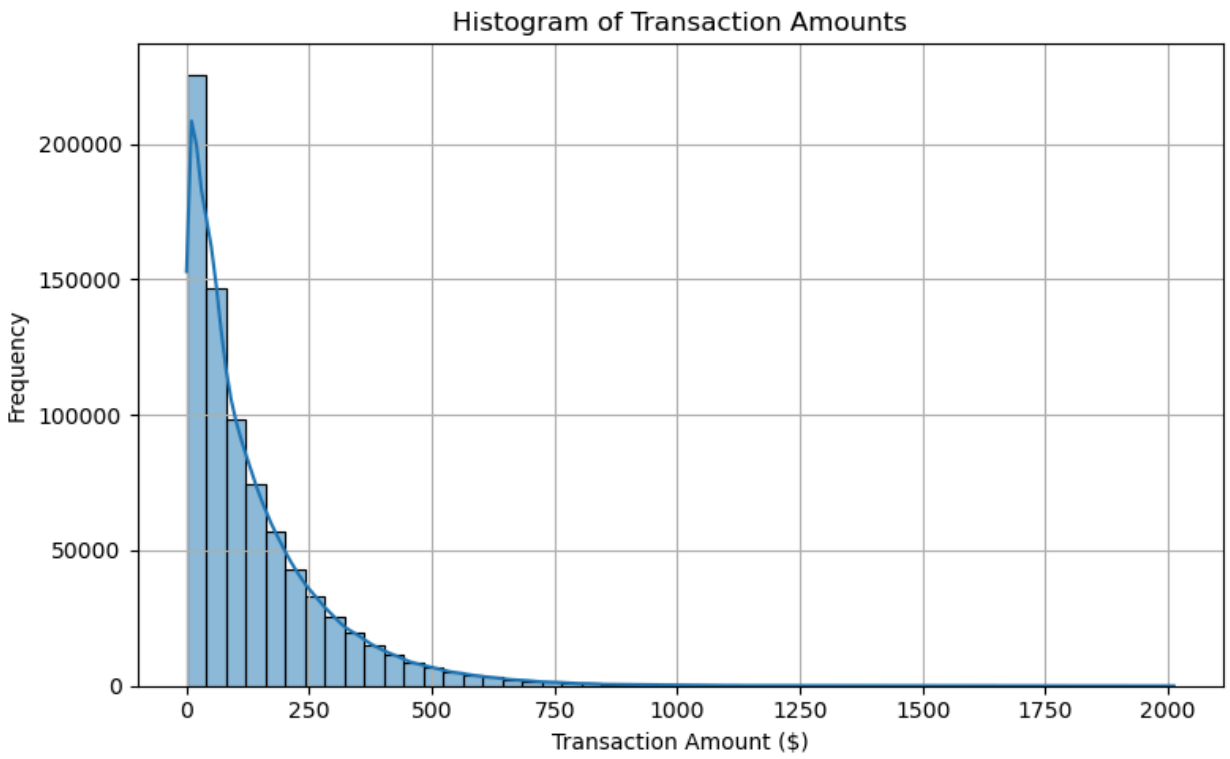
dateOfLastAddressChange	0	2184	1989-08-22 00:00:00	2016-12-30 00:00:00	2015-04-14 06:46:41.127723520	2016-01-13 00:00:00
currentBalance	0.0	487318.0	0.0	47498.81	4508.739089	2451.76
currentExpPeriod	0	165	2019-12-01 00:00:00	2033-08-01 00:00:00	2026-09-25 23:50:42.724542720	2026-10-01 00:00:00

Categorical Features:

	Null Count	Unique Values	Most Frequent	Frequency Count	Empty String Count
accountNumber	0	5000	380680241	32850	0
customerId	0	5000	380680241	32850	0
merchantName	0	2490	Uber	25613	0
acqCountry	0	5	US	774709	4562
merchantCountryCode	0	5	US	778511	724
posEntryMode	0	6	05	315035	4054
posConditionCode	0	4	01	628787	409

merchantCategoryCode	0	19	online_retail	202156	0
currentExpDate	0	165	03/2029	5103	0
cardCVV	0	899	869	33749	0
enteredCVV	0	976	869	33424	0
cardLast4Digits	0	5246	593	32946	0
transactionType	0	4	PURCHASE	745193	698
echoBuffer	0	1		786363	786363
merchantCity	0	1		786363	786363
merchantState	0	1		786363	786363
merchantZip	0	1		786363	786363
cardPresent	0	2	False	433495	0
posOnPremises	0	1		786363	786363
recurringAuthInd	0	1		786363	786363
expirationDateKeyInMatch	0	2	False	785320	0
isFraud	0	2	False	773946	0

Question 2: Plot



The histogram is right skewed, with most transactions clustering towards the left (median=$87.9). The longer tail extends towards higher amounts, with purchases of over $750 being very rare. The peak of the histogram is at very low amounts, indicating most transactions are probably less than $20.

I suspect this is because consumers tend to spend smaller amounts more frequently rather than making large purchases regularly. These smaller transactions could be recurring daily/weekly purchases like groceries, transit, fast food, coffees as opposed to larger purchases which are often non recurring.

Question 3: Data Wrangling - Duplicate Transactions

1. I was able to programmatically identify reversals of purchases as well as multiswipes (within a threshold of 3 minutes).

2. Total instant reversals identified: 13016
   Dollar Amount for identified instant reversals: $1,900,955.97
   Total Multiswipes identified: 7457
   Dollar Amount for identified multiswipes: $1,104,006.71

3. Out of the 7457 multiswipe transactions totaling $1.1M, only 152 were reversed, totaling $25,849.02. This means that multi-swipes happen often and rack up over a million dollars, but almost never get reversed. This could be the customers tolerating small accidental charges or them simply not noticing them, but the vendors earn a lot of money through these multi-swipes.

Question 4: Model

Data Cleaning & Feature Engineering

I started by removing empty, redundant, and duplicate columns, then engineered a set of behavioral and contextual features. My final features included:

- Match checks (e.g., whether the entered CVV matches the real one, or if the acquiring country matches the merchant's).
- Account-related timing, like days since account opening or last address change.
- Time-based behavior, such as part of day, day of week, hour, and whether the transaction was the first of the day.
- Rolling transaction activity, like transaction counts and total spend over 1-hour, 24-hour, and 7-day windows.
- Merchant patterns, including whether the customer had seen the merchant before, how frequently a merchant appears across the dataset, and how diverse the customer's spending categories were in the past week.
- Behavioral baselines, such as the customer's typical transaction hour, average spend, and deviation from these norms.
- Cumulative fraud history per customer.

These features were designed to capture both short-term spikes in behavior and longer-term patterns, which are often strong indicators in fraud detection.

Sampling Strategy

Given the heavy class imbalance in fraud data, I sampled:

- All fraud transactions for each customer who has experienced fraud.
- A 7.5% sample of non-fraud transactions from those same customers, to retain relevant context.
- A 1% sample of non-fraud transactions from customers who have never experienced fraud, to introduce noise and avoid overfitting to a fraud-heavy subset.

This was selected after some trial and error with the 3 models.

Modeling Approach

I split the data into train, validation, and test sets (60/20/20). Features were categorized into numeric, ordinal, and categorical, and transformed using a sklearn ColumnTransformer.

I experimented with three models: Logistic Regression, Random Forest, and HistGradientBoostingClassifier.

- Logistic Regression
- Random Forest
- HistGradientBoostingClassifier (HGB)

Logistic Regression was a good starting point because it's simple and easy to interpret. Random Forest helped capture more complex patterns in the data, but was ultimately overfitting. In the end, HistGradientBoostingClassifier worked best — it was fast and was great at picking up subtle fraud signals.

Each pipeline included a VarianceThreshold feature selector, and I ran grid search to optimize ROC AUC. I chose to optimize for ROC AUC because it gives a balanced view of how well the model separates fraud from non-fraud across all classification thresholds. This allowed me to later fine-tune the decision threshold specifically for F1 score, depending on the trade-off between false positives and false negatives.

Threshold Tuning

Since fraud is rare, raw predicted probabilities are usually low, and using a default 0.5 threshold would miss most fraud cases. After selecting the best model (HistGradientBoostingClassifier with a ROC AUC of 87.42), I fine-tuned the threshold using the validation set by sweeping values from 0.0 to 1.0, selecting the threshold that maximized F1 score to balance precision and recall. This threshold came out to be 0.26, showing that the model is not very confident about fraud classifications simply because of the overwhelming number of non-fraud samples.

Final Metrics on Test Set

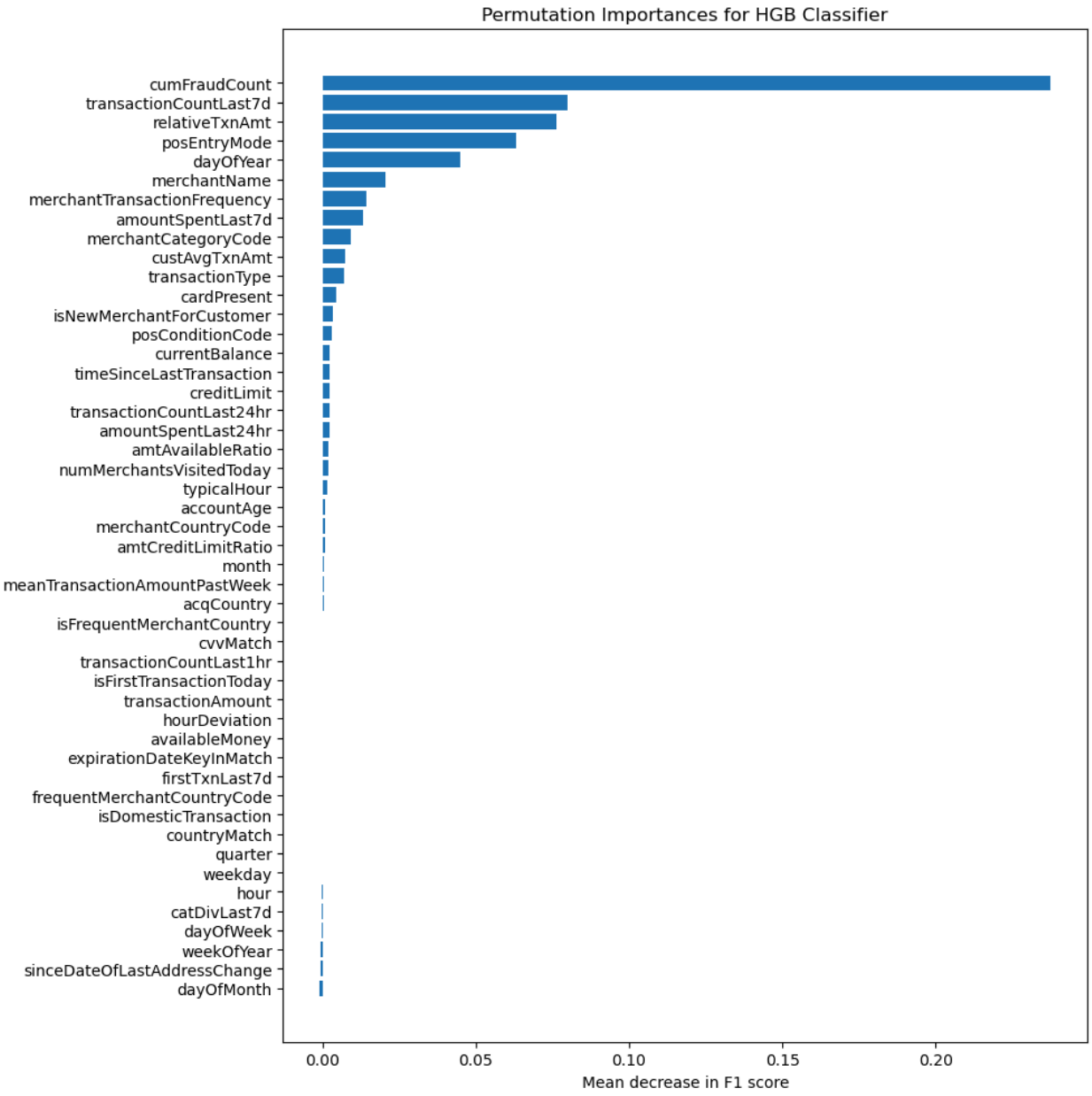
With the selected threshold, the final performance on the test set was:

Precision: 0.53
Recall: 0.75
F1-score: 0.62
FPR: 0.14

The model effectively identifies most fraudulent cases while keeping false positives at a manageable level.

Feature Importance

Since HistGradientBoostingClassifier doesn't expose importances directly, I used permutation importance and implemented a custom model wrapper to ensure it used my fine-tuned threshold (since the default is 0.5).



Key findings:

- cumFraudCount was the most predictive feature, showing that prior fraud involvement is a strong signal — likely reflecting either high-risk customers or compromised accounts.
- transactionCountLast7d and relativeTxnAmt were also highly ranked, indicating that sudden changes in activity or abnormal spend levels are common red flags.
- posEntryMode and dayOfYear show that how and when a transaction happens can influence risk — e.g., manually entered cards and certain periods may be riskier.
- Merchant metadata like category and frequency contributed to the model but played a supporting role, providing additional context rather than being primary features.

What I would next with more time

I would explore a larger hyperparameter grid. I kept the current grid fairly limited due to the computational cost of grid search, but a broader search could help fine-tune the model further.

I'd also be interested in experimenting with attention-based models, which might be better at capturing complex transaction patterns and sequential behavior, especially in customer histories.

Additionally, I'd look into better understanding the nature of false positives and whether certain types of fraud require separate treatment. If there were more context around why a transaction was marked as fraud, it might be useful to incorporate the fraud type into the model.

What didn't work

At first, I tried sampling the data so that it was perfectly balanced between fraud and non-fraud transactions by undersampling non-fraud cases to match the number of fraud cases. The idea was to give the model an equal chance to learn both classes. But in practice, it led to very poor precision — the model started flagging too many legitimate transactions as fraud.

That kind of tradeoff wouldn't work in a real banking environment, where false alarms would frustrate customers and overwhelm fraud investigation teams. So I moved away from strict balancing and instead used a sampling approach that gave the model enough fraud examples to learn from, while keeping the number of false positives at a more manageable level.