Big Data Group 9 Project
Real-Time Credit Card Fraud Detection

Team members:

Kwan, Lydia, l4kwan
Laberge, Steeve, slaberge
Lan, Aiqing, a6lan
Li, Ann, a86li
Li, Sigao, s82li
Li, Chun, cy3li

1. Objectives

This custom project involved the implementation of a real-time framework for credit card fraud predictions. Our specific objectives were as follows:

- Conduct standard ML pipeline development activities, including exploratory data analysis, feature engineering, and model design and evaluation as taught in the program.

- Explore the techniques for streamed transaction processing and ML predictions using Spark Streaming and Kafka.

2. Analysis

Data Source

Our source of data for this project was this synthetic (simulated) credit card fraud detection dataset published on Kaggle:

- https://www.kaggle.com/datasets/kartik2112/fraud-detection/data

It would have been extremely difficult to find real fraud data. However, this dataset was considered by many as quite realistic and rich in patterns. As such, we felt it would be adequate to support the data analysis and ML pipeline development phase of the project. The dataset came in the form of a CSV file containing more than 1.2 million records. As far as preparation goes, we designed a schema to convert date and numerical features to the proper data types for analysis. The schema was also a key component for the real-time prediction component of the project, allowing the streaming dataframe to ingest and transform events the same way we did for the initial load and ML training.

Exploratory Data Analysis

Our preliminary scan of the dataset showed that there were no missing values or duplicate values. The features of interest to us had a relatively normal distribution with the exception of city population and the amounts involved in transactions. We have therefore applied logarithmic transformations to these elements as part of the pipeline.

Unsurprisingly, the target class, “is_fraud”, was heavily imbalanced. Less than 1% of transactions are fraudulent (Figure 1). We had anticipated that this would be the case. As such, at the onset of the project, we made plans to assign weights to the target class.

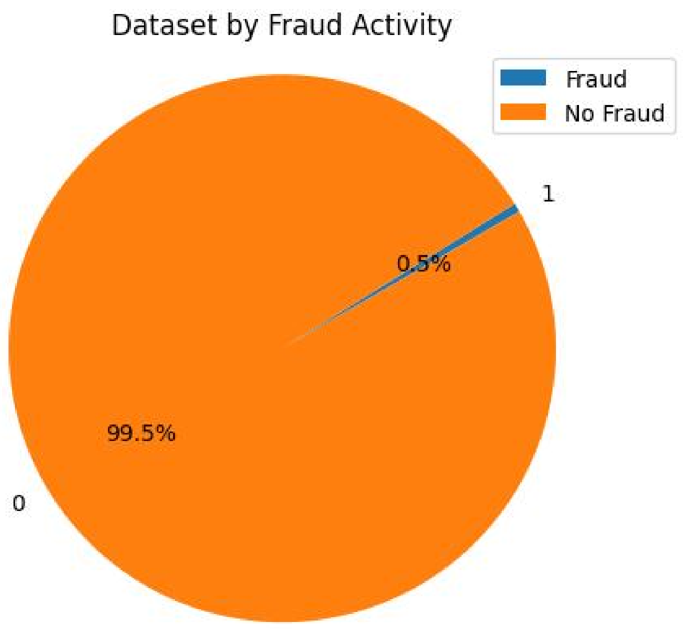


Figure 1. Dataset group by Fraudulent Activity (0 - No Fraud, 1 - Fraud)

Fraud by Hour of the Day

Notably, there is a significant increase in fraud incidents during late-night hours, between 10:00 PM and 12:00 AM. Another surge in fraudulent activities is observed in the early morning, especially between 1:00 AM and 3:00 AM (Figure 2). These patterns suggest that fraudsters might be exploiting the late-night and early morning hours, possibly capitalizing on the likelihood of reduced scrutiny from cardholders and financial institutions alike.

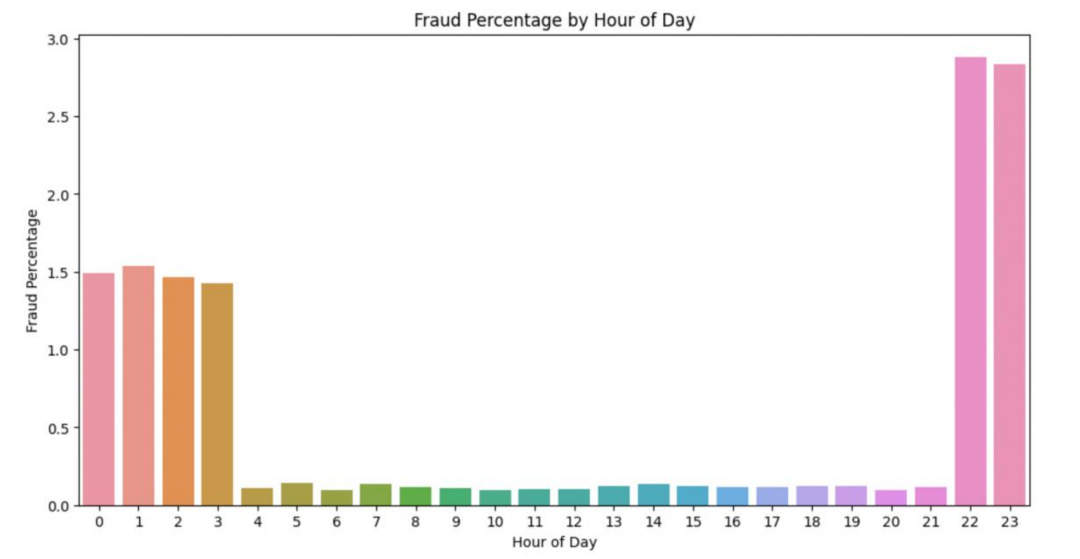


FIgure 2. Fraudulent Activity by Hour of Day

Fraud per Shopping Category

Online shopping and miscellaneous internet-based transactions exhibit the highest rates of fraud, everyday transaction categories, such as groceries and retail shopping, both online and at physical points of sale, also attract a considerable number of fraudulent activities, potentially due to their routine nature and lower individual transaction scrutiny (Figure 3). Conversely, sectors associated with lifestyle, essentials, and niche services, like travel, dining, and health, demonstrate lower fraud rates, possibly reflecting more robust security measures or less targeting by fraudsters due to higher risk and lower volume.

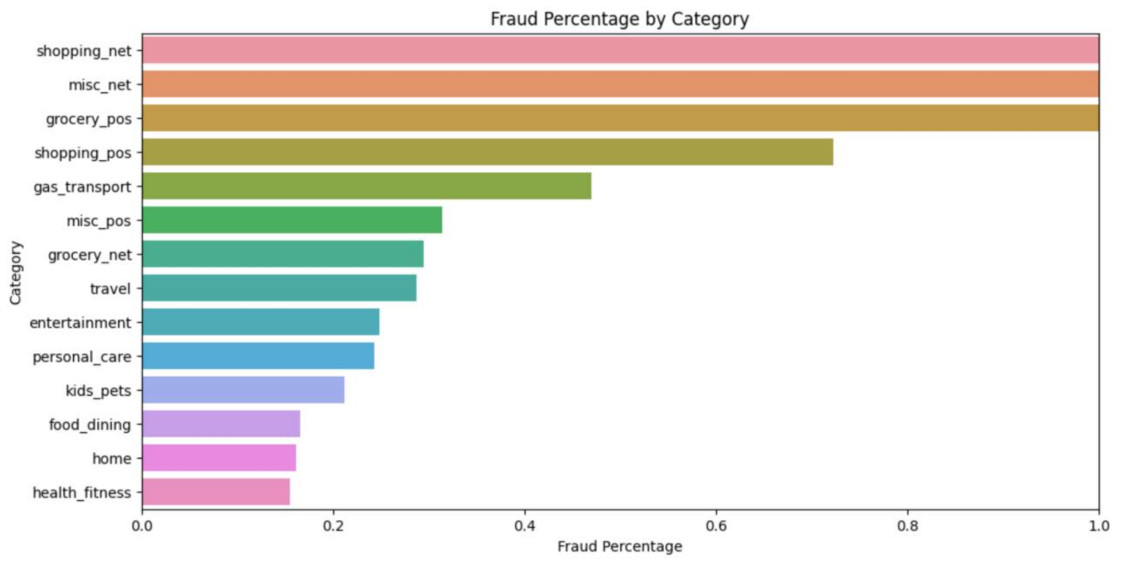


Figure 3. Fraudulent Activity by Shopping Category

Fraud by Vendor

The top five list of merchants with the highest fraud percentages — Kozey-Boehm, Herman LLC, Treutel-Dickens, Kerluke-Abshire, and Brown PLC — suggests that these vendors may be more susceptible to fraudulent transactions or targeted by fraudsters (Figure 4). These findings could be a signal to payment processors and financial institutions to apply more stringent fraud checks for transactions involving these vendors.

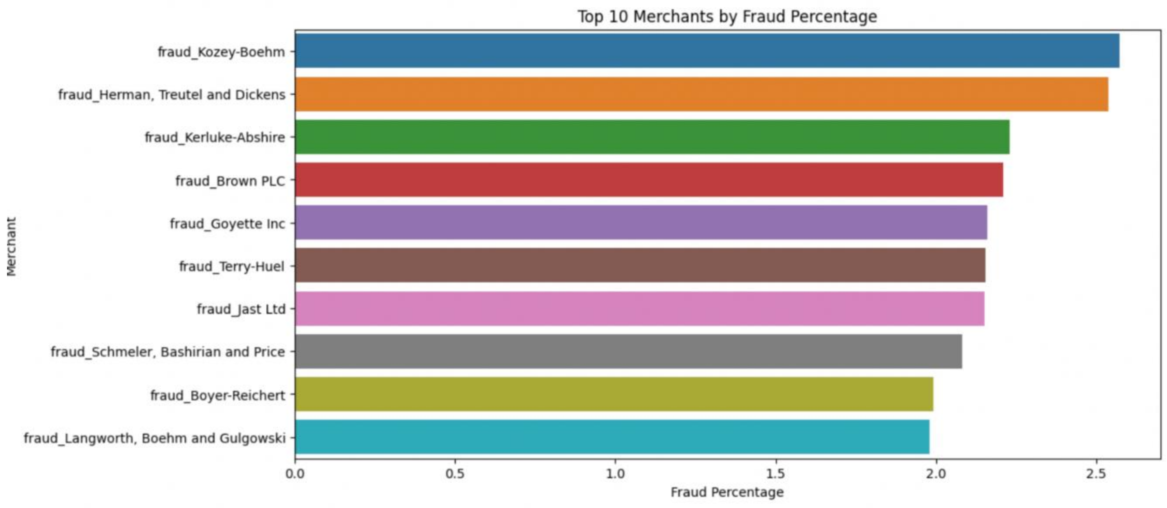


Figure 4. Fraudulent Activity by Vendor

Fraud Ratio by Age Group

Is fraud more prevalent in certain age groups than others? The analysis below suggests that seniors aged 80+ are more exposed to fraudulent transactions than other groups (Figure 5). This made age a potentially viable feature for the machine learning pipeline.

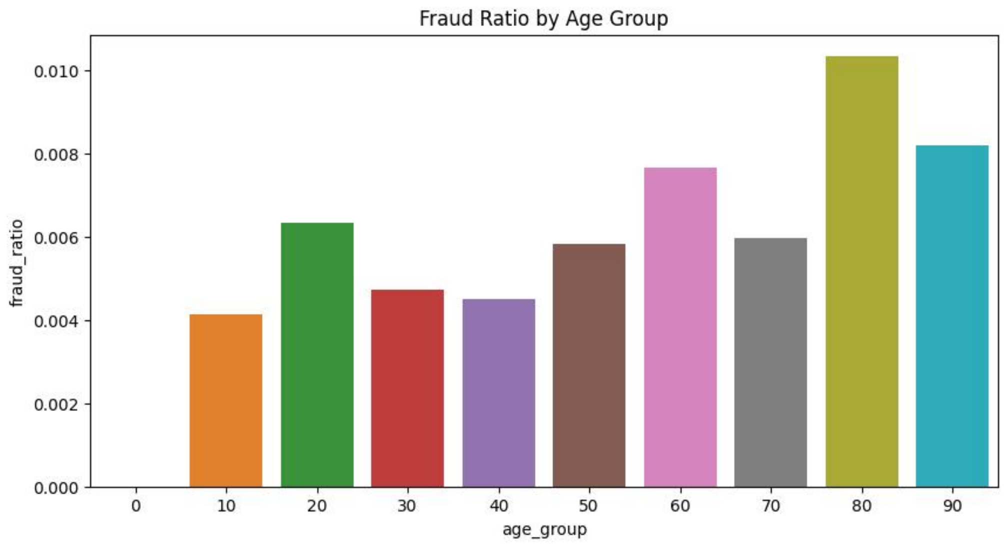


Figure 5. Fraudulent Activity by Age Group

Fraud by Age Group and Gender

We took a further look into age groups and split by gender to identify potential patterns in fraudulent activities (Figure 6). The analysis revealed that males in the age group 50 exhibited the highest incidence of fraudulent transactions whereas fraudulent activity was highest for females in the age group 30. Surprisingly, this trend indicates the lowest number of fraudulent activities among seniors, contradicting our initial observation. It is important to note that these analyses focus on the sheer number of fraudulent transactions and not the ratio of fraudulent transactions to total transactions. Both analyses yielded intriguing insights.

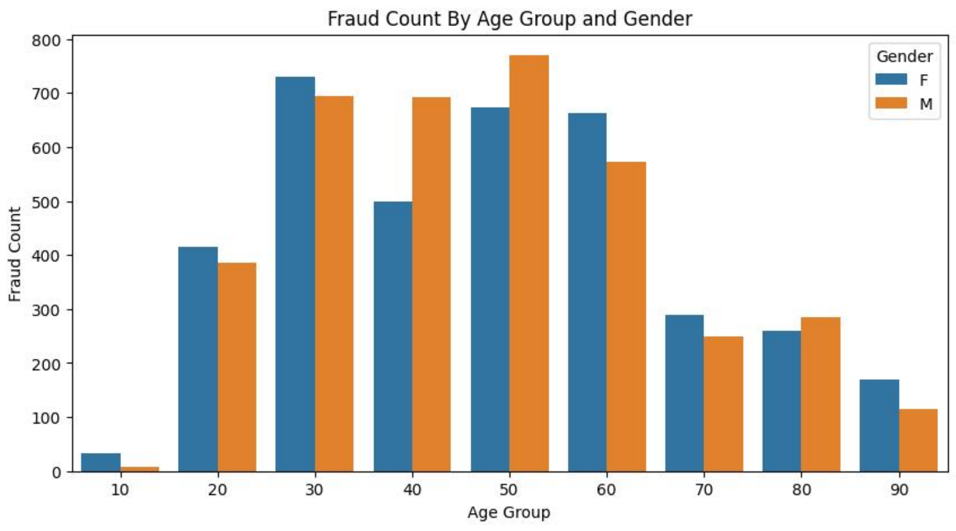


Figure 6. Fraudulent Activity by Age Group and Gender

Fraud by Transaction

Lastly we delved into fraudulent activity based on transactional amount and discovered that fraud activities were most prevalent for transactions under $1500 (Figure 7). Percentage of fraud was highest for transactions below $100 and peaked again between $200-$400 before tapering down. Transactions between $800-$1100 also exhibited a higher percentage of fraud. The concentration of fraudulent activities in transactions below $1500 suggests that fraudsters may be more inclined to target smaller-value transactions (such as those under $100). This could be because smaller transactions are less likely to trigger suspicion and may be overlooked.

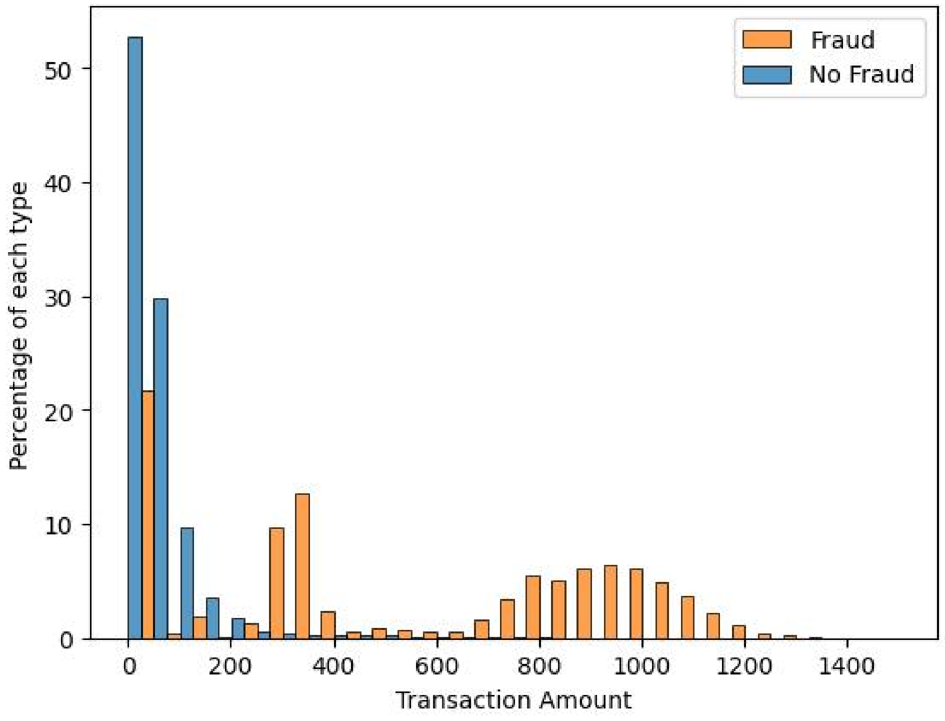
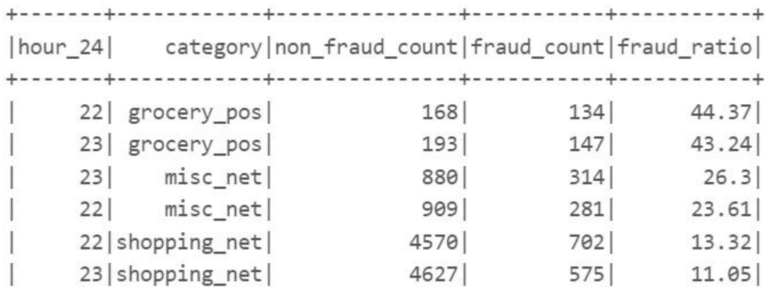


Figure 7. Fraudulent Activity by Transactional Amount

Feature Engineering

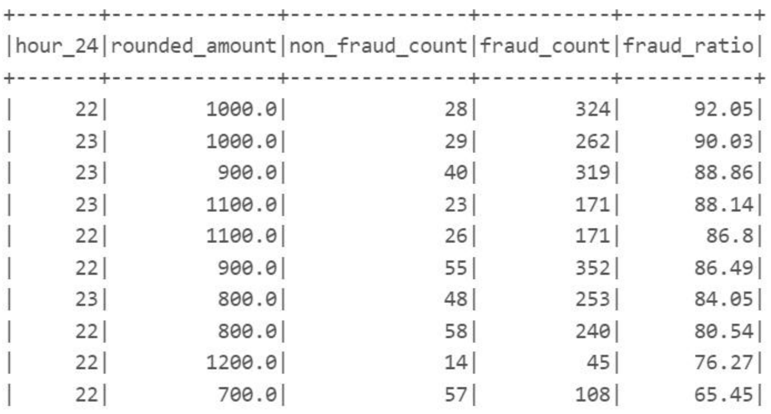
Though some of the individual features alone result in a high fraud transaction fraud rate, we would often see a high false positive rate if a rule is created using only one feature. To optimize the performance of individual rules, it is often a good idea to pair features together.

In our case, a strong feature that we have identified is the hour of the day. It is very evident that around midnight, there is a high volume of fraud. However, using time of the day alone would cause our engine to over-fire.



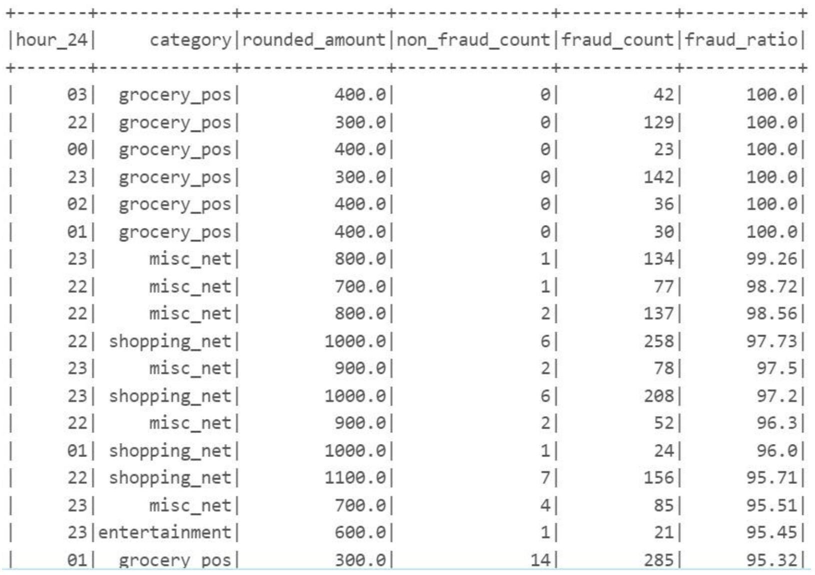
```
+-------+------------+---------------+-----------+-----------+
|hour_24|    category|non_fraud_count|fraud_count|fraud_ratio|
+-------+------------+---------------+-----------+-----------+
|     22| grocery_pos|            168|        134|      44.37|
|     23| grocery_pos|            193|        147|      43.24|
|     23|    misc_net|            880|        314|       26.3|
|     22|    misc_net|            909|        281|      23.61|
|     22|shopping_net|           4570|        702|      13.32|
|     23|shopping_net|           4627|        575|      11.05|
```

As seen above, grocery_pos, misc_net and shopping_net, along with hour_of_day being almost midnight generate a fraud rate of over 10%, which is acceptable for a fraud engine. We can use these two features to create the first rule.



```
+-------+--------------+---------------+-----------+-----------+
|hour_24|rounded_amount|non_fraud_count|fraud_count|fraud_ratio|
+-------+--------------+---------------+-----------+-----------+
|     22|        1000.0|             28|        324|      92.05|
|     23|        1000.0|             29|        262|      90.03|
|     23|         900.0|             40|        319|      88.86|
|     23|        1100.0|             23|        171|      88.14|
|     22|        1100.0|             26|        171|       86.8|
|     22|         900.0|             55|        352|      86.49|
|     23|         800.0|             48|        253|      84.05|
|     22|         800.0|             58|        240|      80.54|
|     22|        1200.0|             14|         45|      76.27|
|     22|         700.0|             57|        108|      65.45|
```

Pairing the hour_of_day with amount also gives us a very strong sense of rule creation around these two features.



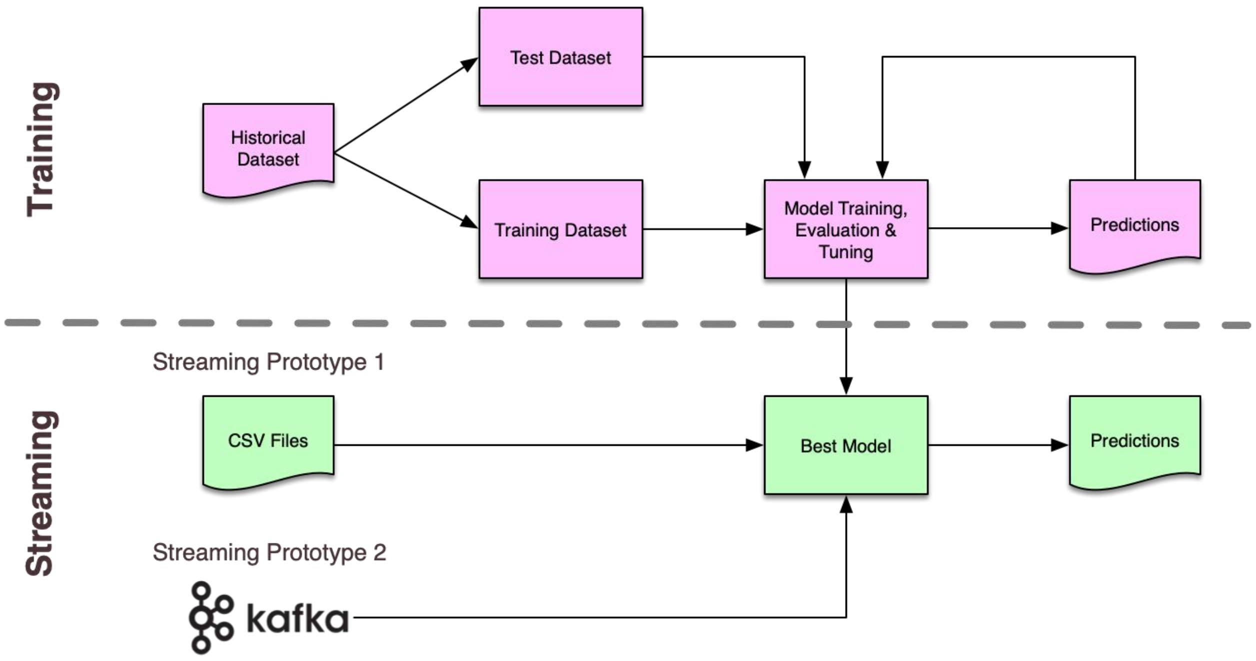
```
+-------+-------------+--------------+---------------+-----------+-----------+
|hour_24|     category|rounded_amount|non_fraud_count|fraud_count|fraud_ratio|
+-------+-------------+--------------+---------------+-----------+-----------+
|     03|  grocery_pos|         400.0|              0|         42|      100.0|
|     22|  grocery_pos|         300.0|              0|        129|      100.0|
|     00|  grocery_pos|         400.0|              0|         23|      100.0|
|     23|  grocery_pos|         300.0|              0|        142|      100.0|
|     02|  grocery_pos|         400.0|              0|         36|      100.0|
|     01|  grocery_pos|         400.0|              0|         30|      100.0|
|     23|     misc_net|         800.0|              1|        134|      99.26|
|     22|     misc_net|         700.0|              1|         77|      98.72|
|     22|     misc_net|         800.0|              2|        137|      98.56|
|     22| shopping_net|        1000.0|              6|        258|      97.73|
|     23|     misc_net|         900.0|              2|         78|       97.5|
|     23| shopping_net|        1000.0|              6|        208|       97.2|
|     22|     misc_net|         900.0|              2|         52|       96.3|
|     01| shopping_net|        1000.0|              1|         24|       96.0|
|     22| shopping_net|        1100.0|              7|        156|      95.71|
|     23|     misc_net|         700.0|              4|         85|      95.51|
|     23|entertainment|         600.0|              1|         21|      95.45|
|     01|  grocery_pos|         300.0|             14|        285|      95.32|
```

Last but not least, we can combine the above 2 rules together to alert transactions by categories, time and amount. As seen in the above graph, with specific combinations from our training dataset, we should be able to confidently identify fraudulent transactions.

Machine Learning Pipeline

Overall Design

We designed and tested a functional end-to-end machine learning pipeline. After evaluation of different algorithms and tuning, the best model was retained and used to support the real-time prediction process. The first streaming prototype was based on CSV file inputs. The second prototype was based on Apache Kafka. Shown below is a high level overview of our training and real-time streamed event processing pipelines.



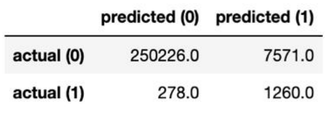
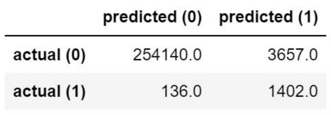
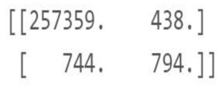
The ML pipeline was designed in such a way as to be able to ingest raw inputs and perform all necessary transformations as distinct stages in the pipeline. This was also a requirement to enable structured streaming capabilities. One major challenge in that area was to incorporate into the pipeline the various data transformations and extra features needed to establish spending patterns. The Spark MLlib package provided a number of common transformations (e.g. VectorAssembler, StringIndexer, OneHotEncoder, etc..). However, to achieve our goal, we had to define our own transformers.

As an additional preprocessing step before training, we balanced the dataset by computing a weight column based on fraud ratio and then configuring the models to use that column to apply the appropriate weight for each record.

Model Training, Tuning and Evaluation

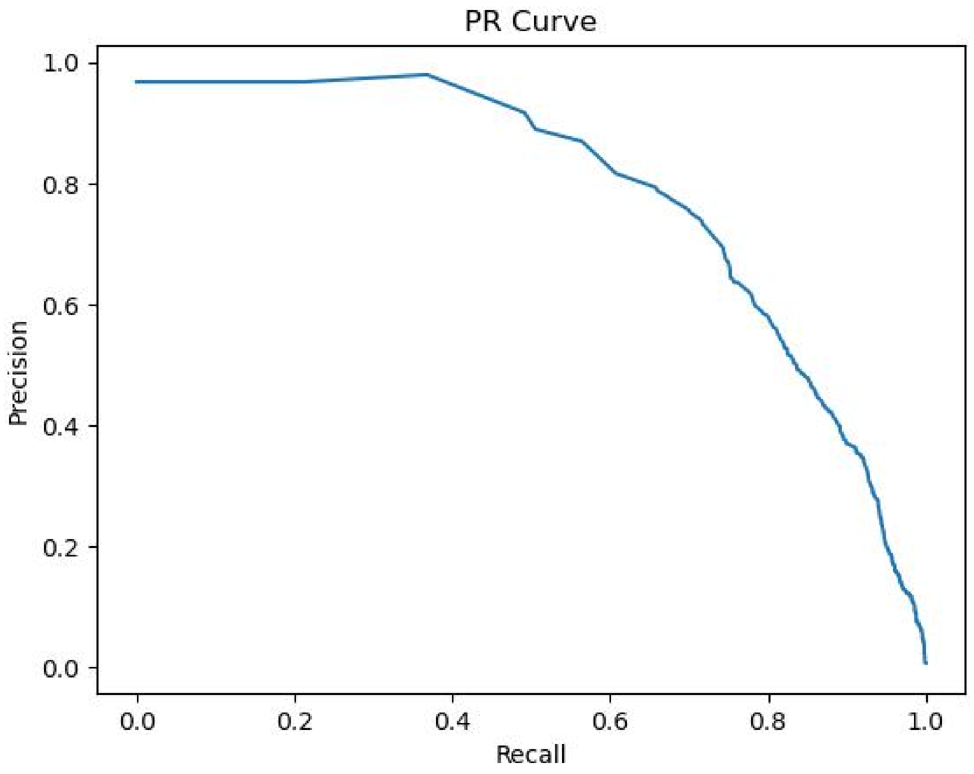
We trained and compared the performance of three models based on the RandomForestClassifier, GradientBoostingTreesClassifier and Feedforward Neural Network algorithms. As the dataset was quite large (1.2M records, 500MB storage), we ran crossvalidation and hyperparameter tuning outside the shared workspace on the most powerful machines the group had. We then integrated the best parameters in a single tuned model pipeline.

Confusion matrices and metrics for the three algorithms we evaluated are shown below.

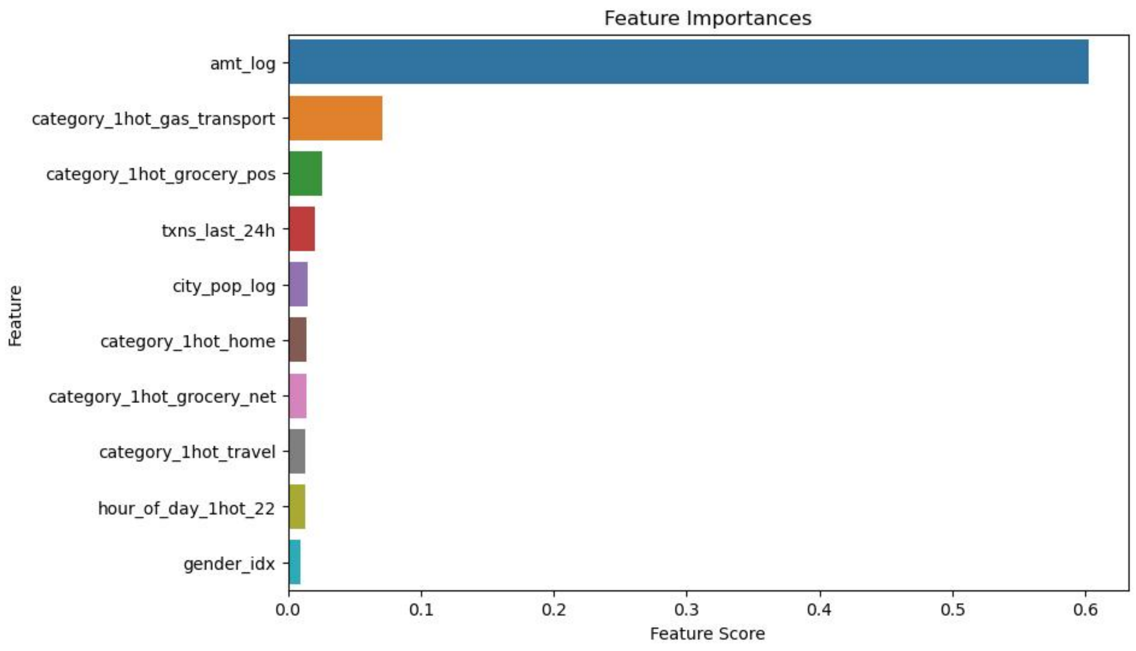
Evaluation metrics	Model #1: Random Forest	Model #2: Gradient Boosting Trees	Model #3: Feedforward Neural Network
areaUnderPR	0.525	0.770	0.419
Confusion matrix	 predicted (0) predicted (1) actual (0) 250226.0 7571.0 actual (1) 278.0 1260.0	 predicted (0) predicted (1) actual (0) 254140.0 3657.0 actual (1) 136.0 1402.0	 `[[257359.    438.]` ` [   744.    794.]]`
Precision	0.142	0.277	0.644
Recall	0.819	0.912	0.516
F1 Score	0.243	0.425	0.573

As we can see from the evaluation results above, the Gradient Boosting Trees model produced the best results in terms of our main evaluation metric, areaUnderPR. We were also able to match the neural network's F1 scores through adjustments to the prediction thresholds. As such, we selected this model for the real-time pipeline. When compared to random forest model, gradient boosting model improved in both precision and recall scores. The recall increased from 81.9% to 91.2%, meaning 91% of fraud records can be detected in the test set. The precision went up from 14.2% to 27.7%, meaning that false alarm rate dropped by more than 13 percentage points. While the precision of model #3, the neural network model, is much higher than the first two models, recall of the neural network model is only 0.516. This number is relatively low for a fraud detection scenario.

As the target class (is_fraud) is heavily imbalanced, we felt that Precision-Recall metrics would be more meaningful than the AUC-ROC analysis. The P-R curve we produced out of this exercise highlights the compromise that would have to be reached between precision and recall should a different probability threshold be selected.



Feature importances extracted from the tuned model were consistent with our EDA findings. Unusual transaction amounts are the strongest differentiators for fraud detection, followed by specific merchant categories (e.g. gas stations and grocery stores).



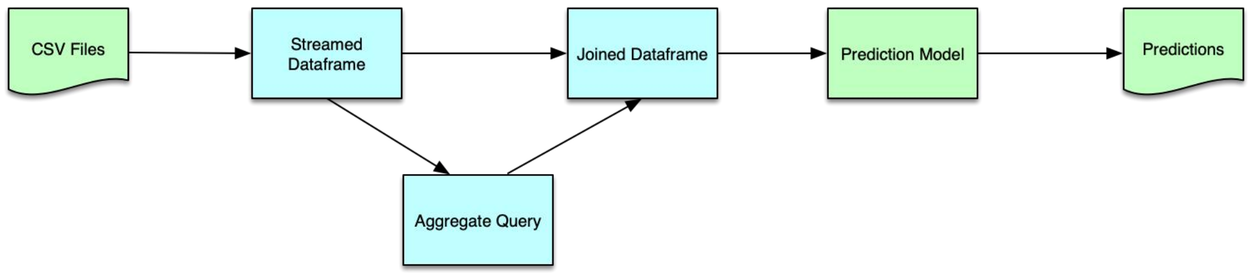
As a final step, we proceeded with finding the best balance between precision and recall by testing different probability thresholds. Given that the F1 metric takes into account precision and recall, we choose F1 as the evaluation metric. As the table below shows, a threshold of 90% yielded the best score. As a result, we programmed the best model to use that threshold for the real-time pipeline.

Probability Threshold	0.5	0.6	0.7	0.8	0.9
F1 Score	0.423	0.470	0.536	0.601	0.673

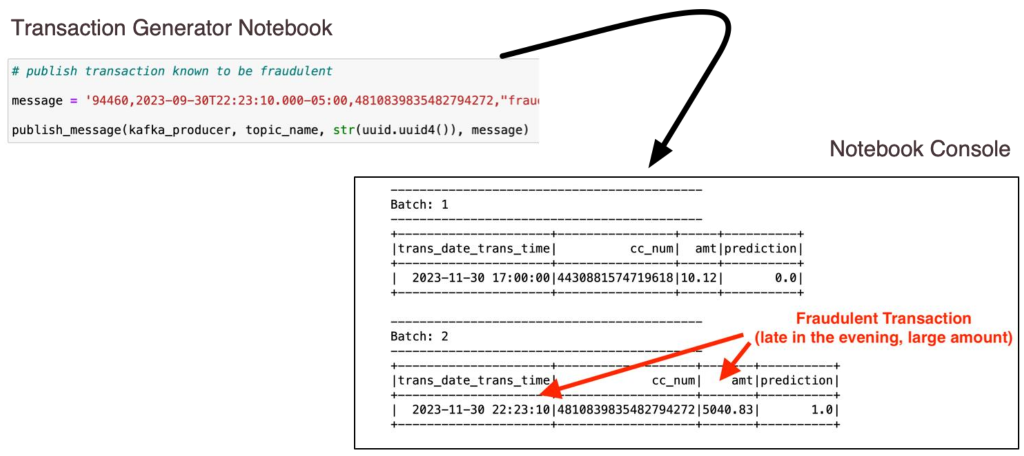
Real Time Prediction Pipeline

We implemented the real-time pipeline with the Spark Structured Streaming framework. Streaming dataframes work in the same way as static dataframes. However there are differences that must be taken into consideration. In a streaming context, data is ingested in small batches. This makes the use of aggregates in the prediction pipeline a little more complex.

For instance, we incorporated a feature containing the number of transactions accumulated over a 24-hour period. In a static dataframe, for training purposes, adding this feature was only a matter of running an aggregation query against the whole dataframe. In a streamed dataframe, where data comes in as micro-batches, a separate dataframe must be created to keep track of daily transaction counts. This “aggregate” dataframe can then be joined with the original streamed dataframe to add the new feature column. The diagram below illustrates the process.



Our first streaming prototype was constructed with CSV files as the format for the transaction events. This provided us with a quick way to generate transactions from the testing dataframe and test our real-time pipeline. Once we confirmed that the pipelined worked properly, we proceeded with Kafka as our streaming source. Setup of Zookeeper and Kafka was a fairly involved process. However, much to our relief, changing the streaming source was straightforward and only required changes to the configuration parameters of the streamed dataframe. The screenshot below shows the pipeline in action and our first real-time detection of a fraudulent transaction.


Transaction Generator Notebook
Notebook Console
Fraudulent Transaction
(late in the evening, large amount)

```
# publish transaction known to be fraudulent

message = '94460,2023-09-30T22:23:10.000-05:00,4810839835482794272,"frau

publish_message(kafka_producer, topic_name, str(uuid.uuid4()), message)
```

```
-------------------------------------------
Batch: 1
-------------------------------------------
+-------------------+----------------+-----+----------+
|trans_date_trans_time|          cc_num|  amt|prediction|
+-------------------+----------------+-----+----------+
|  2023-11-30 17:00:00|4430881574719618|10.12|       0.0|
+-------------------+----------------+-----+----------+

-------------------------------------------
Batch: 2
-------------------------------------------
+-------------------+-------------------+-------+----------+
|trans_date_trans_time|             cc_num|    amt|prediction|
+-------------------+-------------------+-------+----------+
|  2023-11-30 22:23:10|4810839835482794272|5040.83|       1.0|
+-------------------+-------------------+-------+----------+
```

3. Conclusions

This project proved to be as challenging as it was rewarding. From a data analysis and fraud detection standpoint, our main conclusions are as follows:

- Our analysis suggests that most fraudulent transactions involve larger than usual amounts and occur late in the evening.
- Credit card fraud detection is very much a matter of compromise. As we observed through numerous runs involving different parameters, maximizing fraudulent transaction detection can lead to a higher rate of false positives, which would translate into calls from annoyed customers. On the other hand, increasing the probability threshold will lead to a higher count of false negatives, resulting in increased fraud handling costs for the credit agency. We tuned our model to strike the best balance between the two. However a credit card agency may opt to choose a different strategy.

Regarding the technical ML pipeline aspects, our main takeaways are as follows:

- Out of the classification algorithms we evaluated, GradientBoostClassification yielded the best performance (using the AUC-PR metric) for this particular use case.
- In spite of similarities between static and streamed dataframe APIs, an ML pipeline must be adjusted to the micro-batch approach used for all operations. This was particularly important for aggregate features. Enrichment of the streamed dataframes with aggregates must be done by creating separate aggregate queries, which can then be joined with the main inbound dataframe.