CMPE - 255

Junjie Zhu

Project Document

Project Source Code Link:

Final Project Folder [Google Link]

Project Track:

Application

Application Field:

Serverless Real-Time Credit Card Fraud Detection [Reference Google Blog Link]

Key Techniques:

Technology Wise:

- Colab, TensorFlow, BigQuery, AI Platform, Cloud Firestore, Pub/Sub, Dataflow,

Model Wise:

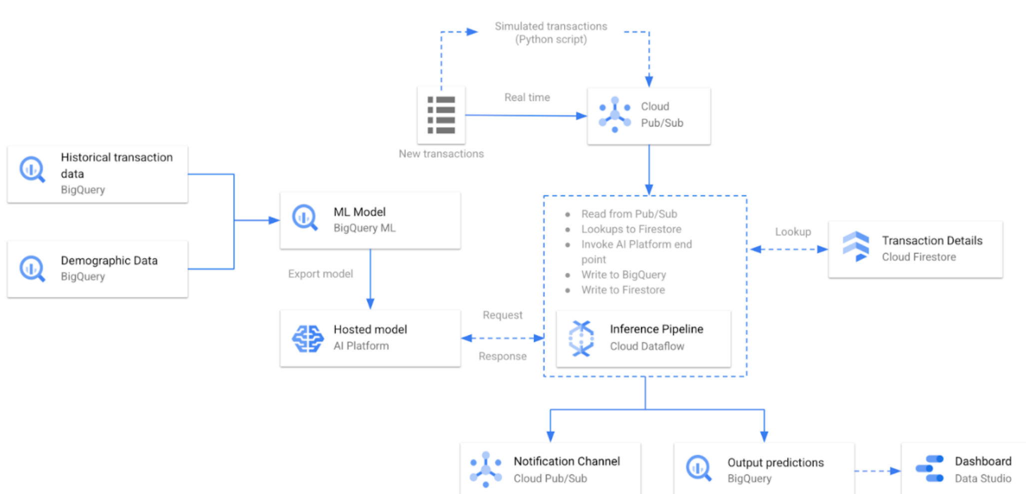
- Boost Tree Model, Deep Neural Network Model, Random Forest Model, Gradient Boost Tree Model, Cart Tree Model

Concept Wise:

- Machine Learning, Data Mining, Feature Engineering, Cloud Computing, Serverless Computing, Stream Processing, Batch Processing

Overall Architecture:

Workflow Architecture Diagram:



Application Objective:

This project aims to predict whether a credit card transaction is fraudulent or not in real time.

After the models are trained and deployed, we will use streaming dataflow pipeline to

- consume incoming transaction details from Cloud Pub/Sub
- does data preprocessing (by calling Firestore for data enrichment using transaction history)
- invokes multiple ML models deployed on AI Platform
- stores the prediction results to BigQuery and
- sends notification to another Pub/Sub topic when a transaction is predicted fraudulent for downstream consumptions

Utilized Dataset:

Fake Credit Card Transaction Data Generator [GitHub Repo Link]

Dataset Details:

BigQuery Tables (for BigQuery model Usage):

- train_raw - Data used for ML model training
- test_raw - Data used for ML model evaluation
- simulation_data - Data to be used for real-time inferences
- train_with_standard - BQ view providing standard features for training model
- train_with_aggregates - BQ view providing aggregates features for training model
- test_with_standard - BQ view providing standard test data for evaluating model
- test_with_aggregates - BQ view providing aggregates test data for evaluating model
- demographics - Data comprising customer demographics like name, gender, address

CSV Tables (for TensorFlow model Usage):

- train_with_aggregates.csv - Same as above
- test_with_aggregates.csv - Same as above

About Models:

For this pattern, we opted for the XGBoost (boost tree) model which worked really well while still retaining some level of model explainability. We initially used the boosted tree classifier in BigQuery ML by using standard SQL to train the model and arrive at the probability score for each transaction. Due to the imbalanced nature of the dataset, we used the F1 score and AUC to evaluate the performance of the model.

After the initial evaluation, to boost the performance of the model we derived additional features from the dataset focusing on the frequency of the transactions and the average transaction amount over a period of time.

As part of the solution, we have used predictions from both models as part of the pipeline. The model using the standard features gives relatively faster results, the other model uses the features derived from looking at historical data to make the predictions, so it is relatively slow.

For performance comparison purposes, we trained other models such as BigQuery DNN model and the TensorFlow Random Forest model, we will illustrate the performance in the model performance section.

Model Differences:

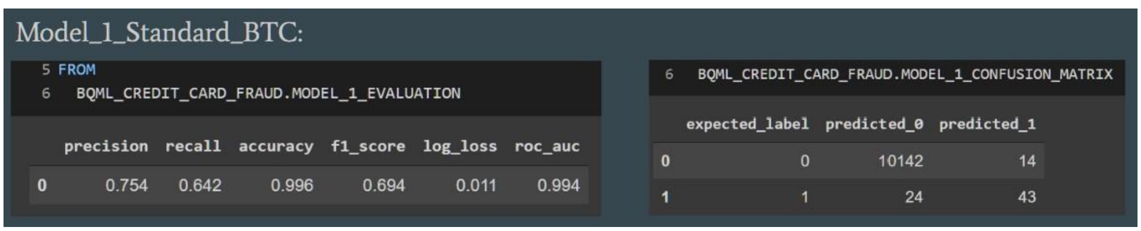
Modesl with standard features: It uses the features which are present in the dataset and don't rely on any feature generation techniques.

Models with aggregate features: Along with the provided features, It uses feature generation techniques to compute transaction frequency, average spend etc. For a given credit card:

- trans_freq_24 - Number of transactions in the last 24 hours
- trans_diff - Time difference between current transaction and last transaction in seconds
- avg_spend_pw - Average transaction amount in the past 1 week
- avg_spend_pm - Average transaction amount in the past 1 month

Model Performance:

1 – Standard Boost Tree Model:

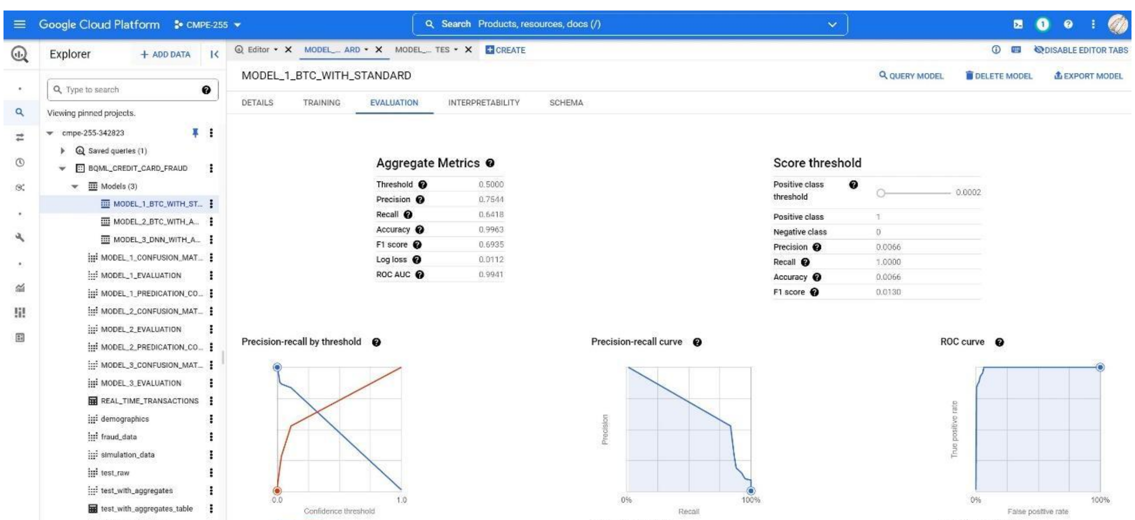

Model_1_Standard_BTC:

```
5 FROM
6   BQML_CREDIT_CARD_FRAUD.MODEL_1_EVALUATION
```

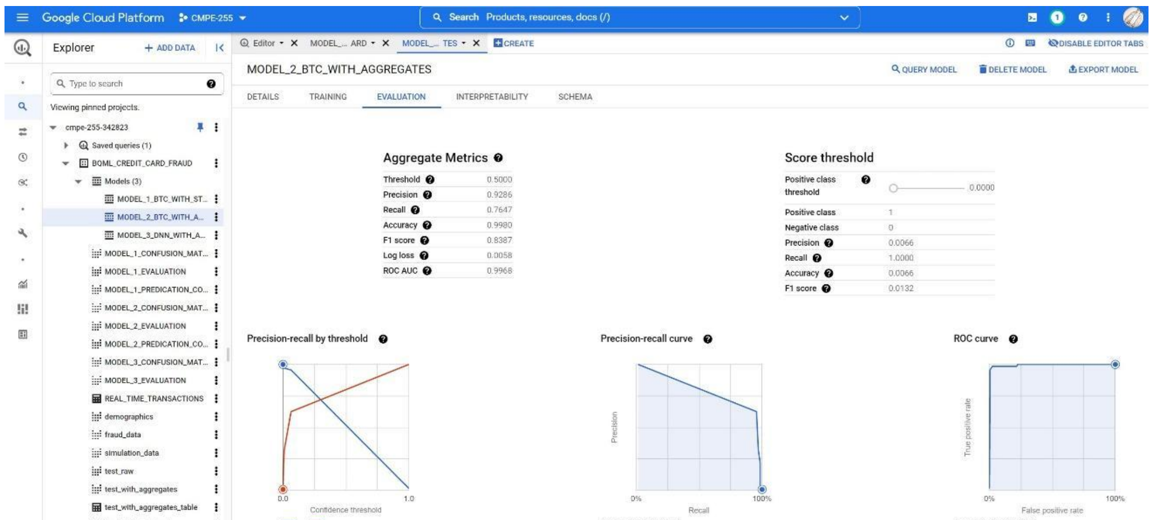
| | precision | recall | accuracy | f1_score | log_loss | roc_auc |
|---|---|---|---|---|---|---|
| 0 | 0.754 | 0.642 | 0.996 | 0.694 | 0.011 | 0.994 |

```
6   BQML_CREDIT_CARD_FRAUD.MODEL_1_CONFUSION_MATRIX
```

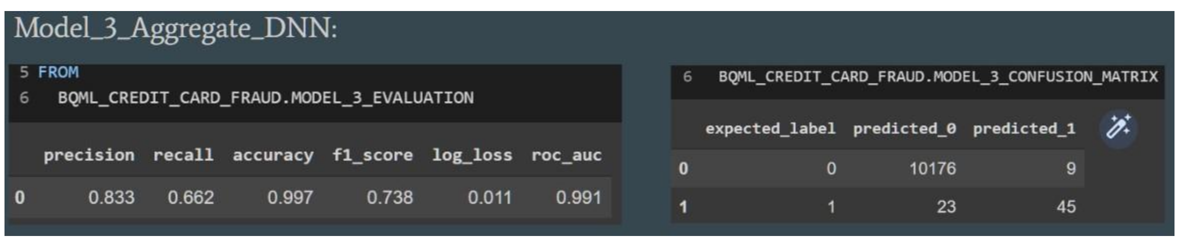
| | expected_label | predicted_0 | predicted_1 |
|---|---|---|---|
| 0 | 0 | 10142 | 14 |
| 1 | 1 | 24 | 43 |

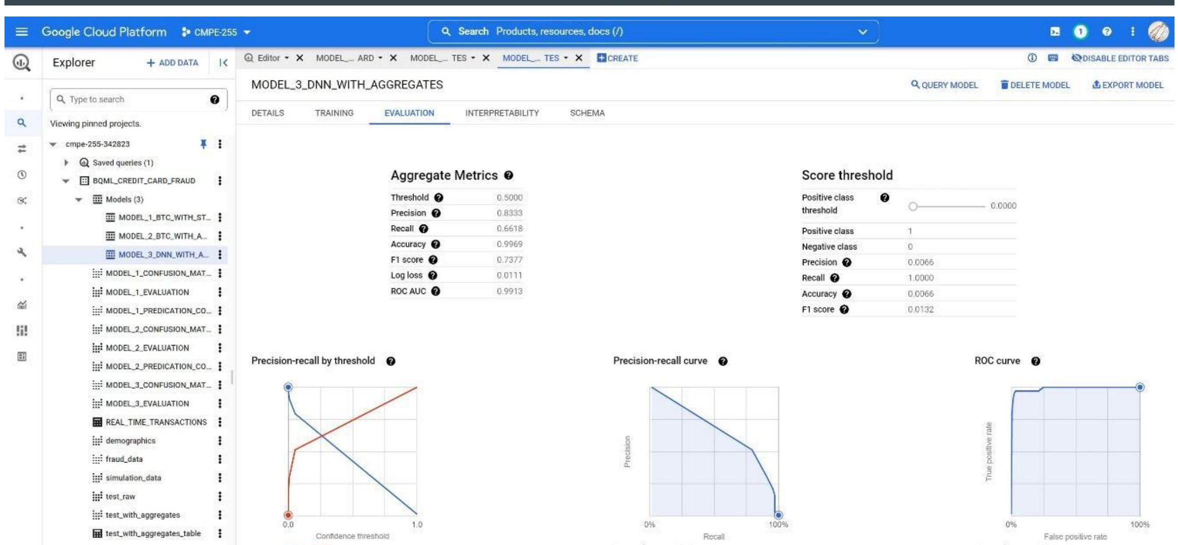


2 – Aggregate XGBoost Model:

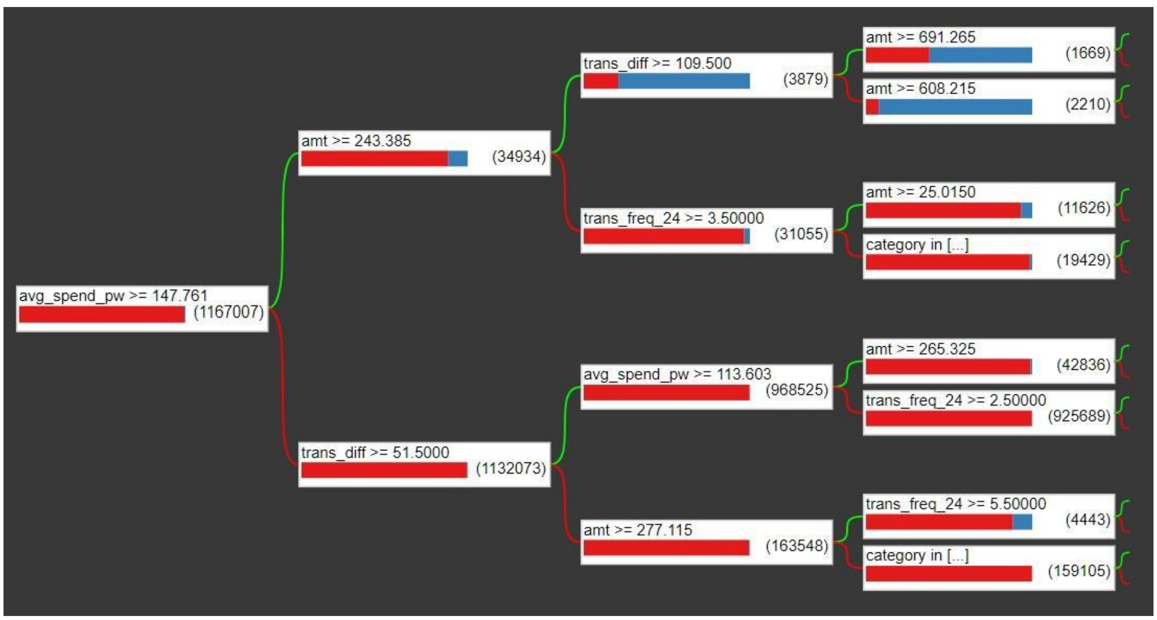

3 – Aggregate DNN Model:

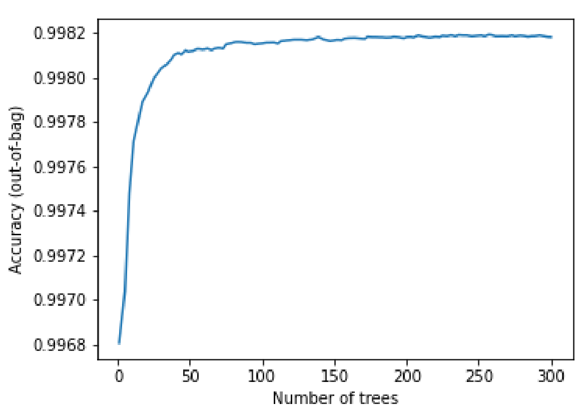


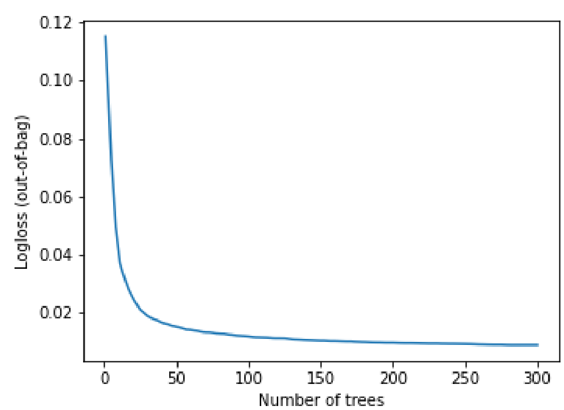



4 – Aggregate Random Forest Model:

```
loss: 0.0000
tp: 1390.0000
fp: 78.0000
tn: 497945.0000
fn: 734.0000
accuracy: 0.9984
precision: 0.9469
recall: 0.6544
auc: 0.9801
prc: 0.8551
f1: 0.7739
```

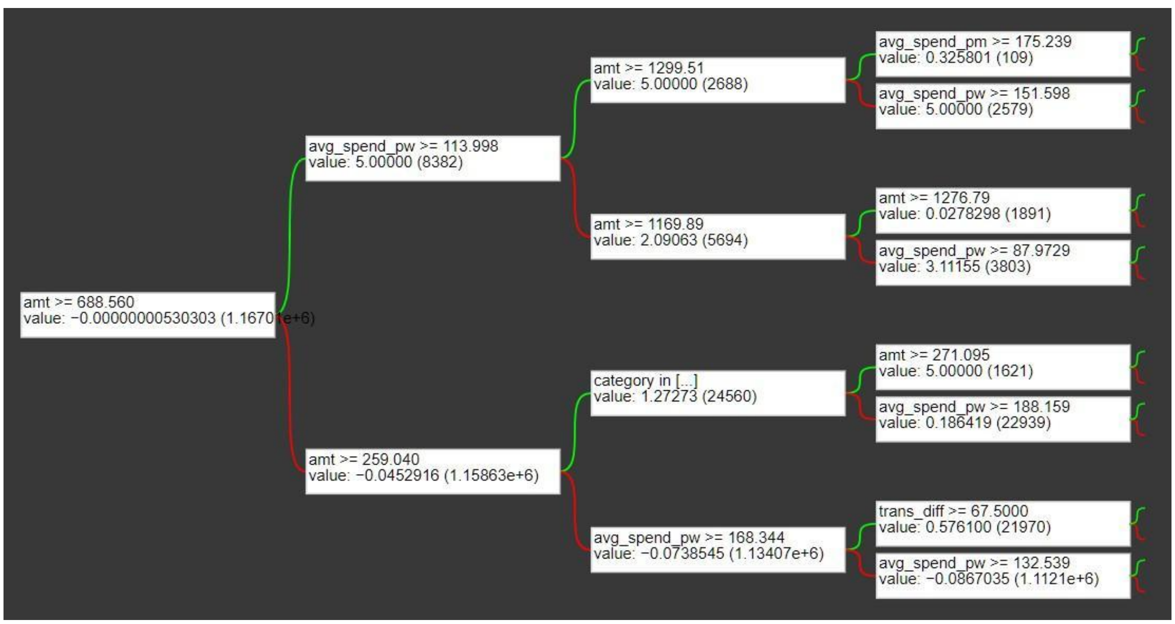


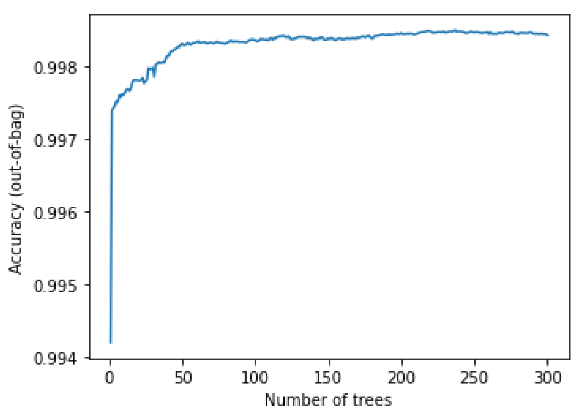



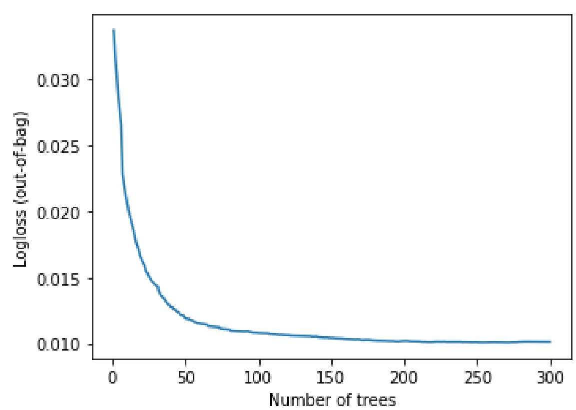



5 – Gradient Boosted Trees Model:

```
loss: 0.0000
tp: 1569.0000
fp: 249.0000
tn: 497774.0000
fn: 555.0000
accuracy: 0.9984
precision: 0.8630
recall: 0.7387
auc: 0.9749
prc: 0.8407
f1: 0.7960
```

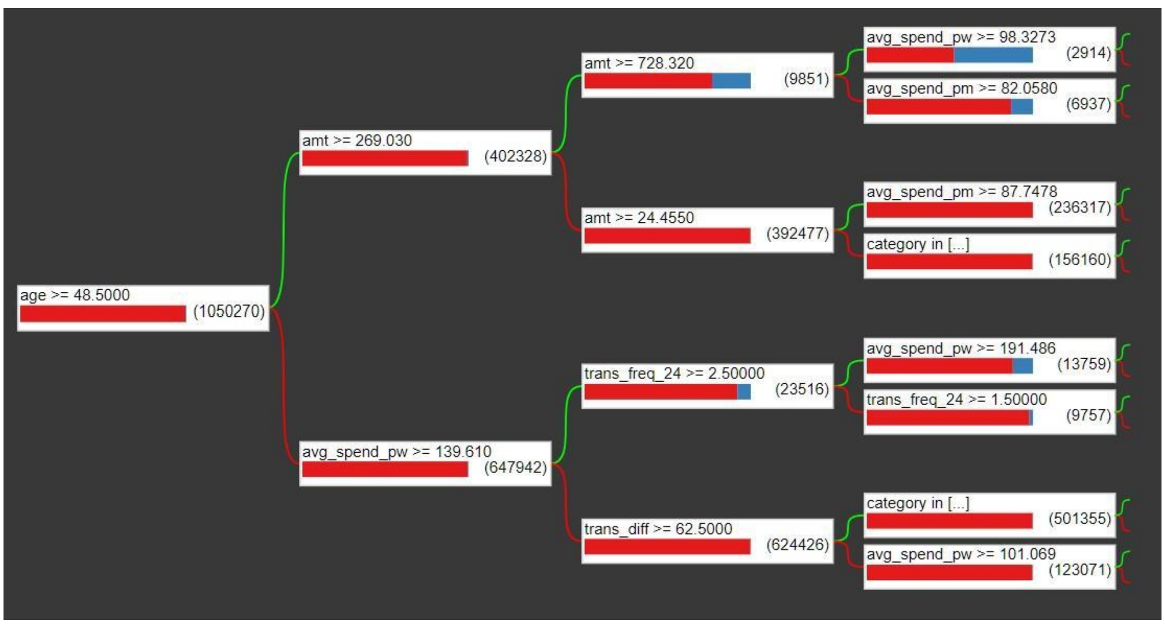


6 – Cart Tree Model:

```
loss: 0.0000
tp: 1321.0000
fp: 282.0000
tn: 497741.0000
fn: 803.0000
accuracy: 0.9978
precision: 0.8241
recall: 0.6219
auc: 0.9178
prc: 0.6820
f1: 0.7089
```



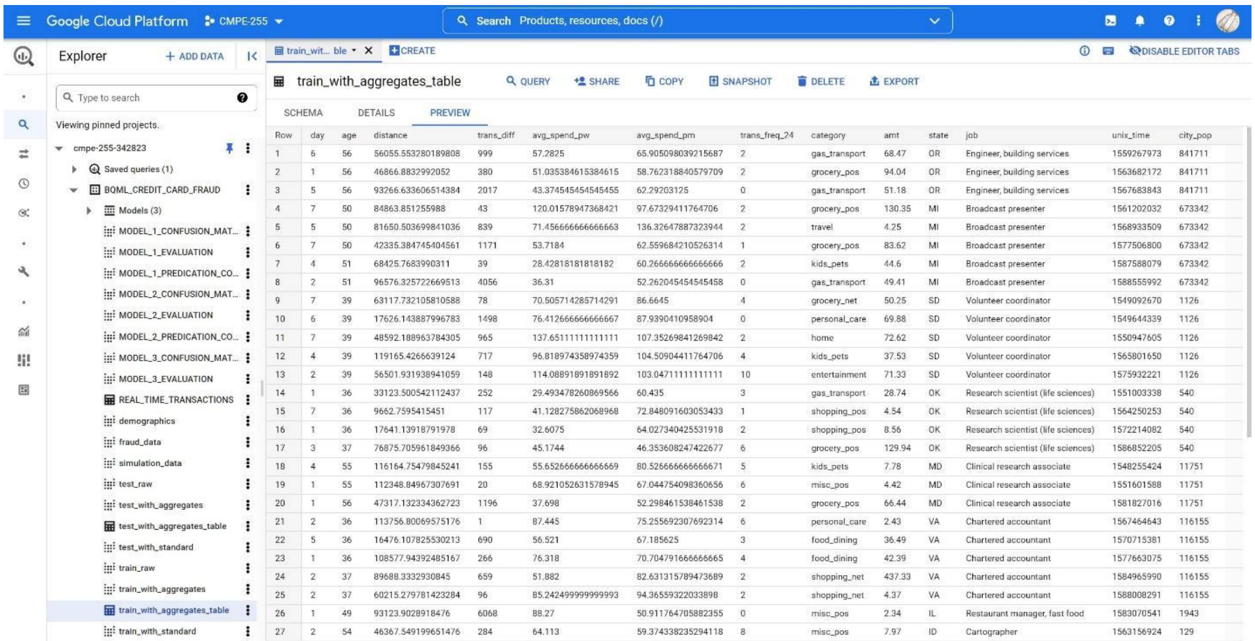

Conclusion:

- We want to capture fraud transactions, so we aim for high TP, and low FN, high recall, and high F1, so overall performance in descending order:
  - Model 2 > Model 5 > Model 4 > Model 3 > Model 6 > Model 1
- All model works well on identifying the normal transaction, and works well on identifying the real fraud transactions (high precision) but works relatively poor on identifying all fraud transactions from all transactions (low recall)

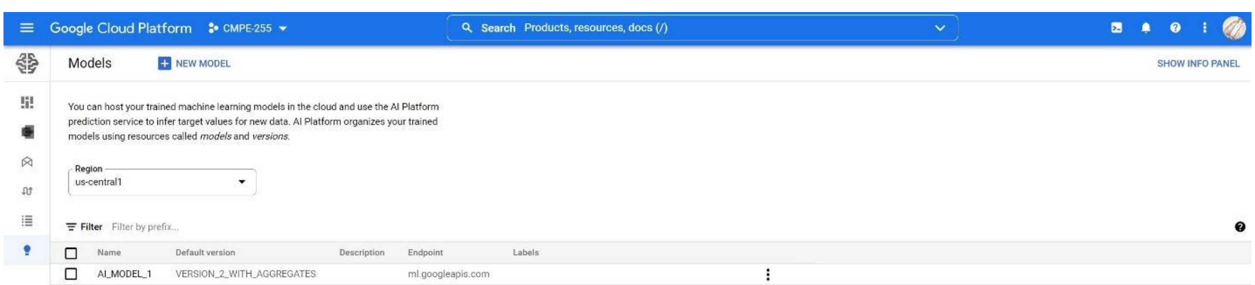
- Compared to the standard features, the aggregates features successfully boost the overall model performance, especially precision and recall
- Without further hyperparameter tunning, the Boost Tree Model works the best for the training dataset (where data is unbalanced)

Workflow Screening:

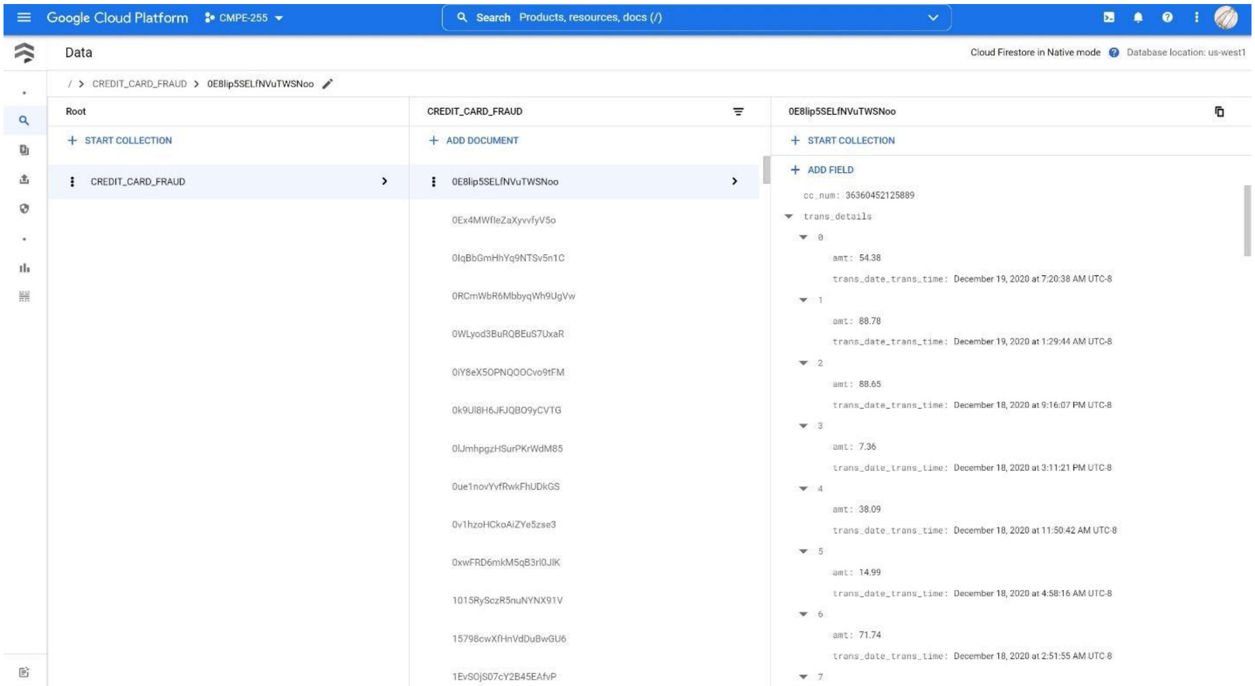
BigQuery



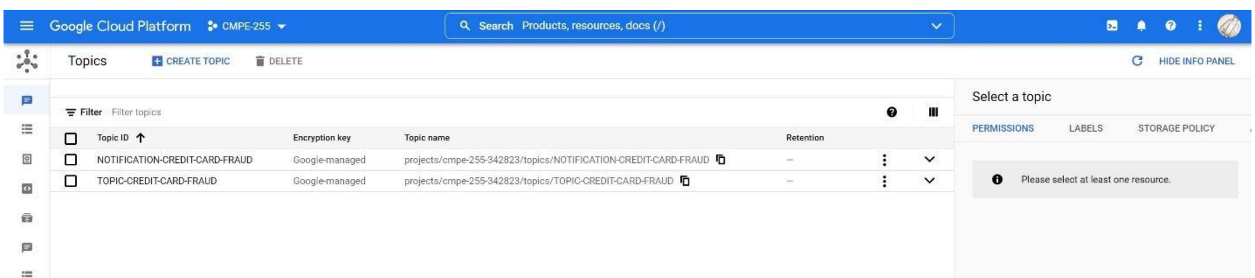

AI Platform

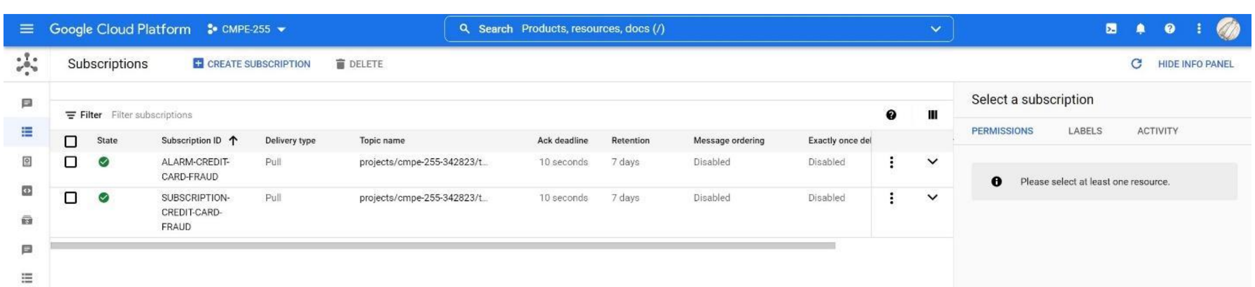


## Firestore

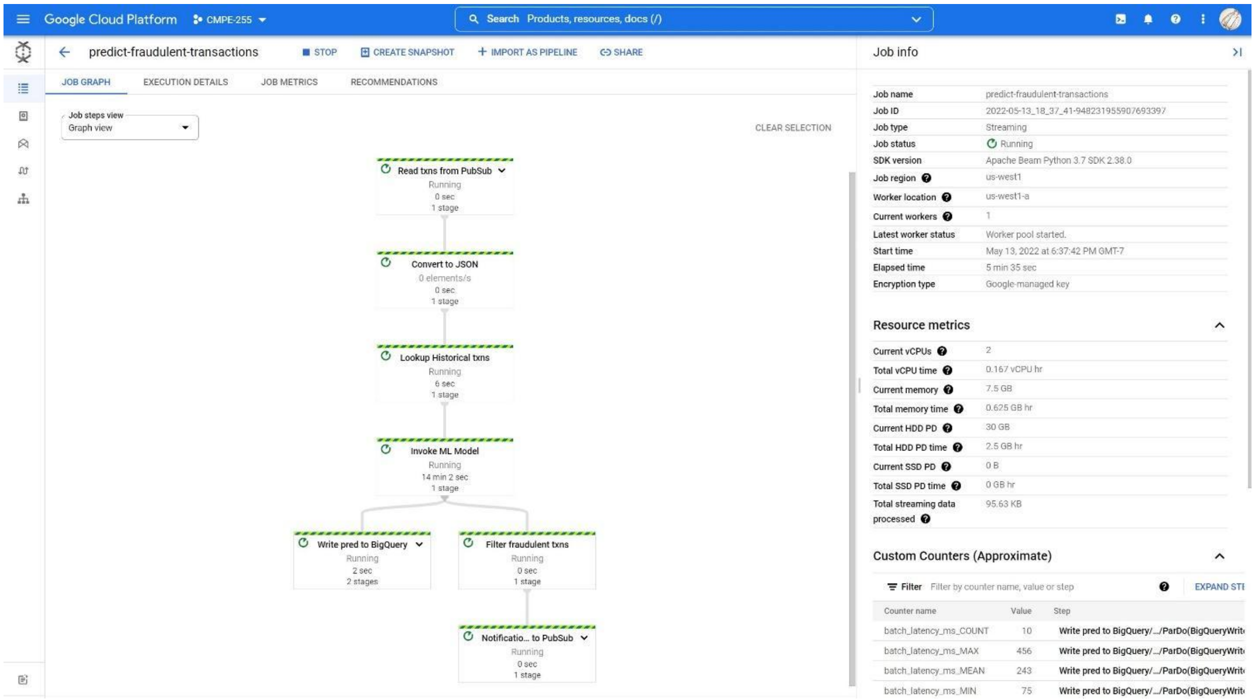



## Pub/Sub



DataFlow

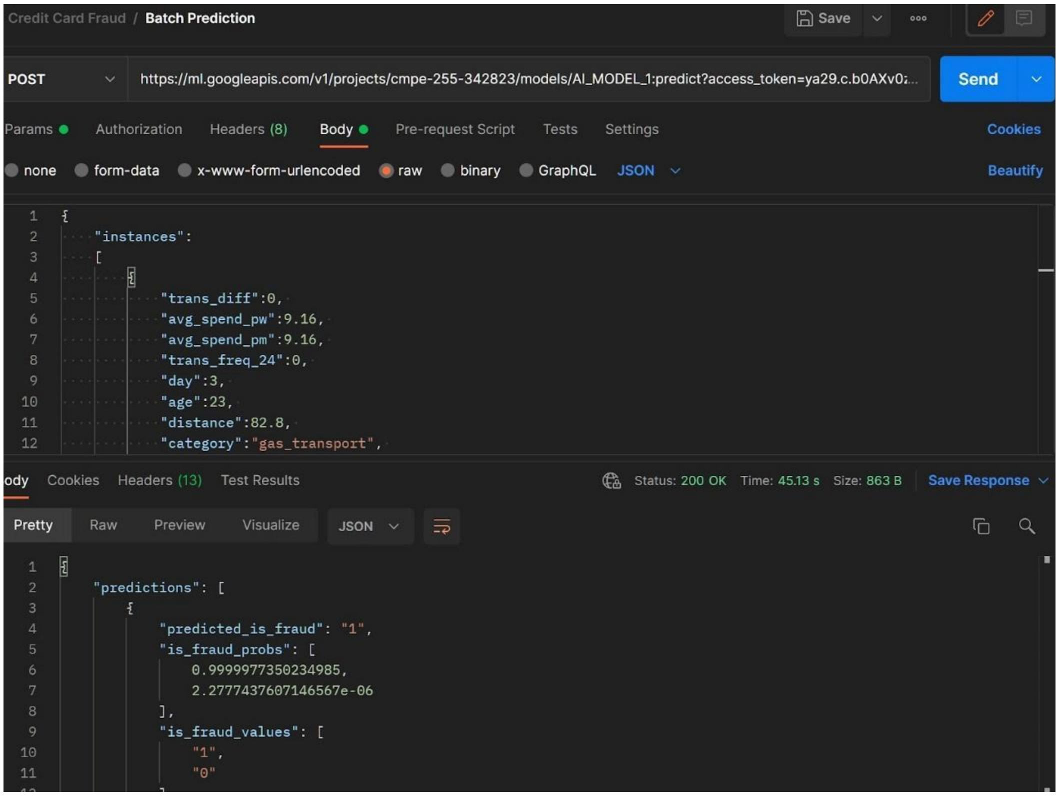



Batch Processing

Via Terminal RPC:

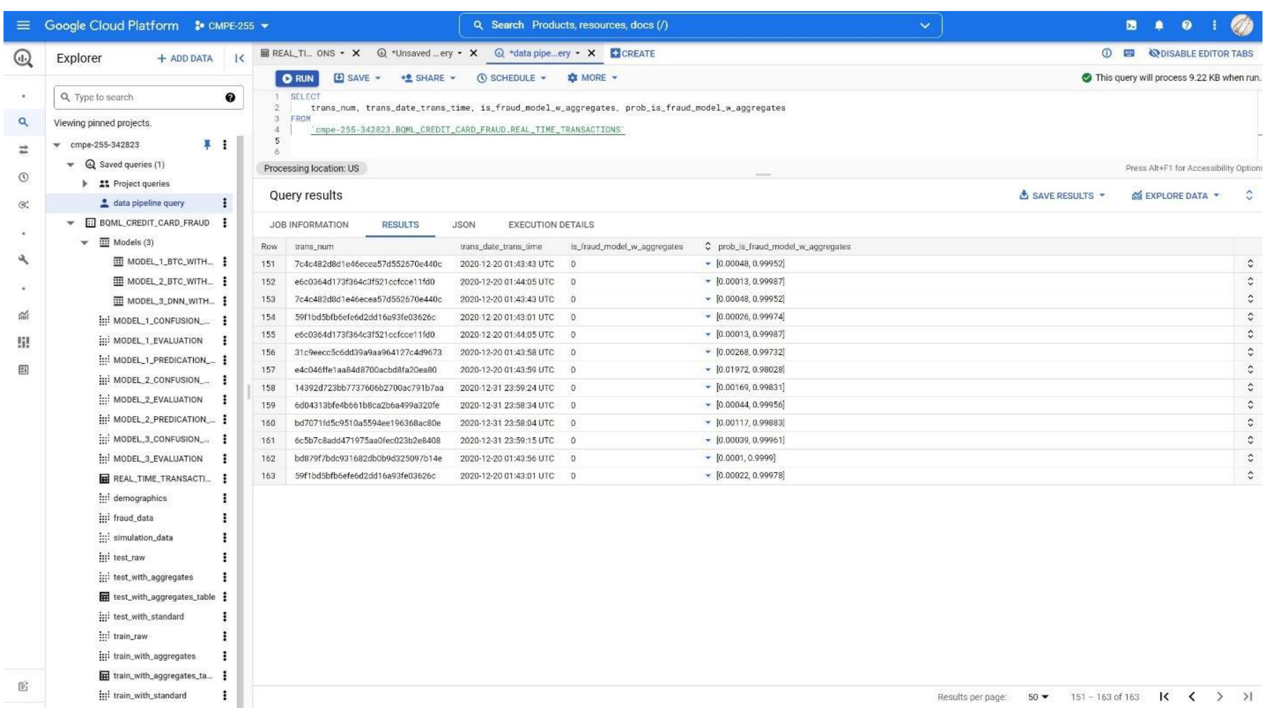
```
[ ]  1 !gcloud ai-platform predict --model $AI_MODEL_NAME \
     2 --version $VERSION_NAME_WITH_STANDARD \
     3 --region global \
     4 --json-instances sample_inputs/input_w_standard.json

    Using endpoint [https://ml.googleapis.com/]
    IS_FRAUD_PROBS                              IS_FRAUD_VALUES  PREDICTED_IS_FRAUD
    [0.9293921589851379, 0.07060789316892624]  ['1', '0']       1

[ ]  1 !gcloud ai-platform predict --model $AI_MODEL_NAME \
     2 --version $VERSION_NAME_WITH_AGGREGATES \
     3 --region global \
     4 --json-instances sample_inputs/input_w_aggregates.json

    Using endpoint [https://ml.googleapis.com/]
    IS_FRAUD_PROBS                                IS_FRAUD_VALUES  PREDICTED_IS_FRAUD
    [0.9999977350234985, 2.2777437607146567e-06]  ['1', '0']       1
```

Via Postman Rest API:



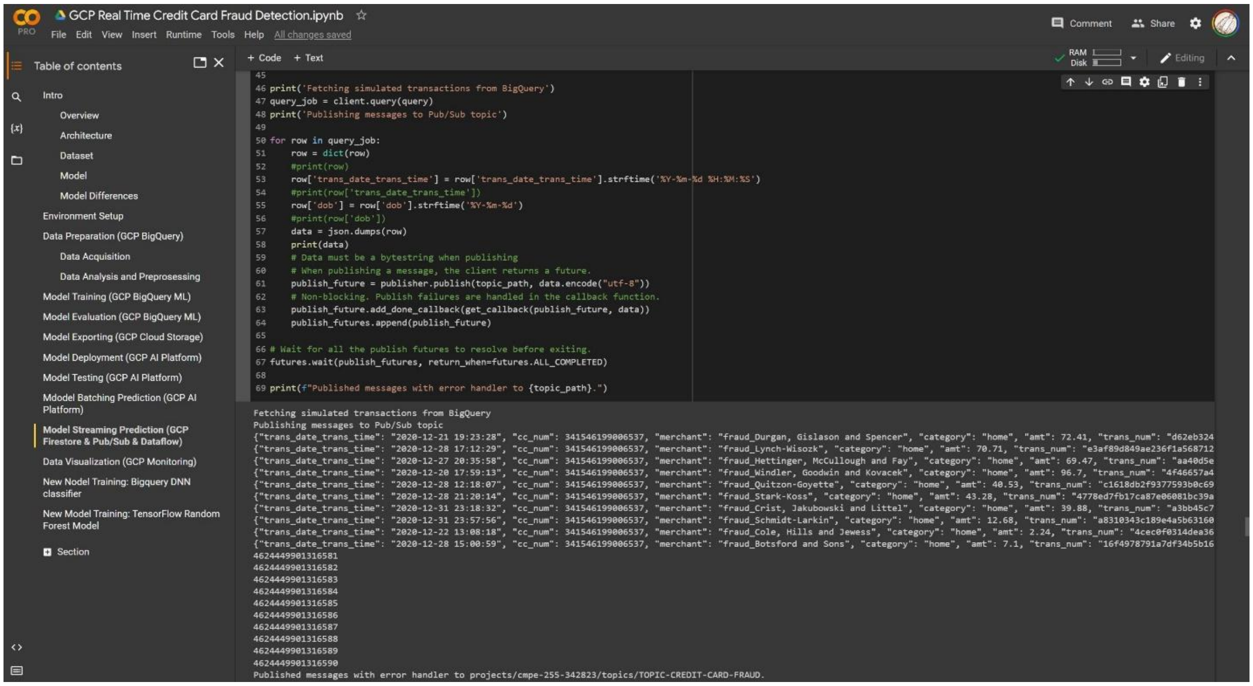

Stream Processing

Real-time transaction table (Before sending transactions to the publisher)

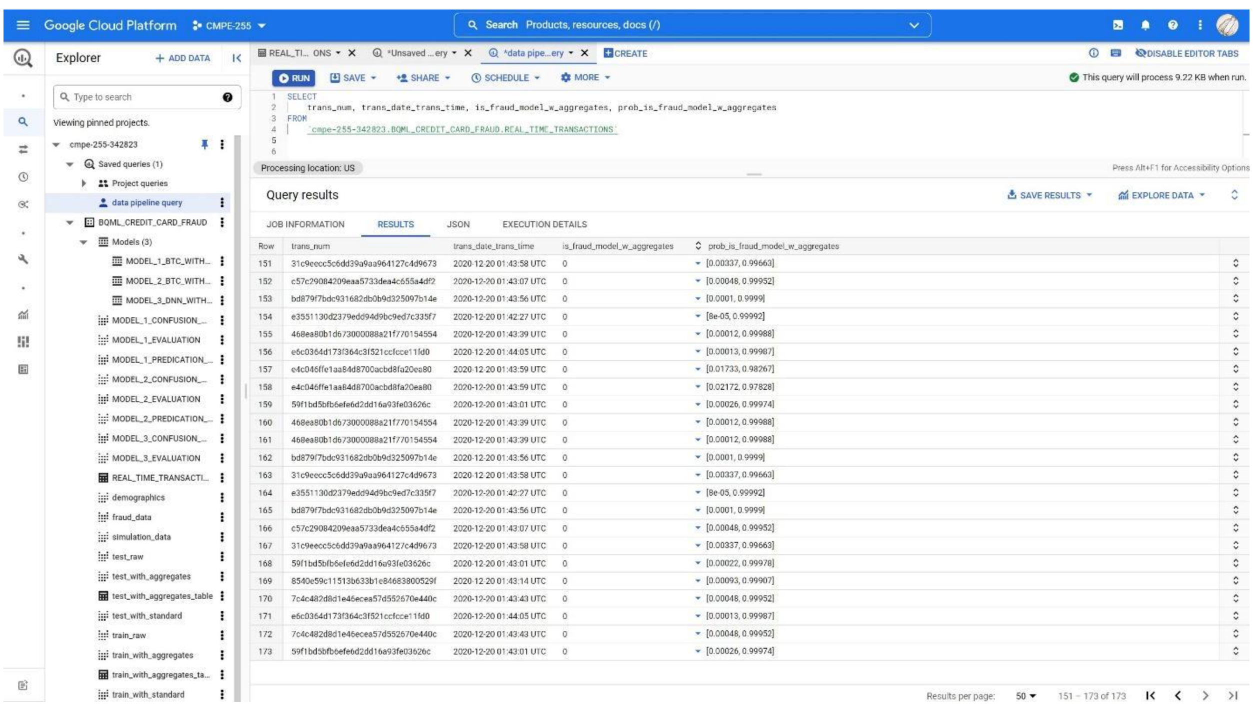



| Row | trans_num | trans_date_trans_time | is_fraud_model_w_aggregates | prob_is_fraud_model_w_aggregates |
|---|---|---|---|---|
| 151 | 7c4c482d8d1e46ecea57d552670e440c | 2020-12-20 01:43:43 UTC | 0 | [0.00048, 0.99952] |
| 152 | e6c0364d173f364c3f521ccfcce11fd0 | 2020-12-20 01:44:05 UTC | 0 | [0.00013, 0.99987] |
| 153 | 7c4c482d8d1e46ecea57d552670e440c | 2020-12-20 01:43:43 UTC | 0 | [0.00048, 0.99952] |
| 154 | 59f1bd5bfb6efe6d2dd16a93fe03626c | 2020-12-20 01:43:01 UTC | 0 | [0.00026, 0.99974] |
| 155 | e6c0364d173f364c3f521ccfcce11fd0 | 2020-12-20 01:44:05 UTC | 0 | [0.00013, 0.99987] |
| 156 | 31c9eecc5c6dd39a9aa964127c4d9673 | 2020-12-20 01:43:58 UTC | 0 | [0.00268, 0.99732] |
| 157 | e4c046ffe1aa84d8700acbd8fa20ea80 | 2020-12-20 01:43:59 UTC | 0 | [0.01972, 0.98028] |
| 158 | 14392d723bb7737606b2700ac791b7aa | 2020-12-31 23:59:24 UTC | 0 | [0.00169, 0.99831] |
| 159 | 6d04313bfe4b661b8ca2b6a499a320fe | 2020-12-31 23:58:34 UTC | 0 | [0.00044, 0.99956] |
| 160 | bd7071fd5c9510a5594ee196368ac80e | 2020-12-31 23:58:04 UTC | 0 | [0.00117, 0.99883] |
| 161 | 6c5b7c8add471975aa0fec023b2e8408 | 2020-12-31 23:59:15 UTC | 0 | [0.00039, 0.99961] |
| 162 | bd879f7bdc931682db0b9d325097b14e | 2020-12-20 01:43:56 UTC | 0 | [0.0001, 0.9999] |
| 163 | 59f1bd5bfb6efe6d2dd16a93fe03626c | 2020-12-20 01:43:01 UTC | 0 | [0.00022, 0.99978] |

Sending transactions to the publisher (10 transactions per call…to save budget!)




Real-time transaction table (After sending transactions to the publisher)



References:


How to build a fraud detection solution| Google Cloud

Real-Time Credit Card Fraud Detection | Gitlab

Credit Card Transaction Data Generator

BigQuery documentation | Google Cloud

BigQuery ML documentation | Google Cloud

Cloud Storage documentation | Google Cloud

Firestore documentation | Google Cloud

AI Platform documentation | Google Cloud

Pub/Sub documentation - Google Cloud

Dataflow documentation | Google Cloud

TensorFlow Decision Forests | TensorFlow