

# **Digital Bank Mobile App - A Descriptive Analysis of Transaction Events and a new suggestion of a fraud decision flow for transactions classification**

Hélio Assakura

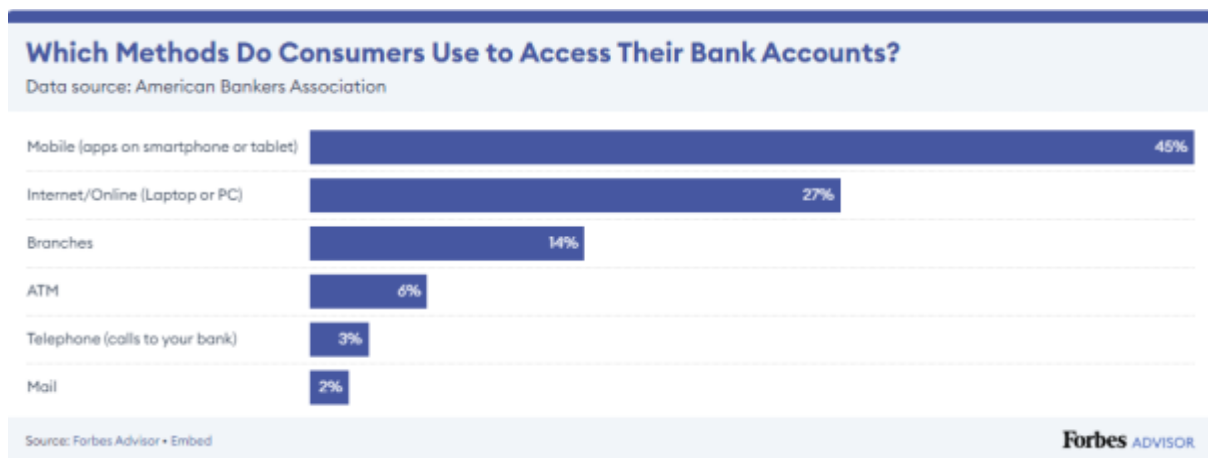
# Summary

<b>Summary.....</b>	<b>2</b>
<b>Introduction.....</b>	<b>4</b>
<b>Descriptive analysis of transaction events.....</b>	<b>6</b>
Which are the variables?.....	6
About the data.....	7
A few notes about it.....	7
How's the data distributed?.....	9
Attribute's deep dive.....	10
Transaction Value.....	10
Distance to frequent location.....	10
Device age days.....	11
Transaction time.....	12
Client decision and other variables.....	12
Client decision.....	12
is_emulator, has_fake_location, has_root_permission and app_is_tampered.....	13
How's the data quality? How can we fix the problems? Can I trust it?.....	14
Null Values.....	14
Better formats for columns.....	14
Odd data.....	14
Is there a strong correlation between the variables?.....	16
Are there other possible variables that we can create?.....	17
Devices per account.....	17
Accounts per device.....	17
Devices per account vs Accounts per device.....	18
<b>Fraud Risk Classification Rules.....</b>	<b>19</b>
Fraud data.....	19
Fraud vs Good transactions.....	19
New decision flow: Criteria and Thresholds.....	20
How we will evaluate the transactions.....	20
High risk.....	21
Medium risk.....	21
Low risk.....	21
In a nutshell.....	22
How did we get to these rules?.....	22
Distance to Frequent Location.....	22
Transaction Value.....	24
Device Age.....	26
Hour of the day.....	28
Flags.....	28
Accounts per Device.....	29
<b>Evaluation of Classification Rules.....</b>	<b>31</b>
Current vs New decision flow.....	31

Possible impacts.....	32
Positive.....	32
Negative.....	33
Next steps.....	33
<b>Conclusion.....</b>	<b>35</b>
<b>Appendix.....</b>	<b>36</b>
Github Repository.....	36
Code snippets.....	36
Descriptive analysis of transaction events.....	36
Fraud Risk Classification Rules.....	40
Evaluation of Classification Rules.....	49

# Introduction

Fraudulent activity is a big concern for all types of transactions in every business sector, specially in Digital banking, and having a robust, frictionless and assertive fraud detection system is a must. A [national survey conducted by the American Bankers Association](#) found that in 2022, **45% of bank customers used apps on smartphones or other mobile devices** as their top option for managing their bank accounts, while **27% used online banking from a traditional computer** (laptop or desktop), and according to the survey, **47% of consumers cited security concerns as the main reason for not using mobile banking services**. (Source: Deloitte)



In this report, we will detail a brief analysis for the Transactions data from Digital Bank Mobile App, identifying patterns, creating a new set of rules for Fraud Risk detection and comparing it with the current decision flow. It's divided in 4 sections:

1. **Descriptive Analysis of Transaction Events:** We will explore the Transactional data, with a brief explanation of variables characteristics and their correlations, data quality and its potential issues, key statistics, their implications and a few insights of user behavior and possible vulnerabilities.
2. **Fraud Risk Classification Rules:** Using the Descriptive Analysis from the previous section, we will create a set of new Fraud Rules, classifying the transactions into **Low, Medium and High Risk**, setting a clear criteria and threshold for each category and explaining how they will contribute to the fraud assessment.
3. **Evaluation of Classification Rules:** This section will compare both decision flows, where in the current one, all transactions go through a Two-Factor authentication (2FA) process. We will consider the impact on user experience and finance,

considering factors such as fraud rate, transaction approval rate, false positives/negatives, user friction, costs and revenue.

4. **Conclusion:** A summary of findings and problems, actionable insights and suggestions on future improvement.

At the end of the report, there will be an appendix with a **link to a Github repository**, where you can find the code that generated the analysis, the instruction to run it, and a few snippets of the code.

# Descriptive analysis of transaction events

## Which are the variables?

We received a “Transactions” dataset from the Mobile App, which contains the following variables and definitions:

- **transaction\_id**: Unique event identifier
- **transaction\_timestamp**: Timestamp of the event in milliseconds
- **transaction\_value**: Monetary value of the transaction in reais (R\$)
- **account\_id**: Identifier of the associated account
- **device\_id**: Identifier of the device used for the event
- **distance\_to\_frequent\_location**: Distance in meters from a frequent location associated with the account at the time of the event
- **device\_age\_days**: Number of days since the device was first associated with the account
- **is\_emulator**: Indicates if the device was identified as an emulator
- **has\_fake\_location**: Indicates if the device was generating false location information
- **has\_root\_permissions**: Indicates if the device had root privileges
- **app\_is\_tampered**: Indicates if the app used was tampered in any way
- **client\_decision**: Apps’ final evaluation of the transaction (approved or denied) after the two-factor authentication (2FA) process, i.e., the response of the 2FA process

Additionally, we have access to a Fraud dataset containing the list of **transaction\_ids that resulted in fraud**. An important note is that in the assessment from the company, it’s said that the transaction is classified as fraud or not later, but we don’t know how long it took. **We are assuming that this evaluation was made after the last event, i.e., we can’t use this information to make any decisions regarding when the transaction was classified as fraud, and can’t create rules such as device or accounts watchlists from previous fraudulent transactions.**

## About the data

The data was sent from the Digital Bank by CSV files. We ingested it to be able to manipulate. Here's an example:

	transaction_id	transaction_timestamp	account_id	device_id	distance_to_frequent_location	device_age_days
0	acb6c8c8-caed-4	1710104394142	632543950	1328197082	6.77	64
1	0e522fe9-f918-4	1711144118565	1905400654	1328303665	1939.71	27
2	90269c82-4b78-4	1710585811679	919867536	1328436135	0.62	479
3	27995f51-3ced-4	1710527459233	1151976282	133030375	7.21	332
4	5b32b66b-1877-4	1709843563536	583448195	133030375	3.38	324

	is_emulator	has_fake_location	has_root_permissions	app_is_tampered	transaction_value	client_decision
0	False	False	False	False	171.69	denied
1	False	False	False	False	6.74	denied
2	False	False	False	False	58.14	denied
3	False	False	False	False	79.61	approved
4	False	False	False	False	94.65	denied

To be able to perform a few operations and get statistics about the data, we will change True/False to 1/0, and for `client_decision`, approved will be considered as 1 and denied, 0.

Here's some basic statistics about the columns:

	distance_to_frequent_location	device_age_days	is_emulator	has_fake_location	has_root_permissions	app_is_tampered	transaction_value	client_decision
count	399014.00	399852.00	399821.00	391755.00	399821.00	399821.00	399852.00	399852.00
mean	20724.95	207.20	0.00	0.00	0.00	0.00	176.42	0.65
std	267485.48	150.95	0.01	0.02	0.03	0.01	294.91	0.48
min	0.00	0.00	0.00	0.00	0.00	0.00	1.68	0.00
25%	2.30	55.00	0.00	0.00	0.00	0.00	55.65	0.00
50%	6.39	226.00	0.00	0.00	0.00	0.00	103.23	1.00
75%	46.08	322.00	0.00	0.00	0.00	0.00	194.70	1.00
max	18600000.00	579.00	1.00	1.00	1.00	1.00	31579.98	1.00

## A few notes about it

With this statistics, we can have a few insights about the data:

1. **Null Values:** The number of records from Transactions is **399852**, but there are columns, such as `has_fake_location`, that have a lot of null values and it's very

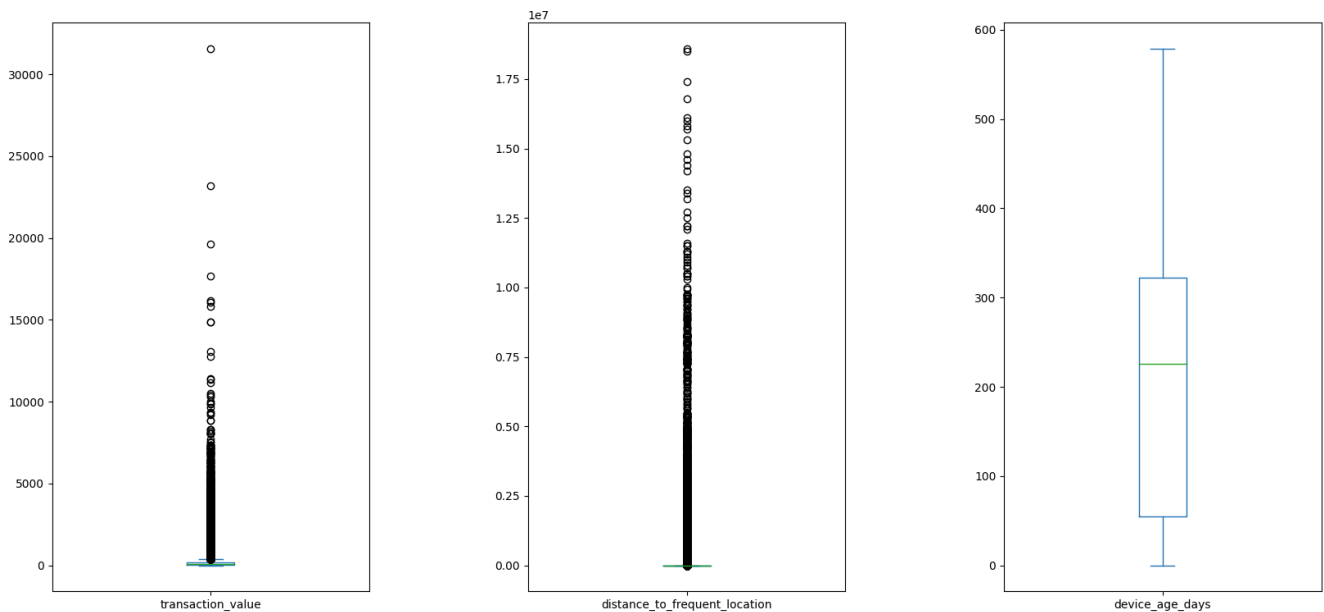
important to know how to handle them. **It can indicate problems with the sensors or even the code that generated this data.**

2. **Data ranges:** There's a big difference between the minimum and maximum in `distance_to_frequent_location` (from 0 to 18.600.000, or 18.600 kilometers) and `transaction_value` (from 1,68 to 31579,98). **This is a good indication of outliers or even odd behaviors from users.**
3. **Strong fraud indicators:** There are columns, such as `is_emulator`, `has_fake_location`, `has_root_permission` and `app_is_tampered` that if True, strongly indicates a fraud attempt. The data shows that there are events where they are flagged, so **it's a very good indicator to not allow these transactions to happen.**
4. **`transaction_timestamp` is not in a human readable format:** From the variables definition, it's the "Timestamp of the event in milliseconds". It needs to be formatted in a more user-friendly way, such as: (Year)-(Month)-(Day) (hours):(minutes):(seconds).(milliseconds). For example, 2024-03-10 20:59:54.142



## How's the data distributed?

To check the distribution and missing values from the Transactions dataset, we will explore a few variables: `transaction_value`, `distance_to_frequent_location`, `device_age_days`, `client_decision`, the **hour where the transaction was made** and a brief look into other columns. The box plot below show how's the data distributed and a few outliers for the first 3 numerical variables

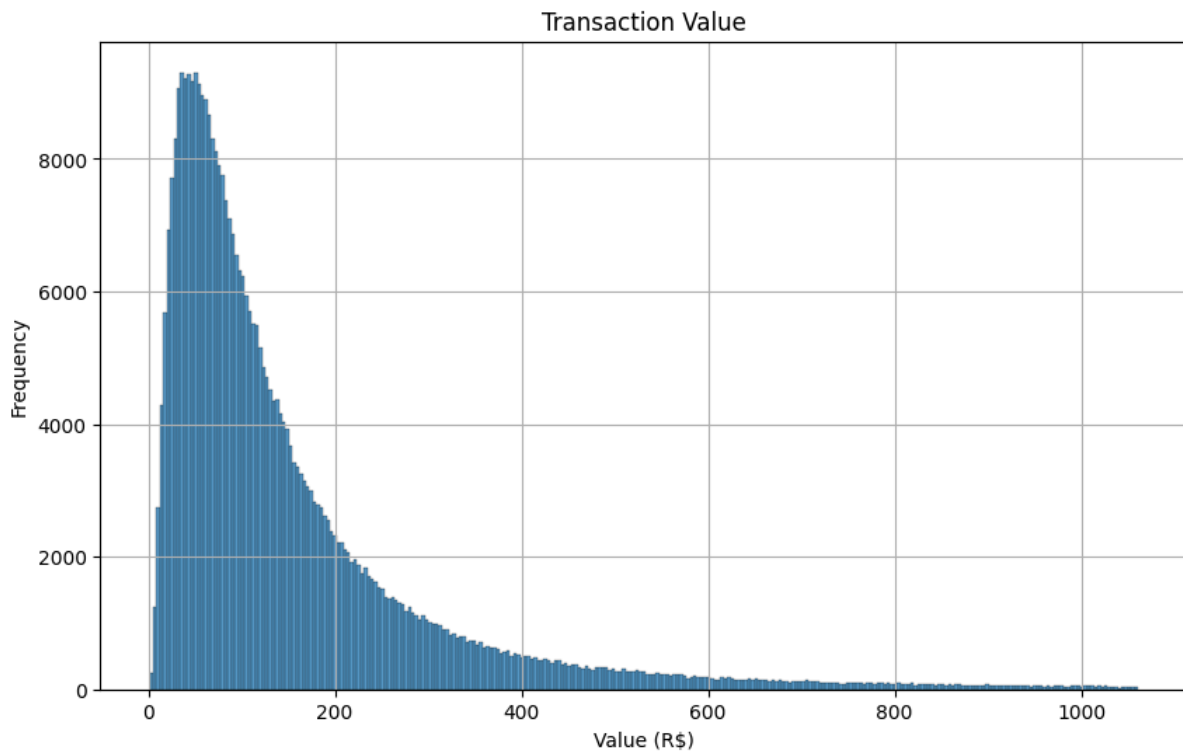


**And it shows that we expect that most of the data consists of low transaction values and they happen close to the frequent location. The device age does not vary a lot.**

## Attribute's deep dive

### Transaction Value

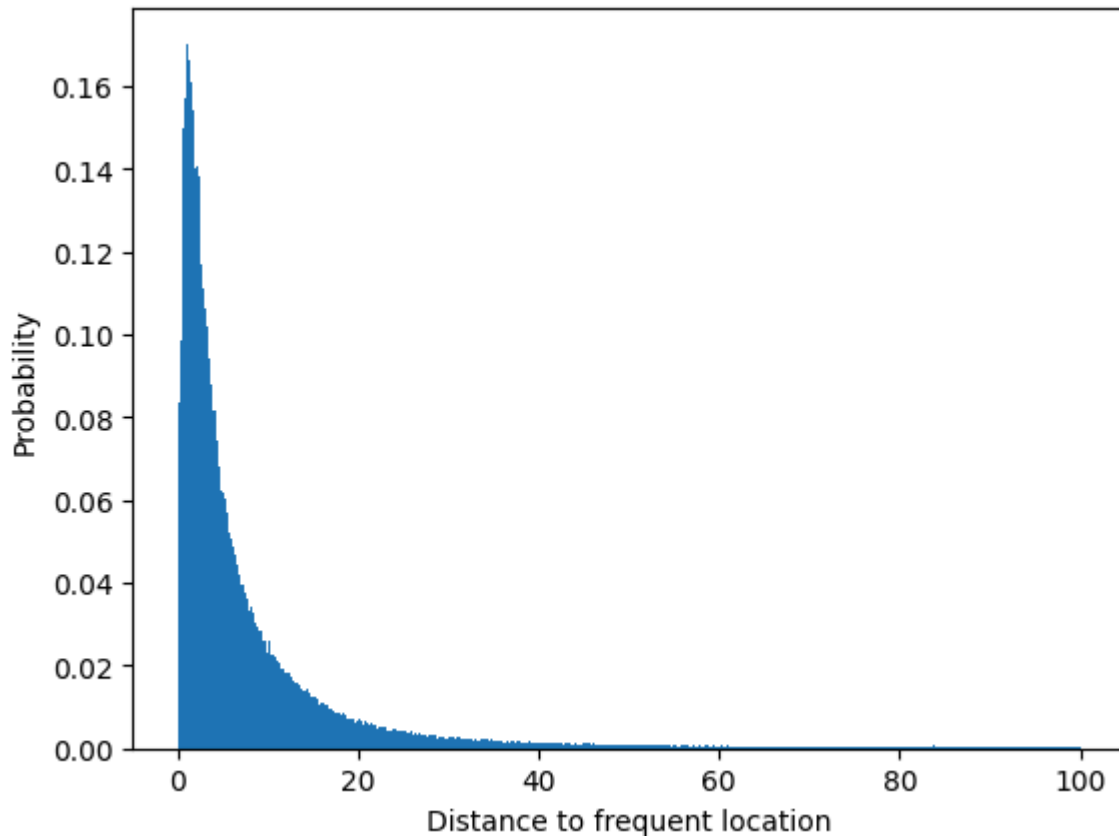
We created a histogram with the distribution of the total value for all transactions, removing the outliers by getting the data with Z-Score < 3 (in a nutshell, it contains ~99.8% of all data, excluding the outliers from both extremes).



**It shows that most of the transactions have transaction value lower than ~ R\$ 200, and transactions with very high tickets need to be checked.**

### Distance to frequent location

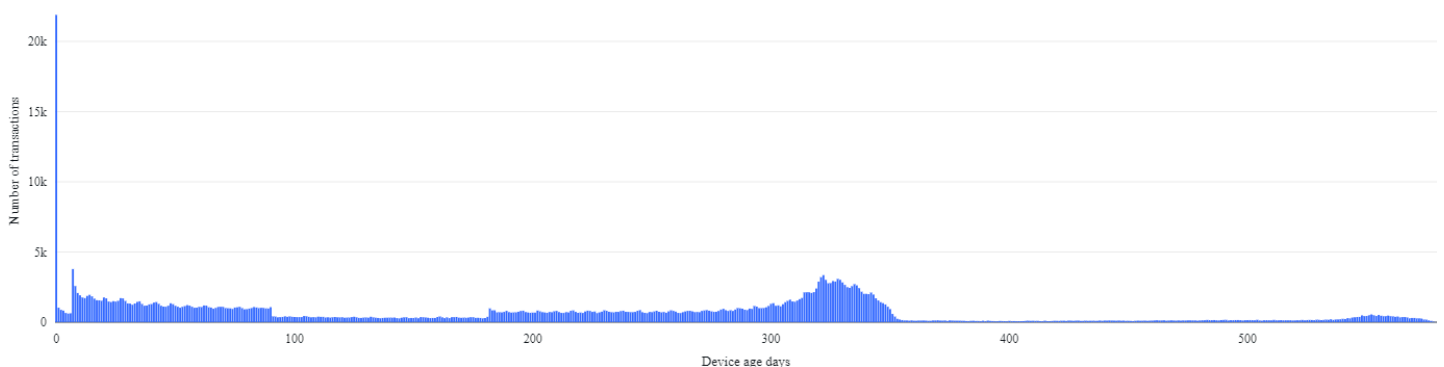
We created the same graph for the distance to the frequent location and removed the outliers, but the data is too sparse (the 3rd quartile is ~46 meters, which means 75% of the data is less than that, and the standard deviation is ~260 kilometers!). The data will be filtered to show only the distribution until 100 meters, and contains approximately 80% of the data.



**Most of the transactions happen closely to the frequent location (< 20 meters),** following the same pattern of the transaction values. If the user creates a transaction very far from the frequent location, **it might indicate an Account Takeover (ATO), for example.**

### Device age days

For device age days, we created the plot for all transactions. The distribution, as shown in the previous topics, **does not have clear outliers, but when we check common patterns, we can see a few:**

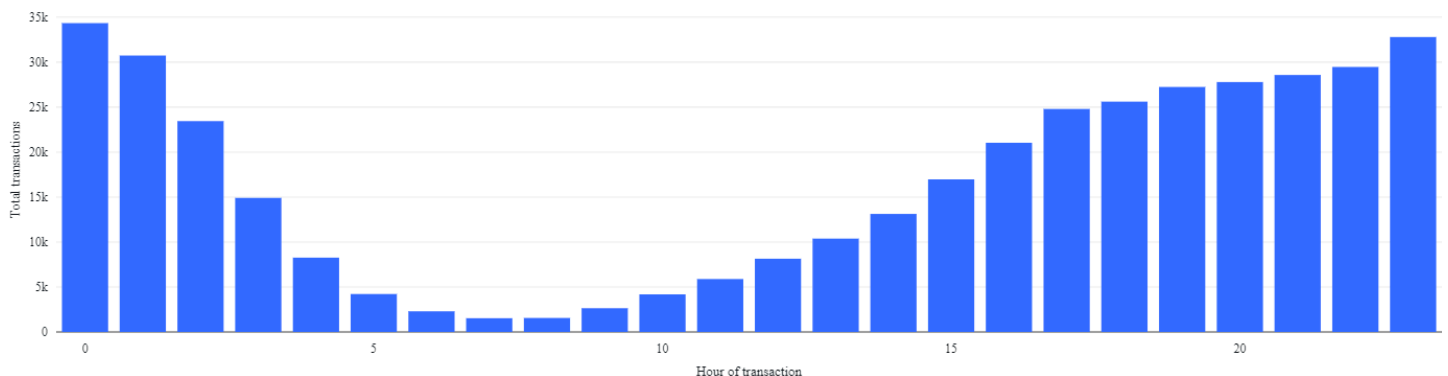


1. **Transactions concentrated in devices with 0 days old:** it indicates that the users are linking the device and right after that, performing a transaction.

2. **An increase in transactions with 7 days old:** Probably a fraud rule, a period that the user must wait to perform a transaction, or one week is the period where the App starts giving incentives to customers.
3. **A gap between 90 and 180 days old:** Another odd behavior. It can be a lot of things, such as stopping promotional incentives at 90 and restarting on 180 days or even data quality problems. We need more information to get a correct answer on why this gap happens.
4. **Another gap between 350 and 530~ days old:** Another odd behavior. We expect to increase the number of transactions from old users / devices, since they didn't churn.

## Transaction time

Following the same principles as in device age days, this plot has the number of all transactions made throughout the day, segmented by hour (from 0 to 23).



This distribution is very odd too, it may need a deeper investigation. **Most of the transactions are happening between late afternoon, at night and at dawn, which is different from what you would expect.** We will check for fraud cases later.

## Client decision and other variables

### Client decision

Here's the comparison between the number of approved and denied transactions from the 2FA process:

**Approved:** 260249 of 399852 (65,08%)

**Denied:** 139603 of 399852 (34,91%)

The amount of verifications that get denied is very high, assuming that the user should perform a type of action to complete the 2FA. There's no Null value in this column.

is\_emulator, has\_fake\_location, has\_root\_permission and app\_is\_tampered

- **is\_emulator**
  - Occurrences: 41 of 399852
  - Null Values: 31
- **has\_fake\_location**
  - Occurrences: 150 of 399852
  - Null Values: 8097
- **has\_root\_permissions**
  - Occurrences: 277 of 399852
  - Null Values: 31
- **app\_is\_tampered**
  - Occurrences: 50 of 399852
  - Null Values: 31

The number of “True” values in these columns are not very relevant, but the **amount of Null Values in the has\_fake\_location column is relatively big, and we need a solution for this case.**

## How's the data quality? How can we fix the problems? Can I trust it?

As we pointed before, there are a few concerns that has to be addressed:

- Null values in the columns, making it hard to evaluate fraud suspicions
- Creating better formats for a few columns
- Odd data in a few columns

**Beyond those points, the data seems to have the right formats, types and is within the expected range.**

### Null Values


Here's a summary of the amount of null values in the dataset

transaction_id	0
transaction_timestamp	0
account_id	0
device_id	0
distance_to_frequent_location	838
device_age_days	0
is_emulator	31
has_fake_location	8097
has_root_permissions	31
app_is_tampered	31
transaction_value	0
client_decision	0

We can see that only 5 columns have null values, but **they are very sensitive for fraud detection**. Because of that, they will not be dropped or filled with data, but **all of these transactions will go through the 2FA validation**. If there's a need to use this data in ML models for training, you may drop these rows.

### Better formats for columns

The only problem regarding column format is the `transaction_timestamp`. There are functions that can solve this problem, such as `timestamp_millis()`. It will look like this:

1711158356605  2024-03-23 01:45:56.605

### Odd data

A few points to look into:

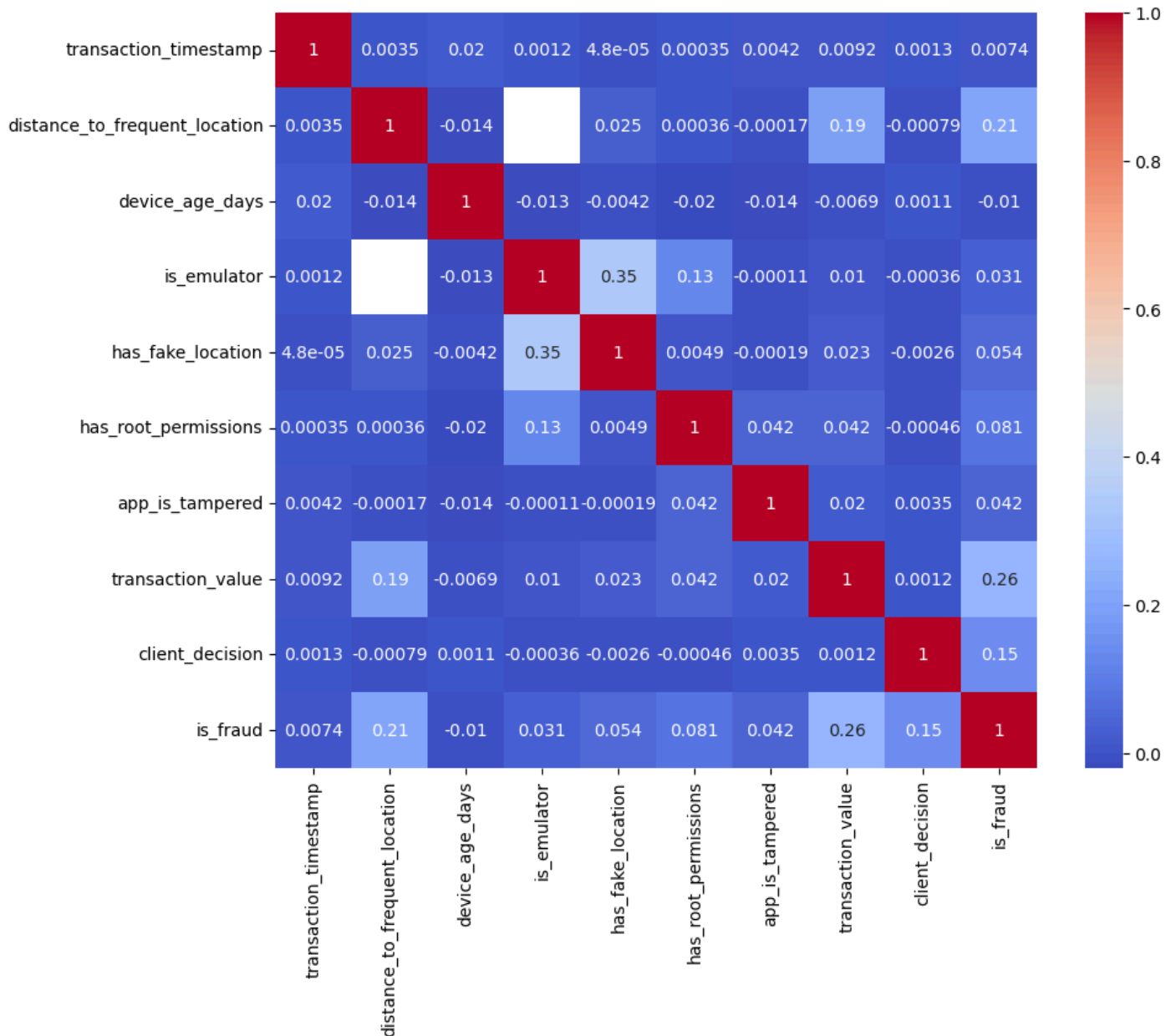
1. The transactions are happening mostly later in the day. It's not very common for banking operation;

2. There are a few gaps in the `device_age_days` that need better investigation;
3. Some of the `distance_to_frequent_location` data is 0. Probably, the distance is very close to the frequent location, or the sensor has some problems.

**Despite those problems, we can continue with the analysis since it doesn't impact the results.**

## Is there a strong correlation between the variables?

Here's a correlation matrix for all variables, plus a column `is_fraud` checking if the transaction was classified as a fraud:



With this matrix, we are able to get a few insights:

1. **`is_emulator` is positively correlated with `has_fake_location`:** An explanation is that when the user uses an emulator, it generates a synthetic location, and probably the App gets this signal;
2. **`is_fraud` is positively correlated with `distance_to_frequent_location` and `transaction_value`:** It indicates that the fraudsters are probably making



transactions from different places (which is expected) and with higher transaction values. If one of those variables is higher than usual, it's a good indication of fraud.

3. **is\_emulator is positively correlated with has\_root\_permissions**: Also expected, since if the person is using an emulation, he needs to have full access to modify any kind of configuration.

This matrix showed that **there are variables that are correlated to each other, and they might be good candidates to identify fraudulent transactions.**

## Are there other possible variables that we can create?

We can create new variables from the ones given by the Digital Bank.

### Devices per account

One of the new possible variables is the **number of devices per account**, which indicates if a user is creating, for example, fake mobiles to perform fraudulent transactions. Checking the data provided, we have this result:

total_devices_by_account ▲	total_accounts ▲
2	35
1	399782

**Indicating that this type of behavior is not happening. Almost all accounts have only one device associated.**

### Accounts per device

Differently from the devices per account variable, there are a significant number of devices with lots of accounts associated, as shown in the images below:

total_accounts_by_device ▲	total_devices ▲		
128	1	22	1
108	1	21	2
106	1	19	4
101	1	18	2
70	1	17	2
63	2	16	1
62	1	15	3
58	2	14	5
56	1	13	4
52	2	12	5
45	1	11	7
43	1	9	25
40	2	8	28
36	1	7	38
34	1	6	49
31	1	5	1
28	1	4	25
26	1	3	939
24	2	2	18117
23	2	1	357763

It shows an odd behavior: there are devices with multiple accounts associated, and we will later see that **a lot of the transactions made by the owners of those devices are classified as fraud.**

### Devices per account vs Accounts per device

With this comparison, we can address a pattern: **some customers are creating new accounts on the same device, and it has a very good chance of fraud attempts.** When we create the new decision flow, it's a pattern to be considered.

# Fraud Risk Classification Rules

The current approach makes all transactions go through a 2FA process, and to create a new decision flow, we need to classify the transactions into 3 groups: **High, Medium and Low risk**. This section will present the fraud data, the criterias and thresholds for each group and the data distribution after the rules are applied.

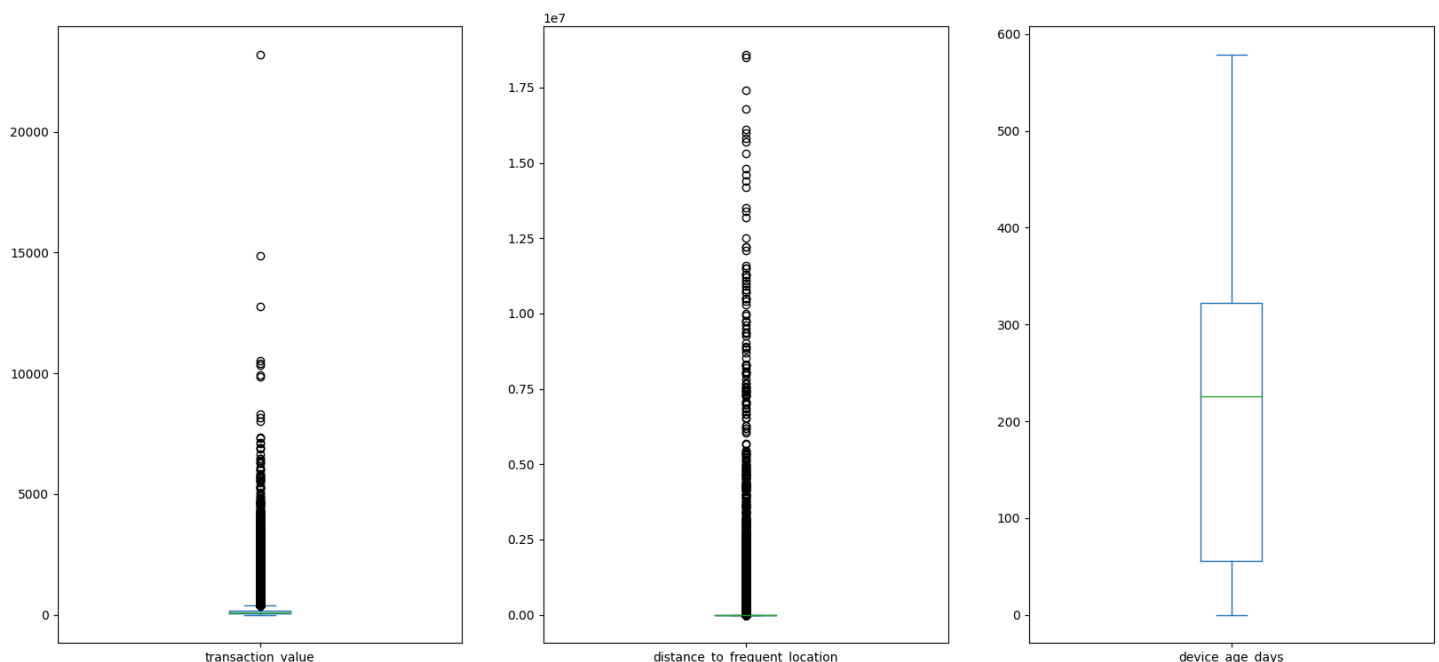
## Fraud data

The Fraud dataset contains the list of transactions that resulted in fraud. It has **15371 transactions\_ids**. We will perform a deep dive on fraud distribution for each relevant variable, showing the patterns that will lead to the thresholds.

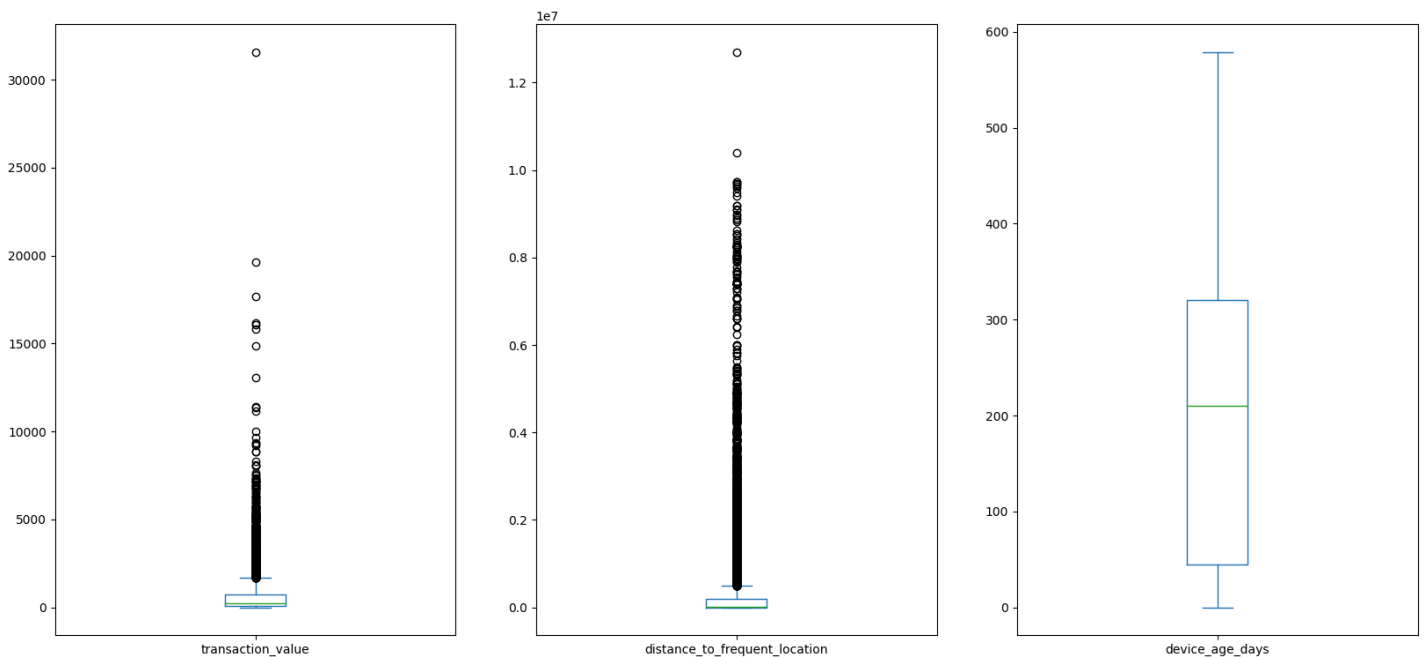
## Fraud vs Good transactions

Plotting the Box plot for both cases, we have these images:

### Good transactions



### Frauds



**Fraudsters have higher transaction values and make them farther from the frequent location, on average. There's no clear difference between them regarding Device Age Days.**

We will dig deeper into these variables and fraud rates (number of fraud transactions vs good ones) in the next section.

## New decision flow: Criteria and Thresholds

### How we will evaluate the transactions

As stated in the documentation:

- **high-risk** transactions would be **immediately blocked**;
- **medium-risk** transactions would follow the current decision flow (**2FA**);
- **low-risk** transaction would be **immediately approved**

For a transaction to be high-risk, we will need to be sure that the transaction has a **very high chance to be a fraud**, since it's immediately blocked. For low-risk, we will prioritize transactions with "common" behavior, and **reducing friction is the main goal**, and for medium-risk, it will be transactions with any kind of **data problems** (null values) or values significant enough to be tested. We will explain the thresholds on the [How did we get these rules](#) section.

## High risk

These transactions have a high chance to be classified as frauds and will be immediately blocked. The rule to classify into high-risk is the following:

1. `distance_to_frequent_location > 1000 km` (or 1.000.000 meters)
2. `is_emulator, has_fake_location, has_root_permission` or `app_is_tampered` are True
3. There are more than 3 accounts linked to the device

## Medium risk

These transactions have at least one variable value that requires a 2FA process to guarantee the approval. The rule to classify into medium-risk is the following:

1. They are not high-risk
2. `transaction_value > R$ 200` OR
3. `distance_to_frequent_location >= 10.000 meters` (10km) OR
4. `device_age_days = 0` OR
5. There's no data in any of the variables (null values)

## Low risk

These transactions will be immediately approved. The rule to classify into low-risk is all transactions that are not in the high or medium-risk, which is:

1. Not in high-risk
2. Not in medium-risk
3. `transaction_value <= R$ 200` AND
4. `distance_to_frequent_location < 10.000 meters` AND
5. `device_age_days > 0`

## In a nutshell

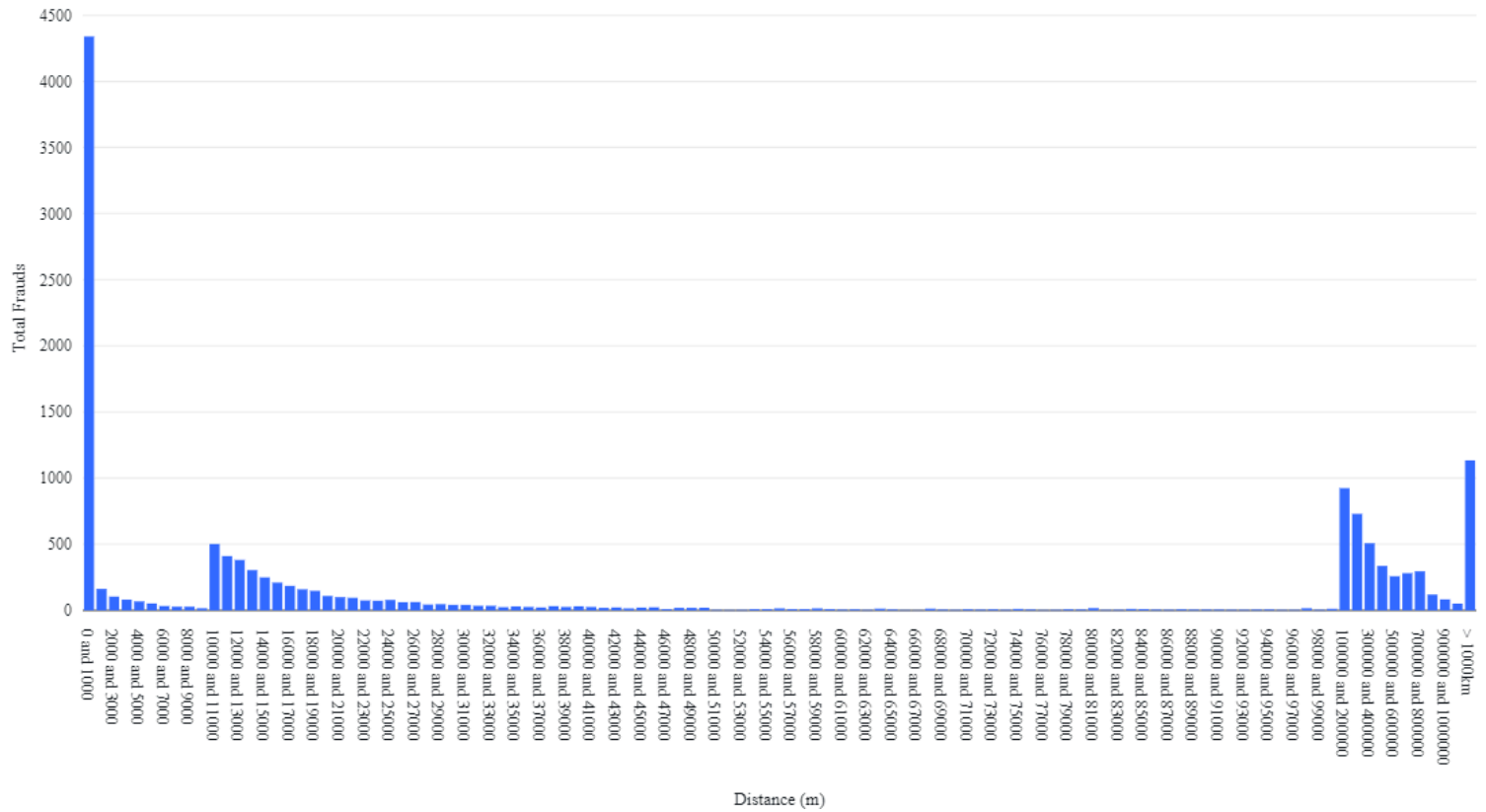
Risk	Criteria and Threshold	# of transactions	%
High	<ul style="list-style-type: none"> <li>- <code>distance_to_frequent_location &gt; 1000 km</code> (or 1.000.000 meters)</li> <li>- <code>is_emulator, has_fake_location, has_root_permission or app_is_tampered</code> are True</li> <li>- There are more than 3 accounts linked to the device</li> </ul>	4662	1,16%
Medium	<ul style="list-style-type: none"> <li>- They are not high-risk</li> <li>- <b>One</b> of the criteria below: <ul style="list-style-type: none"> <li>- <code>transaction_value &gt; R\$ 200</code></li> <li>- <code>distance_to_frequent_location &gt;= 10.000 meters</code> (10km)</li> <li>- <code>device_age_days = 0</code></li> <li>- There's no data in any of the variables (null values)</li> </ul> </li> </ul>	121759	30,45%
Low	<ul style="list-style-type: none"> <li>- They are not high-risk and medium-risk</li> <li>- <b>All</b> of the criteria below: <ul style="list-style-type: none"> <li>- <code>transaction_value &lt;= R\$ 200</code></li> <li>- <code>distance_to_frequent_location &lt; 10.000 meters</code> (10km)</li> <li>- <code>device_age_days &gt; 0</code></li> </ul> </li> </ul>	273431	68,39%

## How did we get to these rules?

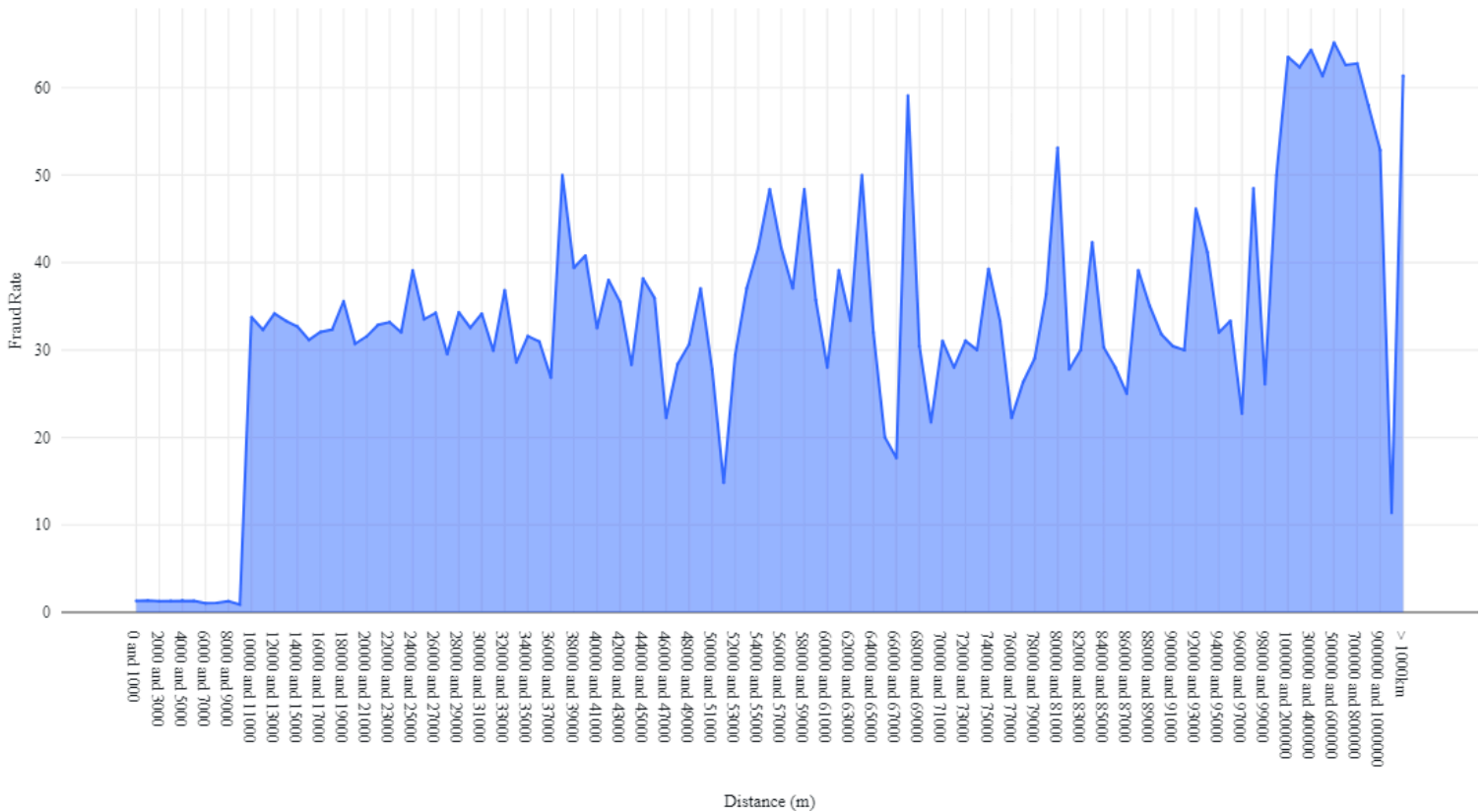
To define the criteria and threshold, we made an analysis of total transactions and total frauds for each meaningful variable.

### Distance to Frequent Location

When we plot the number of fraudulent transactions, segmented by chunks of 1 kilometer (and when it reaches 100km, we create chunks of 100 kms), we can see a pattern: **There's a big difference in total frauds on 1, 10, 100 and > 1000 kms.**



But when we get the Fraud rate (Number of frauds divided by number of transactions), we can see that **it spikes in the 10000 meters (10km), 100.000 (100 km) and > 1.000.000 meters (1.000 kms).**



**We are blocking the transactions with `distance_to_frequent_location > 1000 km (or 1.000.000 meters)`** since it has a very high fraud rate, and it should get international and VPN requests. We chose to not block transactions with `distance_to_frequent_location > 100 km` for user experience purposes. It's very common for people to travel to other cities and still require a banking transaction, so a 2FA requirement should be enough.

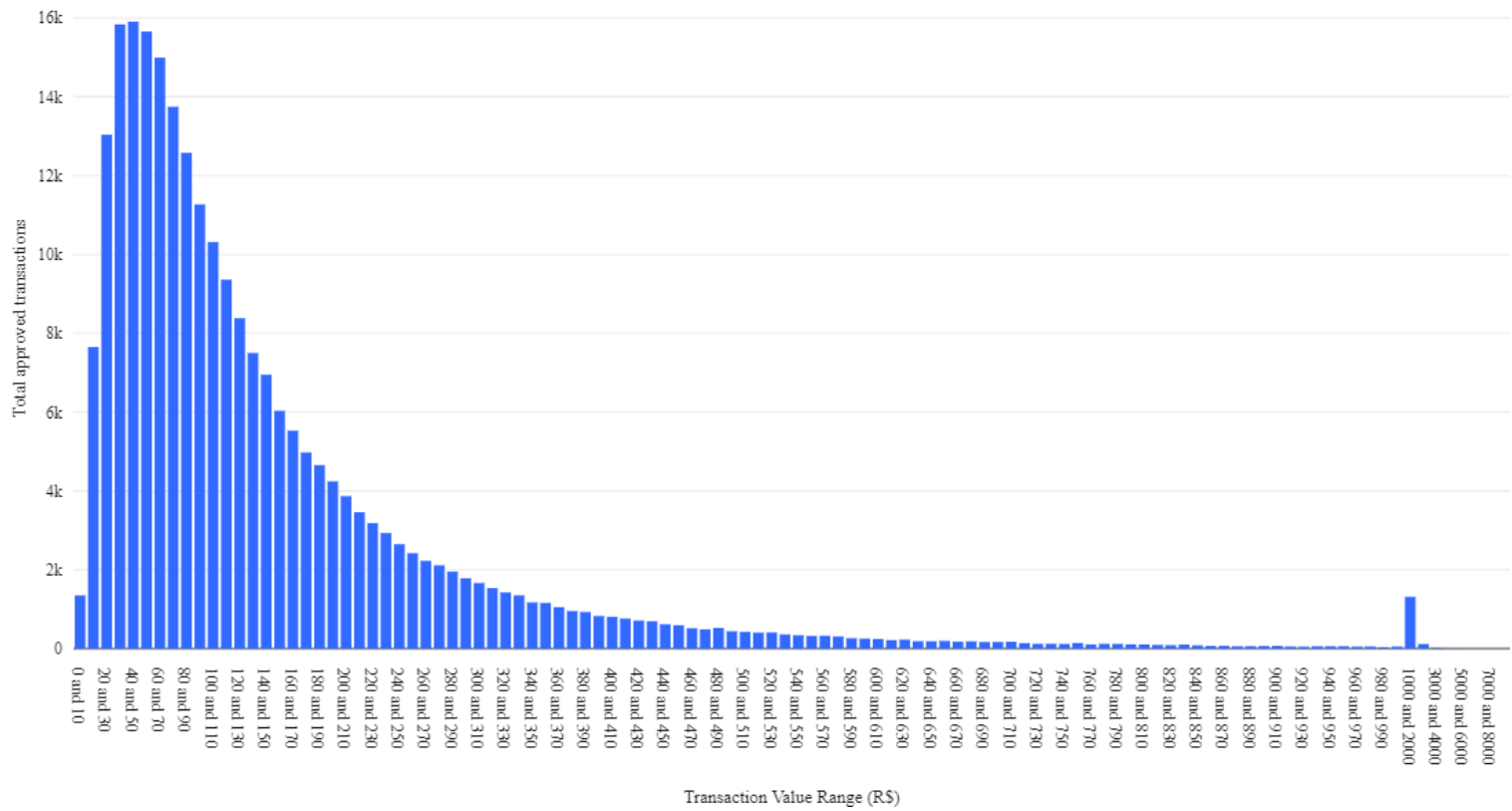
## Transaction Value

For transaction values, we have 2 plots, grouped by value chunks of R\$ 10, and at the end, chunks of R\$ 1000:

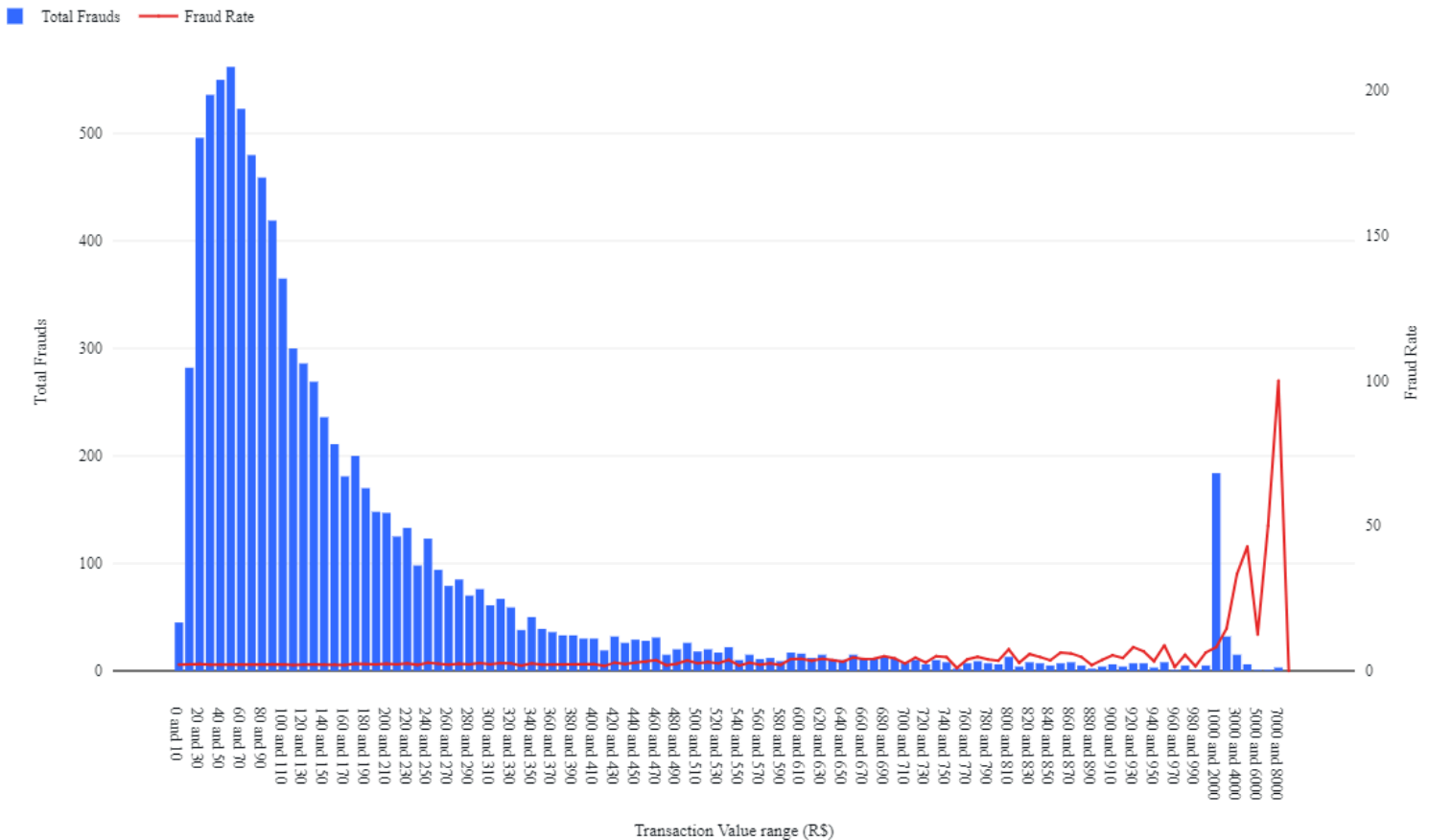
1. The number of approved transactions
2. The number of fraudulent transactions and the fraud rate

As we saw previously, most of the approved transactions have a `transaction_value < R$ 200` (close to 78%), but there's a spike in the R\$ 1000 range that is odd.





But when we check the fraudulent transactions, we can see that only transactions with very high value (> R\$ 1000) have a strong chance of being considered fraud.

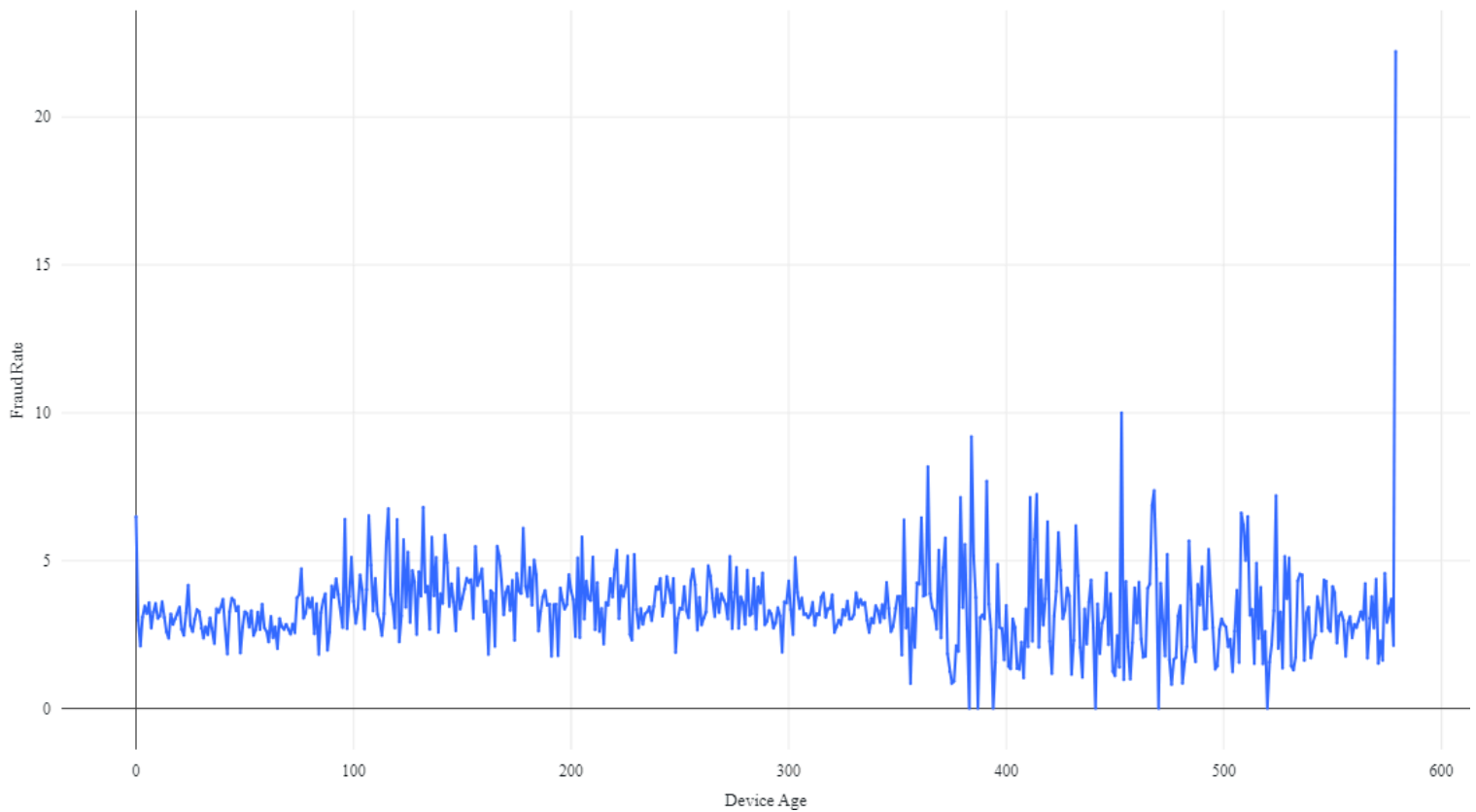
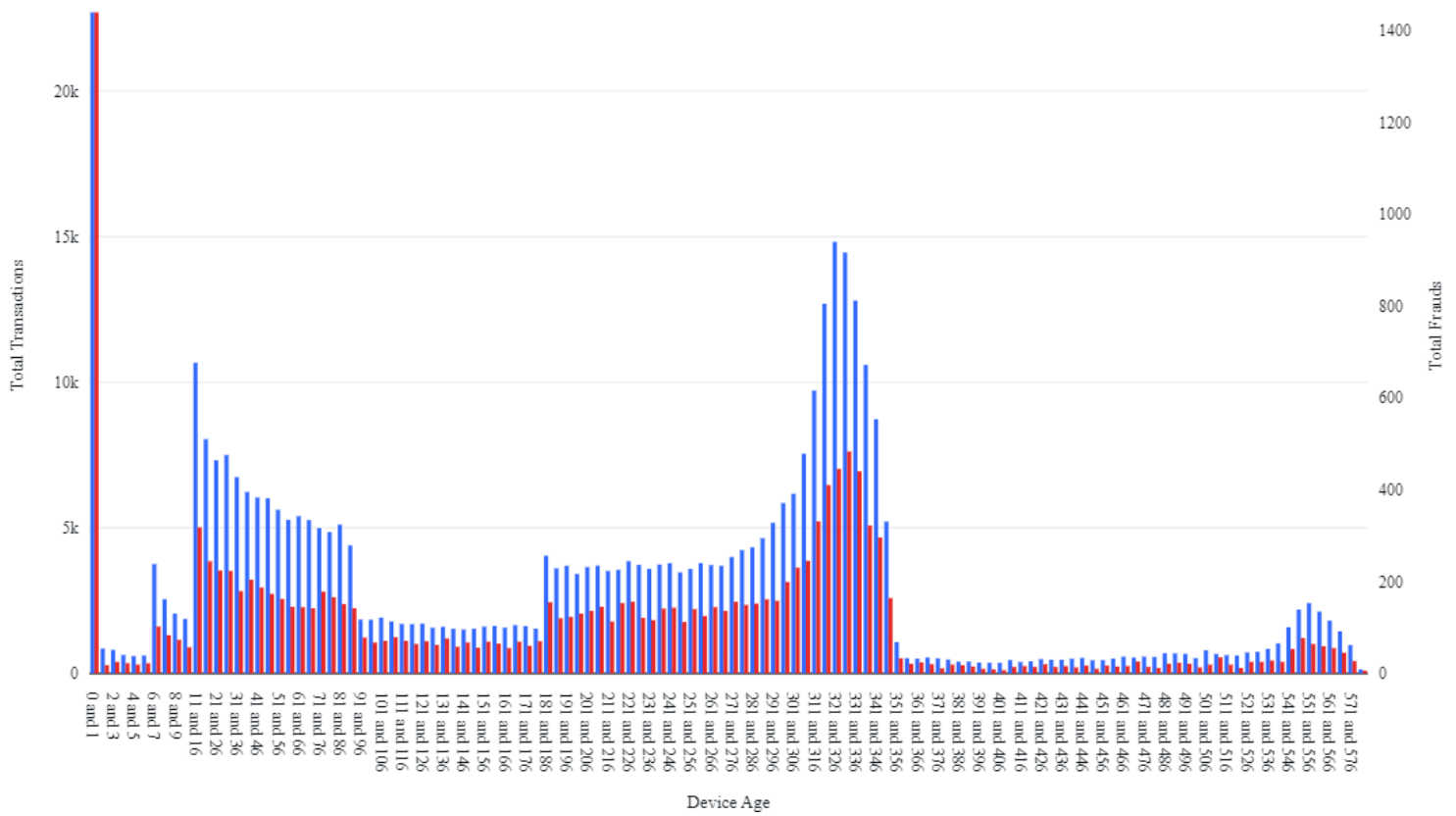


Despite the data showing that a transaction with high value is very likely to be considered a fraud later, for User Experience, **we suggest having a different Authentication on those situations, such as having an Analyst revising it or a “harder” 2FA. Another approach would be having other data from the user, such as previous transaction\_value amount, account balance, etc.**

## Device Age

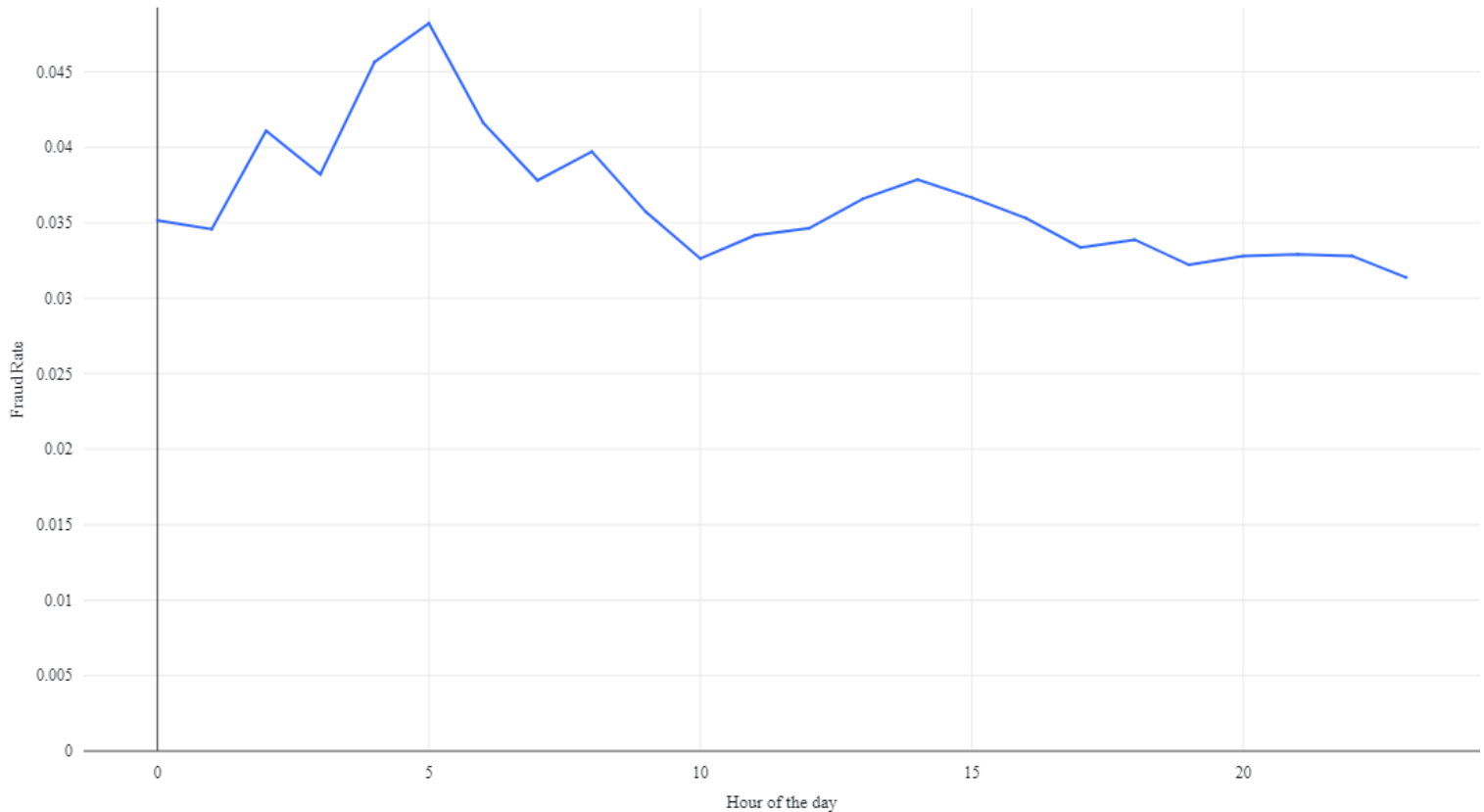
The distribution of frauds vs number of transactions don't show a clear pattern for the device age, but it's very common for fraudsters to **create a transaction at the moment of registration, i.e., device\_age\_days = 0**. The first graph shows the number of transactions and frauds by device age, and the second one, the fraud rating.

■ Total Transactions
 ■ Total Frauds



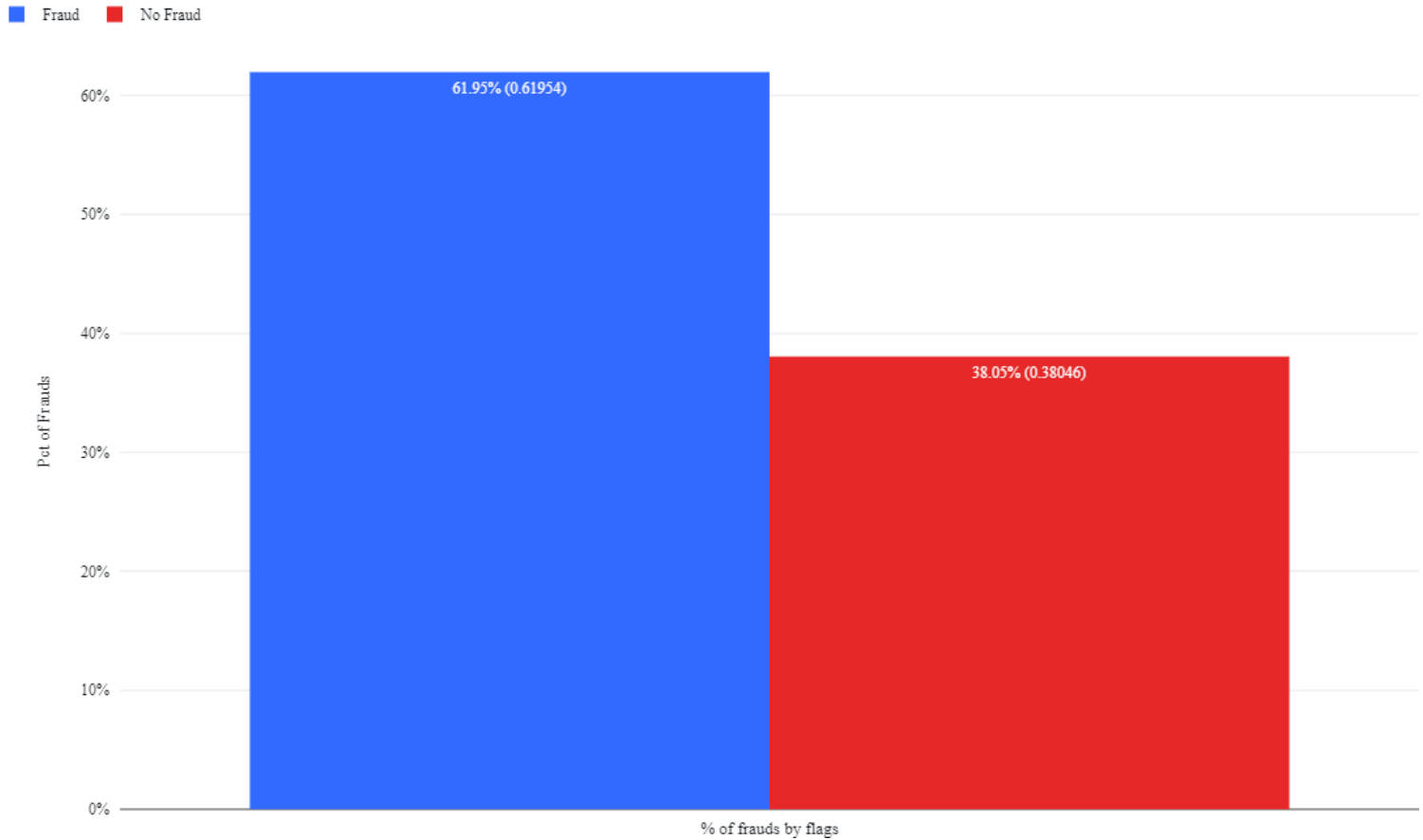
## Hour of the day

Despite the distribution of transactions per hour of the day being odd, **there's no meaningful indication of a period of the day that occurs more fraudulent transactions.**



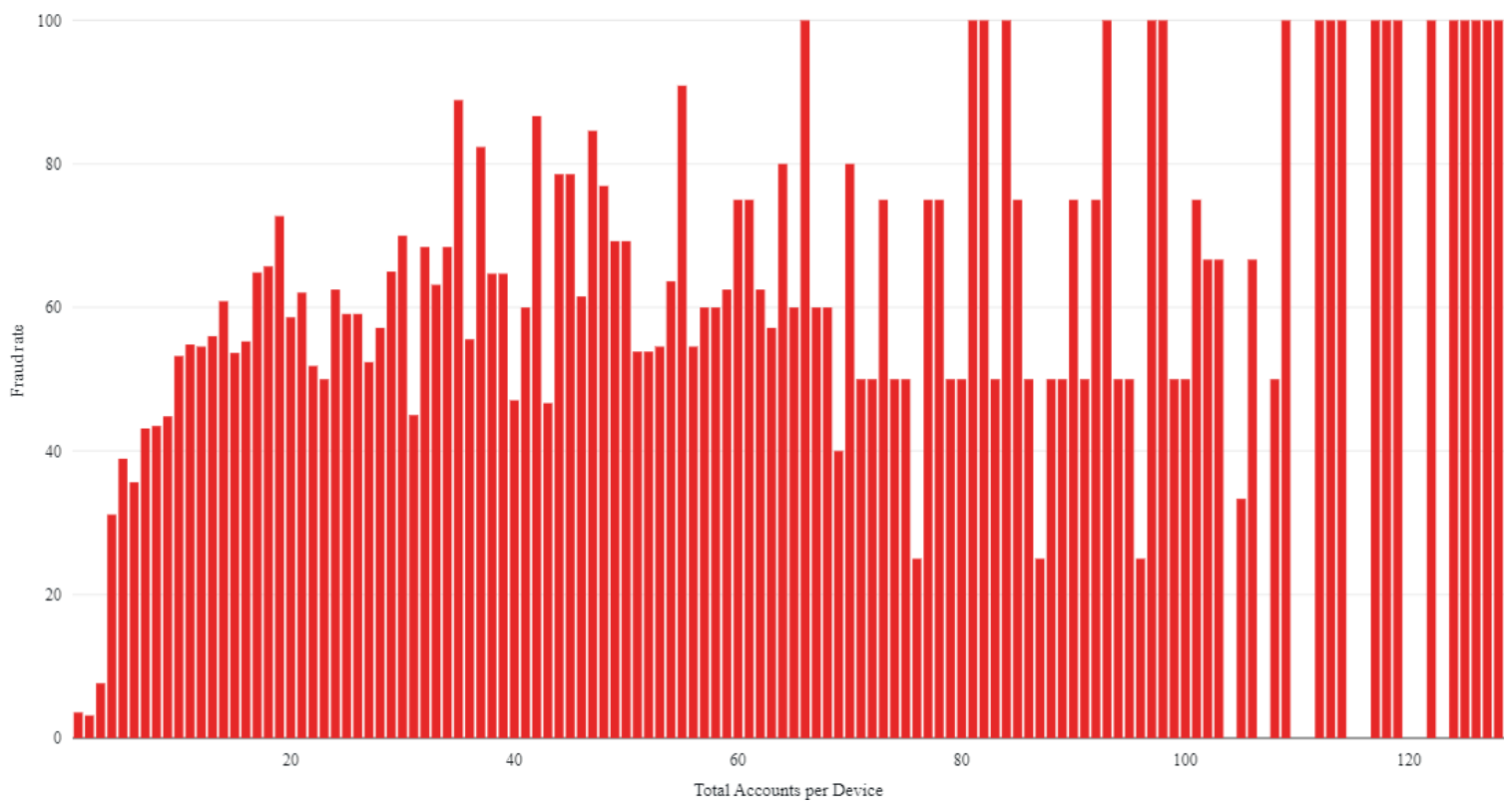
## Flags

It's a very heavy indicator that a transaction is potentially a fraud or will not pass the 2FA, since it's not very common for regular users to have tampered apps, for example. The distribution is the following:



## Accounts per Device

As pointed before, there are devices with multiple accounts linked, and when we plot the Fraud rate, it shows that **devices with more 4 or more accounts associated with them have a high chance (> 35%) of creating fraudulent transactions**, as shown below:



# Evaluation of Classification Rules

## Current vs New decision flow

After implementing these sets of new rules and classification, we expect to create a more assertive fraud detection system and have less friction with “good” users. The current and new decision flow expected behavior is shown below:

Current decision flow	New decision flow
<ul style="list-style-type: none"> <li>all transactions go through 2FA process that costs R\$0.05 per transaction;</li> <li>The app earns 15% of all approved transactions;</li> <li>The app loses 15% of the transaction value for all transactions that are approved and classified as fraud later (return the earned value);</li> </ul>	<p>Based on the new proposed set of rules:</p> <ul style="list-style-type: none"> <li>high-risk transactions would be immediately blocked;</li> <li>medium-risk transactions would follow the current decision flow (2FA);</li> <li>low-risk transaction would be immediately approved</li> </ul> <p>Cost of 2FA and earns from approved transactions remain the same as the current flow</p>

After applying the new rules and classifying the transactions, here’s a comparison table between the current flow and the new one, regarding: Costs & Revenue (Fraud, 2FA, Revenue), Transaction approval rate, False positives and false negatives.

One note: There’s a new concept called (Hard/Soft) False Positives and False Negatives:

- A **Hard False positive** is the person that had the transaction blocked immediately, but was not a fraudster;
- A **Soft False positive** is the person that had to go through the 2FA process, and was not a fraudster;
- A **Hard False negative** is the person that was immediately approved, but ended up being a fraudster;
- A **Soft False negative** is the person that went through the 2FA process, passed and ended up being a fraudster.

	Current	New	Diff (%)
<b>Total fraud transactions</b>	15.371	12.738	- 17,12%
<b>Total fraud costs (Fraudsters + 2FA)</b>	R\$ 1.296.689,72	R\$ 897.158,85*	- 30,81%
<b>Fraud Rate</b>	3,84%	3,19%	- 0,65 pp
<b>2FA Costs</b>	R\$ 19.992,60	R\$ 6.087,40	- 69,55%
<b>Total Revenue (approved - fraud - 2FA cost)</b>	R\$ 4.285.599,5	R\$ 5.566.312,39*	+ 29,88%
<b>Transaction Approvals</b>	260249 (65,09%)	352638 (88,19%)	+ 35,50%
<b>Hard False positives</b>	0 (0%)**	403 (0,1%)	**
<b>Soft False positives</b>	260249 (61,24%)	70043 (17,52%)	- 73,09%
<b>Hard False negatives</b>	0 (0%)**	3585 (0,9%)	**
<b>Soft False negatives</b>	15371 (3,84%)	9153 (2,29%)	- 40,45%

#### Observations:

(\*): It's a "bad realistic scenario", where we expect the users in the low-risk that were denied in the current flow to be fraudster at the same rate as in the general Fraud Rate (3,19%), which is usually higher than we expect, and have the highest possible `transaction_value` (R\$ 200) for the low risk band. The numbers in the most optimistic scenario, where all denied users were actually good users, has Fraud cost = R\$ 805.960,83, with the difference in percentage from the current flow = -37,84%, and Revenue = R\$ 6.803.431,29, with the difference in percentage from the current flow = +58,75%.

(\*\*): The current flow does not block directly any transaction, so it's not possible to compare it.

## Possible impacts

With this change in the decision flow, we will have both positive and negative impacts.

### Positive

- **29,88% uplift** on revenue by **reducing fraud losses by 30,81%** and **2FA costs by 69,55%**;



- **68,39% of transactions** would go through the decision flow **frictionless**, making the **user experience much better**;
- **Protection** from device emulation, tampered apps, location spoofing, jailbreaks, etc
- Increased the **approval rate to 88,19%**;
- Reduced the number of Soft False positives by - **73,09%** and Soft False negatives by - **40,45%**.

## Negative

- Non fraudulent transaction may be **blocked**:
  - One of the rules blocks transactions with distance from the frequent location greater than 1000 kms, and it may cause good international transactions to not be allowed. There's a need to address this problem, such as creating a "Travel mode" for the user.
  - The new high-risk classification blocks devices with more than 3 accounts associated, and there might be good users with this characteristic. Having a message to contact the Fraud Support team should be enough to handle those cases.
- **There might be fraudsters in the low-risk classification**. Having more data and variables will make the decision more precise and with less False Negatives.

## Next steps

Following the implementation of the new decision flow, it might be good to look for more improvements:

- **Data from previous transactions per account**: The Transactions dataset has almost only 1 transaction per account, and it would be very beneficial to get more data from previous interactions, for example:
  - Average transaction values;
  - Total completed transactions
  - Distance from other "frequent" locations,
  - Risk Score from Bureaus
  - etc

And it would make the **detection of Account Takeovers (ATOs)** by difference in the customer behavior;

- **Develop a more robust fraud detection model**, using ML and other technologies: This new decision flow is a rule-based one, and a ML model would get fraudster patterns much better and adapt to new ones. **A combination of rules and models is a good approach.**
- **Increase the number of variables** since we have just 7, and having more data makes the fraud assessment easier and more precise. Examples of new variables:
  - **Network**: User frequent IP, Wi-fi networks, number of fraudulent devices nearby, etc
  - **Location**: High-risk regions, mismatches, etc

- **Previous accounts and devices with fraudulent transactions:** Create a blacklist (or a watchlist if the intent is to not block) with these devices, and increase the chance of blocking or performing a 2FA
- Device information: Year, model, etc
- And many more
- **Create new solutions for corner cases.** For example, people traveling and creating transactions will not be allowed to do that.
- **Having analysts** blocking or allowing transactions with high values and detecting new fraud patterns
- **Developing more types of 2FA:** One that is harder, such as taking an ID photo, and another that is softer, such as inserting the password again.

## Conclusion

With this descriptive analysis, we were able to better understand the users behavior and identify some of the fraud patterns, which led to the creation of new fraud risk classification rules and the new decision flow.

After applying these rules, that classifies the transactions into high, medium and low risk, the Digital Bank will be able to get a **29,88% revenue uplift**, **reduce fraud losses by 30,81%**, **2FA costs by 69,55%**, improve the user experience by letting **68,39% of transactions** happen **frictionless**, **block transactions from device emulation, tampered apps, location spoofing, jailbreaks**, etc, increase the **approval rate to 88,19%** and **reduce the number of Soft False positives by 73,09%** and **Soft False negatives by 40,45%**.

The performance can improve even more if, in the future, they get more variables, such as previous transactions and network data, develop other 2FA solutions and ways to deal with the corner cases, evaluate high-ticket transactions with Fraud Analysts and create a more robust fraud detection model, combining ML models, rules and analysts.

# Appendix

## Github Repository

You can find the link for the Github repository [HERE](#). In the [README.md](#) file, you have the instructions to set the environment.

## Code snippets

The codes that generated the clean data, visualizations and risk bands were generated using **SQL, Python and PySpark**, with the **pandas** and **matplotlib** libraries. For each document section, we will display the relevant code that was used to make the final datasets and graphs.

The development was made in the Databricks environment, using a Community Edition license (free). The CSVs with the data were ingested and stored as a parquet file inside the Databricks File System (dbfs), and the tables created as:

```
CREATE TABLE (table_name)
USING parquet
OPTIONS (path "dbfs:/user/hive/warehouse/(csv_name)")
```

## Descriptive analysis of transaction events

### [About the data](#)

**Showing the first 5 rows of data:**

```
import pandas as pd
from pyspark.sql.functions import *

df_transactions_raw = spark.read.parquet("dbfs:/user/hive/warehouse/transactions")
df_transactions = df_transactions_raw.select("*").toPandas()
df_transactions.head()
```

**Generating a DataFrame with fraud feedback and fixing data type issues:**

```
df_transactions_feedback =
spark.read.parquet("dbfs:/user/hive/warehouse/transactions_fraud_feedback") \
    .withColumn("is_fraud", lit(1))
```

```
df_transactions_with_feedback_raw = df_transactions.join(df_transactions_feedback,
on = "transaction_id", how = "left")

df_transactions_with_feedback =
df_transactions_with_feedback_raw.select("*").toPandas()

df_transactions_with_feedback["is_emulator"] =
df_transactions_with_feedback["is_emulator"].astype('float').astype('Int64')
df_transactions_with_feedback["has_fake_location"] =
df_transactions_with_feedback["has_fake_location"].astype('float').astype('Int64')
df_transactions_with_feedback["has_root_permissions"] =
df_transactions_with_feedback["has_root_permissions"].astype('float').astype('Int64')
df_transactions_with_feedback["app_is_tampered"] =
df_transactions_with_feedback["app_is_tampered"].astype('float').astype('Int64')
df_transactions_with_feedback["client_decision"] =
df_transactions_with_feedback["client_decision"].map({'approved':1, 'denied':0})
df_transactions_with_feedback["is_fraud"] =
df_transactions_with_feedback["is_fraud"].fillna(0)
df_transactions_with_feedback["transaction_timestamp_formatted"] =
pd.to_datetime(df_transactions_with_feedback['transaction_timestamp'], unit='ms')
```

### Getting basic statistics:

```
pd.set_option('display.float_format', lambda x: '%.2f' % x)
df_transactions.describe()
```

### [How's the data distributed?](#)

#### Creating the boxplot

```
import matplotlib.pyplot as plt

df_plot = df_transactions_with_feedback[["transaction_value",
"distance_to_frequent_location", "device_age_days"]]
df_plot.plot(kind='box', subplots=True, sharey=False, figsize=(20, 9))

plt.subplots_adjust(wspace=0.5)
plt.show()
```

## [Attribute's deep dive](#)

### Transaction Value plot

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

data_zscore =
df_transactions_with_feedback[np.abs(stats.zscore(df_transactions_with_feedback["t
ransaction_value"])) < 3]
plt.figure(figsize=(10, 6))
sns.histplot(data=data_zscore, x='transaction_value')
plt.title('Transaction Value')
plt.xlabel('Value (R$)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

### Distance to frequent location plot

```
df_transactions_with_feedback_clean =
df_transactions_with_feedback[["distance_to_frequent_location", "device_age_days",
"transaction_value"]].dropna()
data_zscore = df_transactions_with_feedback_clean
[np.abs(stats.zscore(df_transactions_with_feedback_clean
["distance_to_frequent_location"])) < 3]
data_zscore = data_zscore.where(data_zscore["distance_to_frequent_location"] <
100)
plt.hist(data_zscore["distance_to_frequent_location"], density=True, bins = 500)
plt.ylabel('Probability')
plt.xlabel('Distance to frequent location');
```

### Device age days plot

```
SELECT
    device_age_days,
    count(DISTINCT transaction_id) AS total_transactions
FROM transactions
```

```
GROUP BY 1  
ORDER BY 1
```

### Transaction Time plot

```
with formatted_timestamp as(  
    select t.transaction_id,  
           timestamp_millis(cast(transaction_timestamp as bigint)) as  
transcation_timestamp_formatted  
from transactions t  
)  
  
select  
    hour(transcation_timestamp_formatted) as ts_hour,  
    count(formatted_timestamp.transaction_id) as total_transactions  
from formatted_timestamp  
group by 1  
order by 2 desc
```

[How's the data quality? How can we fix the problems? Can I trust it?](#)

### Null Values

```
df_transactions_with_feedback.isnull().sum()
```

[Is there a strong correlation between the variables?](#)

```
import seaborn as sns  
import matplotlib.pyplot as plt  
  
correlation_matrix = df_transactions_with_feedback.corr()  
plt.figure(figsize=(10, 8))  
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')  
plt.show()
```

[Are there other possible variables that we can create?](#)

### Devices per account

```
SELECT  
    account_id,  
    count(distinct device_id)
```

```
FROM transactions
group by 1
order by 2 desc
```

## Accounts per Device

```
SELECT
    device_id,
    count(distinct account_id) as total_accounts
FROM
    transactions
group by 1
order by 2 desc
```

## Fraud Risk Classification Rules

### [Fraud Data](#)

#### Good transactions box plot

```
df_plot = df_transactions_with_feedback[["distance_to_frequent_location",
"device_age_days", "transaction_value", "is_fraud", "client_decision"]]
df_plot = df_plot[(df_plot["is_fraud"] == 0)]
df_plot = df_plot[["transaction_value", "distance_to_frequent_location",
"device_age_days"]]
df_plot.plot(kind='box', subplots=True, sharey=False, figsize=(20, 9))

plt.subplots_adjust(wspace=2)
plt.show()
```

#### Fraud transactions box plot

```
df_plot = df_transactions_with_feedback[["distance_to_frequent_location",
"device_age_days", "transaction_value", "is_fraud", "client_decision"]]
df_plot = df_plot[df_plot["is_fraud"] == 1]
df_plot = df_plot[["transaction_value", "distance_to_frequent_location",
"device_age_days"]]
df_plot.plot(kind='box', subplots=True, sharey=False, figsize=(20, 9))
plt.show()
```



## New decision flow: Criteria and Thresholds

### High risk band table creation

```
CREATE OR REPLACE TABLE high_risk_band as(
  with has_more_than_4_accounts_per_device as(
    with timestamp_formatted as (
      select
        *,
        timestamp_millis(cast(transaction_timestamp as bigint)) as
transaction_timestamp_formatted
      from
        transactions
    ),
    devices_ordered_transaction_history as (
      SELECT
        a.account_id,
        a.transaction_id,
        a.transaction_timestamp_formatted,
        a.device_id,
        count(distinct b.account_id) as total_accounts
      FROM
        timestamp_formatted a
        INNER JOIN timestamp_formatted b ON a.device_id = b.device_id
        and b.transaction_timestamp_formatted <= a.transaction_timestamp_formatted
      GROUP BY
        a.account_id, a.transaction_id, a.device_id,
a.transaction_timestamp_formatted
    )
    select
      transaction_id
    from
      devices_ordered_transaction_history
    where
      total_accounts >= 4
  ),
  remove_flags as(
    select
      transaction_id
    from
```

```
transactions
where
    has_fake_location = true OR is_emulator = true OR has_root_permissions =
true OR app_is_tampered = true
),
remove_big_distances as(
    select
        transaction_id
    from
        transactions
    where
        distance_to_frequent_location > 1000000
)

select transaction_id
from has_more_than_4_accounts_per_device
union
select transaction_id from remove_flags
union
select transaction_id from remove_big_distances
)
```

### Mid and Low risk band table creation

```
CREATE OR REPLACE TABLE medium_low_band AS(
    select
        t.*,
        case when tff.transaction_id is not null then 1 else 0 end as is_fraud,
        case when t.distance_to_frequent_location is null
            OR t.is_emulator is null
            OR t.has_fake_location is null
            OR t.has_root_permissions is null
            OR t.app_is_tampered is null then 1
            else 0
        end as has_null_value
    from
        transactions t
    left anti join high_risk_band hrb on t.transaction_id = hrb.transaction_id
```

```
left join transactions_fraud_feedback tff on t.transaction_id =  
tff.transaction_id  
)
```

### Creating final table with all risk bands

```
high_risk_query = '''  
select transaction_id from high_risk_band  
'''  
  
all_data_query = '''  
select t.*, case when tff.transaction_id is not null then 1 else 0 end as is_fraud  
from transactions t  
left join transactions_fraud_feedback tff on t.transaction_id = tff.transaction_id  
'''  
  
df_all = spark.sql(all_data_query).toPandas()  
df_high = spark.sql(high_risk_query).toPandas()  
  
df_low_mid = spark.read.table("medium_low_band").select("*").toPandas()  
  
df_low = df_low_mid[(df_low_mid["transaction_value"] < 200) &  
(df_low_mid["distance_to_frequent_location"] < 10000) &  
(df_low_mid["device_age_days"] > 0) & (df_low_mid["has_null_value"] == 0)]  
  
df_mid = df_low_mid[~((df_low_mid["transaction_value"] < 200) &  
(df_low_mid["distance_to_frequent_location"] < 10000) &  
(df_low_mid["device_age_days"] > 0) & (df_low_mid["has_null_value"] == 0))]  
  
df_low = df_low[["transaction_id"]].copy()  
df_low['risk_band'] = "low"  
  
df_mid = df_mid[["transaction_id"]].copy()  
df_mid['risk_band'] = "mid"  
  
df_high = df_high[["transaction_id"]]  
df_high['risk_band'] = "high"  
  
df_all_bands = pd.concat([df_low, df_mid, df_high])
```

```
df_all_with_bands = pd.merge(df_all, df_all_bands, on = "transaction_id", how =  
'left')
```

```
spark_df = spark.createDataFrame(df_all_with_bands)  
spark_df.write.mode("overwrite").saveAsTable("transactions_with_bands")
```

## [How did we get to these rules?](#)

### Distance to Frequent Location Plot

```
%sql  
  
SELECT  
    CASE  
        when distance_to_frequent_location between 0 and 1000 then "0 and  
1000"  
        when distance_to_frequent_location between 1000 and 2000 then "1000  
and 2000"  
        -- ... suppressed because it's too many lines  
        when distance_to_frequent_location between 98000 and 99000 then  
"98000 and 99000"  
        when distance_to_frequent_location between 99000 and 100000 then  
"99000 and 100000"  
        when distance_to_frequent_location between 100000 and 200000 then  
"100000 and 200000"  
        -- ... suppressed because it's too many lines  
        when distance_to_frequent_location between 900000 and 1000000 then  
"900000 and 1000000"  
        when distance_to_frequent_location > 1000000 then "> 1000km"  
    else "unknown" end as distance_break,  
    count(distinct transaction_id) as total_transactions,  
    SUM(is_fraud) as total_frauds,  
    100 * total_frauds / coalesce(total_transactions, 1) as fraud_rate  
from transactions_without_high_risk  
group by 1  
order by  
    case when distance_break = "0 and 1000" then 1  
    when distance_break = "1000 and 2000" then 2  
    -- ... suppressed because it's too many lines
```

```

when distance_break = "98000 and 99000" then 99
when distance_break = "99000 and 100000" then 100
when distance_break = "100000 and 200000" then 101
-- ... suppressed because it's too many lines
when distance_break = "900000 and 1000000" then 109
else 110
end

```

## Transaction Value Plot

```

%sql

SELECT
  CASE
    when transaction_value between 0 and 10 then "0 and 10"
    when transaction_value between 10 and 20 then "10 and 20"
-- ... suppressed because it's too many lines"
    when transaction_value between 980 and 990 then "980 and 990"
    when transaction_value between 990 and 1000 then "990 and 1000"
    when transaction_value between 1000 and 2000 then "1000 and 2000"
-- ... suppressed because it's too many lines
    when transaction_value between 9000 and 10000 then "9000 and 10000"
    when transaction_value > 10000 then "> 10000"
  else "unknown" end as value_break,
  count(distinct transaction_id) as total_transactions,
  SUM(is_fraud) as total_frauds,
  100 * total_frauds / coalesce(total_transactions, 1) as fraud_rate
from transactions_without_high_risk
where distance_to_frequent_location < 100000
group by 1
order by
  case when value_break = "0 and 10" then 1
  when value_break = "10 and 20" then 2
-- ... suppressed because it's too many lines
  when value_break = "980 and 990" then 99
  when value_break = "990 and 1000" then 100
  when value_break = "1000 and 2000" then 101
-- ... suppressed because it's too many lines
  when value_break = "9000 and 10000" then 109
  else 110

```

```
end
```

## Device age days Plot

```
SELECT
    CASE
        when device_age_days between 0 and 1 then "0 and 1"
        when device_age_days between 1 and 2 then "1 and 2"
-- ... suppressed because it's too many lines
        when device_age_days between 9 and 10 then "9 and 10"
        when device_age_days between 11 and 16 then "11 and 16"
        when device_age_days between 16 and 21 then "16 and 21"
-- ... suppressed because it's too many lines
        when device_age_days between 576 and 581 then "576 and 581"
        when device_age_days > 581 then "> 581"
    else "unknown" end as age_break,
    count(distinct transaction_id) as total_transactions,
    SUM(is_fraud) as total_frauds,
    100 * total_frauds / coalesce(total_transactions, 1) as fraud_rate
from transactions_without_high_risk
group by 1
order by
    case
        when age_break = "0 and 1" then 1
        when age_break = "1 and 2" then 2
-- ... suppressed because it's too many lines
        when age_break = "9 and 10" then 10
        when age_break = "11 and 16" then 11
        when age_break = "16 and 21" then 12
        when age_break = "576 and 581" then 124
-- ... suppressed because it's too many lines
    else 125
end
```

## Hour of the day Plot

```
with timestamp_formated as(
    select
        transaction_id,
```

```

        is_fraud,
        timestamp_millis(cast(transaction_timestamp as bigint)) as
transaction_timestamp_formatted
    from
        transactions_without_high_risk
)
select
    hour(transaction_timestamp_formatted) as ts_hour,
    count(transaction_id) as total_transactions,
    sum(is_fraud) / total_transactions as fraud_rate
from
    timestamp_formatted
group by 1
order by 2 desc

```

## Flags Plot

```

with remove_flags as(
    select
        t.transaction_id,
        case when tff.transaction_id is not null then 1 else 0 end as is_fraud
    from
        transactions t
        left join transactions_fraud_feedback tff on t.transaction_id =
tff.transaction_id
    where
        has_fake_location = true OR is_emulator = true OR has_root_permissions =
true OR app_is_tampered = true
)

select
    count(case when is_fraud = 1 then transaction_id end) / count(transaction_id) as
total_frauds,
    count(case when is_fraud = 0 then transaction_id end) / count(transaction_id) as
total_no_frauds
from
    remove_flags

```

**Accounts per Device Plot - When did a device was linked with another account and created a transaction**

```
%sql
with timestamp_formatted as (
    select
        *,
        timestamp_millis(cast(transaction_timestamp as bigint)) as
transaction_timestamp_formatted
    from
        transactions
),
devices_ordered_transaction_history as (
    SELECT
        a.account_id,
        a.transaction_id,
        a.transaction_timestamp_formatted,
        a.device_id,
        count(distinct b.account_id) as CountDistinct
    FROM
        timestamp_formatted a
    INNER JOIN timestamp_formatted b
        ON a.device_id = b.device_id
        and b.transaction_timestamp_formatted <= a.transaction_timestamp_formatted
    GROUP BY
        a.account_id,
        a.transaction_id,
        a.device_id,
        a.transaction_timestamp_formatted
)
select
    CountDistinct,
    count(distinct devs.transaction_id) as total_transactions,
    count(distinct t.transaction_id) as total_frauds,
    100 * total_frauds / coalesce(total_transactions, 1) as fraud_rate
from
    devices_ordered_transaction_history devs
    left join transactions_fraud_feedback t on devs.transaction_id =
t.transaction_id
    left join (
        select
            transaction_id,
            client_decision,
```



```
transaction_value
from
  transactions
) t2 on devs.transaction_id = t2.transaction_id
group by 1
order by 2 desc
```

## Evaluation of Classification Rules

### Current vs New decision flow

#### Fraud data - Current

```
select
  count(distinct t.transaction_id) as total_transactions,
  count(distinct t2.transaction_id) as total_frauds,
  round(sum(case when t2.transaction_id is not null then transaction_value end) *
0.15, 2) as total_fraud_transaction,
  100 * total_frauds / coalesce(total_transactions, 1) as fraud_rate,
  total_fraud_transaction + round(total_frauds * 0.05, 2) as total_fraud_cost
from
  transactions t
  left join transactions_fraud_feedback t2 on t.transaction_id = t2.transaction_id
```

#### Fraud data - New

```
select
  count(distinct transaction_id) as count_transactions,
  count(distinct case when is_fraud = 1 and risk_band = "low" then transaction_id
end) + count(distinct case when is_fraud = 1 and risk_band = "mid" then
transaction_id end) as count_frauds,
  sum(case when is_fraud = 1 and risk_band = "low" then transaction_value end) as
total_frauds_low_band,
  count(distinct case when is_fraud = 0 and risk_band = "low" and client_decision
= "denied" then transaction_id end) as count_low_band_denied,
  sum(case when is_fraud = 0 and risk_band = "low" and client_decision = "denied"
then transaction_value end) as total_low_band_denied,

  sum(case when is_fraud = 1 and risk_band = "mid" then transaction_value end) as
total_frauds_mid_band,
```

```
count(distinct case when is_fraud = 1 and risk_band = "mid" then transaction_id
end) as count_frauds_mid_band,

total_frauds_low_band * 0.15 + total_frauds_mid_band * 0.15 as total_fraud,
100 * count_frauds / coalesce(count_transactions, 1) as fraud_rate,
total_fraud + round(count_frauds_mid_band * 0.05, 2) as total_fraud_cost,
total_fraud_cost + ((fraud_rate / 100) * 200 * count_low_band_denied * 0.15) as
total_fraud_cost_realistic_case

from
transactions_with_bands
```

## 2FA data - Current

```
select
count(distinct t.transaction_id) as total_transactions,
round(total_transactions * 0.05, 2) as total_2fa_cost
from
transactions t
```

## 2FA data - New

```
select
count(distinct case when risk_band = "mid" then transaction_id end) as
count_mid_band,
round(count_mid_band * 0.05, 2) as total_2fa_cost
from
transactions_with_bands
```

## Revenue data - Current

```
select
count(distinct t.transaction_id) as total_transactions,
round(sum(case when t2.transaction_id is not null then transaction_value end) *
0.15, 2) as total_fraud_transaction,
round(sum(case when t.client_decision = "approved" and t2.transaction_id is null
then transaction_value end) * 0.15, 2) as total_approved_transaction,
coalesce(total_approved_transaction, 0) - coalesce(total_fraud_transaction, 0) -
coalesce((0.05 * total_transactions), 0) AS total_revenue
```

```
from
  transactions t
  left join transactions_fraud_feedback t2 on t.transaction_id = t2.transaction_id
```

### Revenue data - New

```
select
  sum(case when is_fraud = 0 and risk_band = "low" and client_decision = "denied"
then transaction_value end) as total_low_band_denied,

  count(distinct case when risk_band = "mid" then transaction_id end) as
count_mid_band,

  sum(case when (risk_band = "low") or (risk_band = "mid" and client_decision =
"approved") then transaction_value end) as total_approved,
  sum(case when (is_fraud = 1 and risk_band = "low") or (risk_band = "mid" and
is_fraud = 1 and client_decision = "approved") then transaction_value end) as
total_fraud_approved,

  sum(case when (risk_band = "mid" and is_fraud = 1 and client_decision =
"denied") then transaction_value end) as total_fraud_denied,

  round(coalesce(total_approved, 0) * 0.15, 2) -
round(coalesce(total_fraud_approved, 0) * 0.15, 2) -
round(coalesce(total_fraud_denied, 0) * 0.15, 2) - round(coalesce(count_mid_band,
0) * 0.05, 2) as total_revenue,
  round(coalesce(total_low_band_denied, 0) * 0.15, 2) as
total_approved_cost_if_all_fraudsters,
  total_revenue - total_approved_cost_if_all_fraudsters as worst_case_scenario
from
  transactions_with_bands
```

### Transaction approval rate data - Current

```
select
  count(distinct t.transaction_id) as total_transactions,
  count(distinct case when t.client_decision = "approved" then t.transaction_id
end) as total_approvals,

  total_approvals / total_transactions as total_approval_rate
```

```
from
  transactions t
  left join transactions_fraud_feedback t2 on t.transaction_id = t2.transaction_id
```

### Transaction approval rate data - New

```
select
  count(distinct transaction_id) as count_transactions,
  count(distinct case when (risk_band = "low") or (risk_band = "mid" and
client_decision = "approved") then transaction_id end) as count_approved,

  count_approved / count_transactions as total_approved
from
  transactions_with_bands
```

### False positives (Hard and Soft) data - Current

```
select
  count(distinct t.transaction_id) as total_transactions,
  0 as hard_false_positives,
  hard_false_positives / total_transactions as hard_false_positive_rate,
  count(distinct case when t.client_decision = "approved" and t2.transaction_id is
null then t.transaction_id end) as soft_false_positives,
  soft_false_positives / total_transactions as soft_false_positive_rate

from
  transactions t
  left join transactions_fraud_feedback t2 on t.transaction_id = t2.transaction_id
```

### False positives (Hard and Soft) data - New

```
select
  count(distinct transaction_id) as total_transactions,
  count(distinct case when is_fraud = 0 and risk_band = "high" and client_decision
= "approved" then transaction_id end) as hard_false_positives,
  count(distinct case when is_fraud = 0 and risk_band = "mid" and client_decision
= "approved" then transaction_id end) as soft_false_positives,
  hard_false_positives / total_transactions as hard_false_positive_rate,
  soft_false_positives / total_transactions as soft_false_positive_rate
```

```
from transactions_with_bands
```

### False negatives (Hard and Soft) data - Current

```
select
  count(distinct t.transaction_id) as total_transactions,
  0 as hard_false_negatives,
  hard_false_negatives / total_transactions as hard_false_negative_rate,
  count(distinct case when t.client_decision = "approved" and t2.transaction_id is
not null then t.transaction_id end) as soft_false_negative,
  soft_false_negative / total_transactions as soft_false_negative_rate

from
  transactions t
  left join transactions_fraud_feedback t2 on t.transaction_id = t2.transaction_id
```

### False negatives (Hard and Soft) data - New

```
select
  count(distinct transaction_id) as total_transactions,
  count(distinct case when is_fraud = 1 and risk_band = "low" then transaction_id
end) as hard_false_negative,
  count(distinct case when is_fraud = 1 and risk_band = "mid" and client_decision
= "approved" then transaction_id end) as soft_false_negative,
  hard_false_negative / total_transactions as hard_false_negative_rate,
  soft_false_negative / total_transactions as soft_false_negative_rate

from transactions_with_bands
```