Strategy & Operations
Revolut : Pedro Brito

Helping our FC Squad

Executive Summary (see pages below for detailed analysis)

- Task A1:
  - Gaia tables filtered, and structure of each table analysed, creation of SQL queries.
  - creation of final table containing customer_id, customer_country, transaction_amount and currency.. Relevant code can be seen below in “Task in detail”

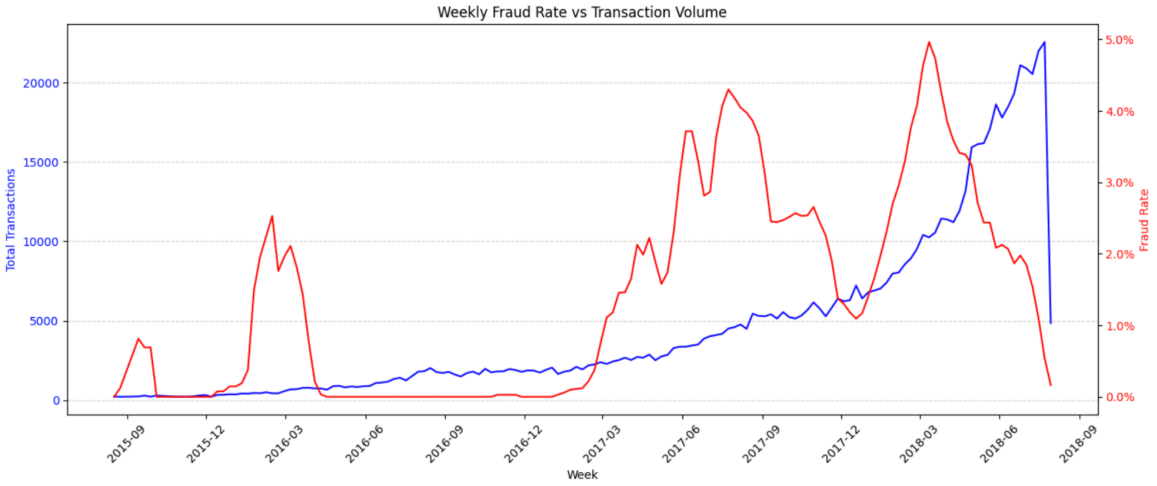
[7]:

	customer_id	customer_country	transaction_amount	currency
0	7285c1ec-31d0-4022-b311-0ad9227ef7f4	GB	37.38	GBP
1	20100a1d-12bc-41ed-a5e1-bc46216e9696	GB	5.88	GBP
2	0fe472c9-cf3e-4e43-90f3-a0cfb6a4f1f0	GB	12.64	GBP
3	20100a1d-12bc-41ed-a5e1-bc46216e9696	GB	0.66	GBP
4	821014c5-af06-40ff-91f4-77fe7667809f	GB	9.68	GBP
5	fbe6dfd9-96de-4fde-af16-32e8a0bb7a25	GB	16.41	GBP
6	dd1f6199-127f-49ff-ba25-81393b2e66f2	GB	65.06	USD
7	dd1f6199-127f-49ff-ba25-81393b2e66f2	GB	96.93	USD
8	20100a1d-12bc-41ed-a5e1-bc46216e9696	GB	2.95	GBP
9	1a9e22bd-4cec-47c4-8196-92038ba2e603	GR	0.89	EUR

- Task A2:
  - data integrity checks completed to check quality of data (negative values, zero-valued transitions, transactions without ID)
  - calculated number of card users that had a first transference about 10 USD (code can be seen in “Task in detail”.
  - **447** users had a first transaction recorded as a successful card payment over 10 USD. Which represents 4.34% of all users in the system (10300 total), or 5.57% when considering only users who performed at least one transaction (8021 total).

- Task B:
  - created metrics by comparing behavior of known-fraudster and non-fraudster.
  - with those metrics created a filter where points were attributed to people that matched fraudulent behavior.
  - came up with top 5 fraudsters IDs:
    - **user ID**: 450dc7bf-d8d2-4989-97b9-b54669255ef0
    - **user ID:** febeeef3-19a9-4e9b-8f57-4398f400f3f8
    - **user ID:** 3011a01f-e9b0-477e-9791-9aa5ae6abd86
    - **user ID:** 30e56a5d-ef22-4ee1-93f7-8ffbf134fb9e
    - **user ID:** df5112c5-11fe-4a59-9e12-cbbab7873b43

- Task B* (optional)
    - compared total transactions with fraud rate using data provided.
    - results showed continuous and exponential transaction growth.
    - results also showed spikes in fraud rate.
    - special attention should be give to the final spike, after 03/2018 where a spike started decaying, even though transactions were still increasing.
    - We suggested further investigation in trying to understand why this happened. See graph below.



Tasks in detail

To complete the tasks in this report, a Jupyther Notebook with Python was used. After some research, it was discovered that the Revolut Team actually uses Trino with Jupyther. However, due to time constraints this was not used. For each problem, the code was shown with a brief paragraph highlighting the rationale behind it.

A1 - Code (customer_id, customer_country, and transaction_amount)

- The following code should be used by the FC team to get the customer_id, customer_country and transaction amount + currency (to make it more complete)
- Make sure to change the folder_path when running the code locally.

[1]:
```python
import pandas as pd

# files can be located in this folder
folder_path = "/Users/pedrobrito/Desktop/Pedro - Revolut/ 2. FC folder/hc_data/"

# Load all provided CSVs
transactions = pd.read_csv(folder_path + "transactions.csv")
users = pd.read_csv(folder_path + "users.csv")
currency_details = pd.read_csv(folder_path + "currency_details.csv")
fx_rates = pd.read_csv(folder_path + "fx_rates.csv")
fraudsters = pd.read_csv(folder_path + "fraudsters.csv")
countries = pd.read_csv(folder_path + "countries.csv")

print("✓ All files loaded successfully")
```

```
✓ All files loaded successfully
```

[5]:
```python
#Check structure of each dataset

print("📄 transactions.csv columns:\n", transactions.columns, "\n")
print(transactions.head(3), "\n")

print("📄 users.csv columns:\n", users.columns, "\n")
print(users.head(3), "\n")

print("📄 currency_details.csv columns:\n", currency_details.columns, "\n")
print(currency_details.head(3), "\n")

print("📄 fx_rates.csv columns:\n", fx_rates.columns, "\n")
print(fx_rates.head(3), "\n")

print("📄 fraudsters.csv columns:\n", fraudsters.columns, "\n")
print(fraudsters.head(3), "\n")

print("📄 countries.csv columns:\n", countries.columns, "\n")
print(countries.head(3), "\n")

currency_details["exponent"].value_counts()
```

[7]:
```
# Step A.1 – Build GAIA transactions summary table

# 1. Filter only GAIA transactions
gaia_tx = transactions[transactions["SOURCE"] == "GAIA"].copy()

# 2. Merge with users to get customer country
gaia_tx = gaia_tx.merge(users[["ID", "COUNTRY"]],
                        left_on="USER_ID", right_on="ID", how="left")

# 3. Merge with currency_details to get exponent for amount conversion
gaia_tx = gaia_tx.merge(currency_details[["currency", "exponent"]],
                        left_on="CURRENCY", right_on="currency", how="left")

# 4. Convert amount to readable value
gaia_tx["transaction_amount"] = gaia_tx["AMOUNT"] / (10 ** gaia_tx["exponent"])

# 5. Select final columns and rename safely
gaia_summary = gaia_tx[["USER_ID", "COUNTRY", "transaction_amount", "CURRENCY"]].rename(
    columns={"USER_ID": "customer_id", "COUNTRY": "customer_country", "CURRENCY":"currency"}
)

#confrimation that its working by just pulling 10 rows
gaia_summary.head(10)
```

[7]:

	customer_id	customer_country	transaction_amount	currency
0	7285c1ec-31d0-4022-b311-0ad9227ef7f4	GB	37.38	GBP
1	20100a1d-12bc-41ed-a5e1-bc46216e9696	GB	5.88	GBP
2	0fe472c9-cf3e-4e43-90f3-a0cfb6a4f1f0	GB	12.64	GBP
3	20100a1d-12bc-41ed-a5e1-bc46216e9696	GB	0.66	GBP
4	821014c5-af06-40ff-91f4-77fe7667809f	GB	9.68	GBP
5	fbe6dfd9-96de-4fde-af16-32e8a0bb7a25	GB	16.41	GBP
6	dd1f6199-127f-49ff-ba25-81393b2e66f2	GB	65.06	USD
7	dd1f6199-127f-49ff-ba25-81393b2e66f2	GB	96.93	USD
8	20100a1d-12bc-41ed-a5e1-bc46216e9696	GB	2.95	GBP
9	1a9e22bd-4cec-47c4-8196-92038ba2e603	GR	0.89	EUR

A1. Explanation (customer_id, customer_country, and transaction_amount)

1. Filtering only GAIA transactions

- The issue was related specifically to “GAIA”, which is one of Revolut’s internal payment systems. Many other systems (example: “HERA”, “MINOS”) exist in the same dataset but are not relevant for this task.
- By filtering by “SOURCE = GAIA”, it ensures we are showing the correct subset of transactions that the FC team is focused on.

2. Inspecting structure of each dataset

- Before building any logic to solve the problem identified in the dashboard, and to understand what data was available on each datasets, a structural check was done.

This was specially useful to understand the information we had available (naming conventions) and how we could join it together.

- The column names of each dataset were printed as well as the first few rows. From this we spotted that some naming conventions did not cross-over between datasheets (“USER_ID” in one dataset and “ID” in another).
- From the transactions.csv file, we saw that the “AMOUNT” column, stores monetary values. However, these are not in standard units (dollars, euros etc). Instead, values are stored in the smallest currency unit (cents USD and EUR, full units for JPY etc). This was confirmed by inspecting currency_details.csv, where each currency had an exponent number associated with it. This refers to how many decimal places we had to apply.
- It was also observed that the transaction.cvs does not contain the customers country directly. Yet, the users.csv file includes this information under the column “COUNTRY”, which we could retrieve by joining on the “USER_ID”.
- Therefore the information required to solve the issue the FC team is facing, is located in different datasets that had to be combined logically. While the transactions.csv contains the raw transactions (“USER_ID”, “CURRENCY”, “AMOUNT”, “SOURCE”), the users.csv can be used to add user-level context (“ID”, “COUNTRY”) and currency_details.csv (“CURRENCY”, “EXPONENT”) enables proper monetary conversion.

3. SQL query and how data was worked

   3.1 customer_country created using users table

- The transaction file only includes a “USER_ID”, not the customer’s country.
- To show the country, we had to combine it with the users table (users.csv file), which maps user “IDs” to their country. This is specially relevant since it creates a missing parameter related to the location of the clients, which is relevant for spotting fraud and patterns associated with user geography.

   3.2 Raw transaction amount converted into readable values

- The transactions.csv file stores amounts in the smallest currency unit (example: pennies for GBP).
- Working with such a small currency unit can be confusing to users and show misleading data. Displaying this raw amount would confuse users or show misleading data (example, "5000" instead of "50.00").
- So we joined with the currency_details.csv file, which tells us how to convert each currency using its exponent.
- This conversion ensures the numbers shown on the dashboard match what customers actually spend; it's now “real” money.

   3.3 Final result and adding extra “CURRENCY” column

   The FC team was asking for customer_id, customer_country, and transaction_amount only, which was printed using information mentioned above. However, we noted that even though it is correct, the transaciton_amount on its own

lacks meaning without a reference to its currency. To address this, and even though it can be considered optional, we included the original “CURRENCY” field in the final output to maintain clarity and prevent misinterpretation across different currencies.

A2. Code (Transaction Success Rate KPI - First card payment over 10 USD)

1. Data Integrity Validation Checks - Check if data is reliable and free of issues

```
# Task A2
# Data Integrity Validation Checks - Check if data is reliable and free of issues
print("=== TRANSACTIONS DATA VALIDATION ===")

# 1. CREATED_DATE check
missing_dates = transactions["CREATED_DATE"].isnull().sum()
print(f"Transactions missing a date: {missing_dates}")
try:
    transactions["CREATED_DATE"] = pd.to_datetime(transactions["CREATED_DATE"])
    print("Date format: OK")
except Exception as e:
    print(f"Date format issue: {e}")

# 2. USER_ID check
missing_users = transactions["USER_ID"].isnull().sum()
print(f"Transactions missing a user ID: {missing_users}")

# 3. Duplicate transaction IDs
duplicate_ids = transactions["ID"].duplicated().sum()
print(f"Duplicate transaction IDs: {duplicate_ids}")

# 4. Amount checks
zero_value_tx = (transactions["AMOUNT"] == 0).sum()
negative_value_tx = (transactions["AMOUNT"] < 0).sum()
print(f"Transactions with zero value: {zero_value_tx}")
print(f"Transactions with negative value: {negative_value_tx}")

# 5. Currency consistency
unmatched_currencies = set(transactions["CURRENCY"].unique()) - set(currency_details["currency"].unique())
print(f"Currencies not in currency details file: {len(unmatched_currencies)}")
```

```
=== TRANSACTIONS DATA VALIDATION ===
Transactions missing a date: 0
Date format: OK
Transactions missing a user ID: 0
Duplicate transaction IDs: 0
Transactions with zero value: 12023
Transactions with negative value: 0
Currencies not in currency details file: 0
```

2. Transaction Success Rate (KPI) of users that had a successful first card transaction over 10 USD

```python
# Task A2
# KPI: Users whose FIRST transaction was a successful card payment over $10 USD

# Filter transactions in USD
usd_tx = transactions[transactions["CURRENCY"] == "USD"]
print(f"USD transactions: {len(usd_tx)}")

# Get exponent for USD
usd_exponent = currency_details[currency_details["currency"] == "USD"]["exponent"].values[0]

# Convert amount from cents to dollars (create a safe copy first)
usd_tx = usd_tx.copy()
usd_tx["amount_converted"] = usd_tx["AMOUNT"] / (10 ** usd_exponent)

# Sort by USER_ID and CREATED_DATE to get first transaction per user
usd_tx_sorted = usd_tx.sort_values(by=["USER_ID", "CREATED_DATE"])

# Get the first transaction for each user
first_usd_tx = usd_tx_sorted.groupby("USER_ID").first().reset_index()

# Filter: first transaction must be a completed card payment over $10
first_card_over10 = first_usd_tx[
    (first_usd_tx["STATE"] == "COMPLETED") &
    (first_usd_tx["TYPE"] == "CARD_PAYMENT") &
    (first_usd_tx["amount_converted"] > 10)
]

# Total number of users in the dataset
total_users = users["ID"].nunique()

# Final KPI
kpi_users = first_card_over10["USER_ID"].nunique()
kpi_percentage = round((kpi_users / total_users) * 100, 2)

print(f" - {kpi_users} users had their first transaction as a successful card payment over $10 USD.")
print(f" - That is {kpi_percentage}% of all users.")
```

```
USD transactions: 31542
 - 447 users had their first transaction as a successful card payment over $10 USD.
 - That is 4.34% of all users.
```

```python
# Sanity checks

# 1. Total number of transactions in dataset
total_transactions = len(transactions)

# 2. Number of unique users who performed any transaction
unique_users_in_transactions = transactions["USER_ID"].nunique()

# 3. Number of users from users.csv
total_users_in_users_csv = users["ID"].nunique()

# 4. Number of users whose first USD transaction was a completed card payment over $10
kpi_users = first_card_over10["USER_ID"].nunique()

# Print summary
print("=== SANITY CHECK SUMMARY ===")
print(f"Total transactions: {total_transactions}")
print(f"Unique users in transactions.csv: {unique_users_in_transactions}")
print(f"Total users in users.csv: {total_users_in_users_csv}")
print(f"Users with first transaction being a completed card payment over $10 USD: {kpi_users}")
```

```
=== SANITY CHECK SUMMARY ===
Total transactions: 688651
Unique users in transactions.csv: 8021
Total users in users.csv: 10300
Users with first transaction being a completed card payment over $10 USD: 447
```

A2. Explanation (Transaction Success Rate KPI - First card payment over 10 USD)

1. Data Integrity Validation Checks - Check if data is reliable and free of issues

   Before diving into the actual Transaction Success Rate KPI analysis it was important to ensure that we were dealing with reliable data and free of issues. Therefore a data integrity check was performed on the transactions.csv file to identify issues that could affect the validity of downstream analyses. This check focused on seven key aspects:

   - Missing Dates: we confirmed that all transactions had a valid "CREATE_DATE", ensuring we can trust chronological filters like identifying the user's first transaction. (Transactions missing a date: 0)
   - Data Format: all timestamps were successfully converted to datetime format, enabling accurate sorting and time-based analysis.
   - User Association: no transactions were missing a "USER_ID", meaning every transaction could be traced back to a user.
   - Duplicates: We confirmed that there were no duplicated transaction IDs, which means every transaction was uniquely recorded.
   - Zero-Value Transactions: We found 12,023 transactions with a value of 0. These were flagged yet, they do not materially impact high-level metrics like KPI calculations. We chose to exclude them where appropriate since these represent 1.75% of all transactions (total of 688651 transactions).
   - Negative Values: None of the transactions had negative amounts, avoiding the need for adjustments.
   - Currency Mapping: All currencies used in transactions were present in the currency_details.csv file, ensuring accurate conversion from smallest currency units (e.g. cents) to standard units (e.g. dollars or euros)

   We can therefore conclude that the dataset passed all major integrity checks and is suitable for use-level and financial analysis without further repair or pre-processing.

2. Calculating the Transaction Success Rate KPI of users first card transaction above 10 USD.

   To calculate how many users had their first transsaction as a successful card payment over 10 USD, we took the following steps:

   - Focused on USD transactions only, avoiding exchange rate complexities, which aligned with the task that explicitly referenced USD.
   - Converted amounts from cents to dollars using the exponent from currency_details.csv, so that we could evaluate if the transaction exceeded 10 USD.
   - Identified each user's first transaction by sorting transactions by user and timestamp.
   - Filtered for the first transaction that was completed, card payments and above 10 USD.

From this it can be concluded that a total of **447** users had a first transaction recorded as a successful card payment over 10 USD. Which represents 4.34% of all users in the system (10300 total), or 5.57% when considering only users who performed at least one transaction (8021 total)

B - Code (Comparing behavior patterns in fraudsters vs non-fradusters & find top 5 likely fraudsters)

```
# Task B

# Tag users as known fraudsters
users["is_fraud"] = users["ID"].isin(fraudsters["user_id"])

# Make sure 'CURRENCY' is str
transactions["CURRENCY"] = transactions["CURRENCY"].astype(str)

# Merge transactions with currency exponent
tx = transactions.merge(currency_details[["currency", "exponent"]],
                        left_on="CURRENCY", right_on="currency", how="left")

# Convert amount to major units (e.g., cents to dollars)
tx["amount_major"] = tx["AMOUNT"] / (10 ** tx["exponent"])

# Get FX rates to USD only
fx_to_usd = fx_rates[fx_rates["ccy"] == "USD"]

# Merge FX rate into transactions
tx = tx.merge(fx_to_usd[["base_ccy", "rate"]],
              left_on="CURRENCY", right_on="base_ccy", how="left")

# Convert to USD (or keep same if already in USD)
tx["amount_usd"] = tx["amount_major"] * tx["rate"]
tx.loc[tx["CURRENCY"] == "USD", "amount_usd"] = tx.loc[tx["CURRENCY"] == "USD", "amount_major"]

# Merge with user details
tx_users = tx.merge(users, left_on="USER_ID", right_on="ID", how="left")

# Compute user-level transaction features (in USD)
user_stats = tx_users.groupby("USER_ID").agg(
    num_tx=("ID_x", "count"),
    total_amount=("amount_usd", "sum"),
    avg_amount=("amount_usd", "mean"),
    num_card_payments=("TYPE", lambda x: (x == "CARD_PAYMENT").sum()),
    num_p2p=("TYPE", lambda x: (x == "P2P").sum()),
    num_declined=("STATE_x", lambda x: (x == "DECLINED").sum()),
    num_completed=("STATE_x", lambda x: (x == "COMPLETED").sum()),
).reset_index()
```

```python
# Add fraud label
user_stats["is_fraud"] = user_stats["USER_ID"].isin(fraudsters["user_id"])

# Split users
fraudsters_stats = user_stats[user_stats["is_fraud"] == True]
nonfraud_stats = user_stats[user_stats["is_fraud"] == False]

# Compare average metrics
features = ["num_tx", "total_amount", "avg_amount",
            "num_card_payments", "num_p2p",
            "num_declined", "num_completed"]

fraud_means = fraudsters_stats[features].mean().rename("Fraudsters Avg")
nonfraud_means = nonfraud_stats[features].mean().rename("Non-Fraudsters Avg")

# Create comparison table
comparison = pd.concat([fraud_means, nonfraud_means], axis=1)

# Add % difference
comparison["% Difference"] = 100 * ((comparison["Fraudsters Avg"] - comparison["Non-Fraudsters Avg"]) / comparison["Non-Fraudsters Avg"])

# Make row labels more readable
comparison.index = [
    "Total Transactions per User",
    "Total USD Spent per User",
    "Average USD per Transaction",
    "Card Payments per User",
    "P2P Payments per User",
    "Declined Transactions per User",
    "Completed Transactions per User"
]

# Round for clarity
comparison = comparison.round(2)

# Display
display(comparison)
```

	Fraudsters Avg	Non-Fraudsters Avg	% Difference
Total Transactions per User	48.64	87.30	-44.28
Total USD Spent per User	10609.92	5336.41	98.82
Average USD per Transaction	310.32	79.82	288.78
Card Payments per User	22.91	55.65	-58.84
P2P Payments per User	1.46	7.24	-79.86
Declined Transactions per User	7.31	5.63	30.00
Completed Transactions per User	35.06	74.34	-52.83

```python
# Task B — Identifying top 5 suspected Fraudsters

# Step 1: Filter out known fraudsters
unknown_users = user_stats[user_stats["is_fraud"] == False].copy()

# Step 2: Apply updated fraud behavior flags (now consistent with analysis table)
unknown_users["flag_high_avg_amount"] = unknown_users["avg_amount"] > 300
unknown_users["flag_high_declined"] = unknown_users["num_declined"] > 7
unknown_users["flag_low_card_tx"] = unknown_users["num_card_payments"] < 30
unknown_users["flag_low_p2p"] = unknown_users["num_p2p"] < 3
unknown_users["flag_low_completion_ratio"] = (unknown_users["num_completed"] / unknown_users["num_tx"]) < 0.5
unknown_users["flag_low_total_tx"] = unknown_users["num_tx"] < 50

# Step 3: Compute fraud score (sum of all binary flags)
flag_cols = [col for col in unknown_users.columns if col.startswith("flag_")]
unknown_users["fraud_score"] = unknown_users[flag_cols].sum(axis=1)

# Step 4: Select top 5 users with highest fraud score
top5_flags = unknown_users.sort_values("fraud_score", ascending=False).head(5)

# Step 5: Define columns for display
cols_to_display = [
    "USER_ID", "fraud_score"
] + flag_cols + [
    "avg_amount", "num_tx", "num_declined", "num_card_payments",
    "num_p2p", "num_completed"
]

# Transpose for vertical presentation
top_users_transposed = top5_flags[cols_to_display].set_index("USER_ID").T

# Add sanity check for known fraudsters
is_known_fraudster = top_users_transposed.columns.isin(fraudsters["user_id"])
top_users_transposed.loc["Known Fraudster?"] = ["Yes" if val else "No" for val in is_known_fraudster]

# Move sanity check row to the top
new_index_order = ["Known Fraudster?"] + [i for i in top_users_transposed.index if i != "Known Fraudster?"]
top_users_transposed = top_users_transposed.loc[new_index_order]
```

```python
# Make row labels readable
readable_names = {
    "fraud_score": "Fraud Risk Score",
    "flag_high_avg_amount": "High Avg Transaction Amount",
    "flag_high_declined": "Many Declined Transactions",
    "flag_low_card_tx": "Low Card Transactions",
    "flag_low_p2p": "Low P2P Transactions",
    "flag_low_completion_ratio": "Low Completion Rate",
    "flag_low_total_tx": "Low Total Transactions",
    "avg_amount": "Average USD per Transaction",
    "num_tx": "Total Transactions",
    "num_declined": "Declined Transactions",
    "num_card_payments": "Card Payments",
    "num_p2p": "P2P Payments",
    "num_completed": "Completed Transactions",
    "Known Fraudster?": "Known Fraudster?"
}

top_users_transposed = top_users_transposed.rename(index=readable_names)

# Display result
display(top_users_transposed)
```

USER_ID	450dc7bf-d8d2-4989-97b9-b54669255ef0	febeeef3-19a9-4e9b-8f57-4398f400f3f8	3011a01f-e9b0-477e-9791-9aa5ae6abd86	30e56a5d-ef22-4ee1-93f7-8ffbf134fb9e	df5112c5-11fe-4a59-9e12-cbbab7873b43
Known Fraudster?	No	No	No	No	No
Fraud Risk Score	6	5	5	5	5
High Avg Transaction Amount	True	False	True	False	False
Many Declined Transactions	True	True	False	True	True
Low Card Transactions	True	True	True	True	True
Low P2P Transactions	True	True	True	True	True
Low Completion Rate	True	True	True	True	True
Low Total Transactions	True	True	True	True	True
Average USD per Transaction	382.042575	10.65612	403.672359	44.289686	2.511492
Total Transactions	27	33	12	30	8
Declined Transactions	10	8	7	23	8
Card Payments	6	13	9	29	8
P2P Payments	0	0	0	0	0
Completed Transactions	13	5	5	5	0

B - Explanation (Comparing behavior patterns in fraudsters vs non-fradusters & find top 5 likely fraudsters)

1. Identifying how a fraudster user behaves

   In order to identify the 5 fraudsters, we first had to understand if we could spot any differences in behavior between these outlaws and “normal” users. This understanding would not only help us validate the characteristics of existing fraudsters but also inform our strategy for identifying potential new fraud cases. To do this, we chose to calculate and compare key transaction metrics per user, such as the number of transactions, average amount, payment method, and success vs. decline rates.

   It was early spotted that our data set contains multiple currencies, each one with different exponents and exchange rates. Therefore it would never be fair to compare this raw data directly, as it would be inconsistent and misleading (example: 200 USD can't be compared to 200 GBP). To ensure a fair and accurate comparison between all users, the following was done:

   - First converted every transaction to major units (example: cents to dollars)
   - Then we applied a foreign exchange rate to USD, using the fx_rates.csv file (converted all non-USD transactions into USD).
   - For transactions already in USD, we left the values the same.

   With the above, we then grouped all transactions by user and averaged the following behaviors per user:

   - Total number of transactions

- Total and average transaction amount (all converted to USD)
- Number of card payments
- Number of peer-to-peer (P2P) payments
- Number of declined and completed transactions

This was done for fraudsters and non-fraudsters. The difference in results was then converted to a percentage so that we could easily distinct patterns in behavior. The key findings were the following:

- Fraudsters spend much more per transaction when compared to non-fradusters (+289%)
- Despite fewer transactions, fraudsters move more money overall (+99%)
- Fraudsters avoid using card payments (-59%)
- Fraudsters rarely engage in peer to peer transactions (-80%)
- Fraudsters have more declined transactions per user (+30%)
- Fraudsters have less success rate in completing transactions (-53%)
- Despite moving more money, fraudsters do fewer transactions overall (-44%)

2. Finding 5 most likely fraudsters that are not in our fraudsters database

The analysis carried out in the previous section allowed us to learn the *modus-operandi* of a fraudster, using our fraudsters.csv file. So, it was now crucial to create an interpretable scoring system that would allow us to flag the top 5 suspicious users that are not yet tagged in our system, and that follow the same behavioral pattern.

To achieve this, a scoring logic was created using the average results we got from the table below:

	Fraudsters Avg	Non-Fraudsters Avg	% Difference
Total Transactions per User	48.64	87.30	-44.28
Total USD Spent per User	10609.92	5336.41	98.82
Average USD per Transaction	310.32	79.82	288.78
Card Payments per User	22.91	55.65	-58.84
P2P Payments per User	1.46	7.24	-79.86
Declined Transactions per User	7.31	5.63	30.00
Completed Transactions per User	35.06	74.34	-52.83

Based on the above a fraud score was created, from 0 to 6. Users that got the highest score will be flagged. The scoring followed that results from the table above, and was created assuming the following:

- Average transaction amount > 300 USD (average per fraudster is 310 USD)
- Declined transactions > 7 (average per fraudster was 7.31)

- Card payments < 30 (average per fraudster is 23 but for non-fraudster was 56)
- Number of P2P transactions < 3 (average per fraudster is 1.5 but for non-fraudster is 5.63)
- Completion rate <0.5 (completed transactions/total transactions)
- Total transaction < 50 (average per fraudster is 35 but for non-fraudster is 74)

It's worth mentioning that the scoring method detailed above can be fine tuned and adjusted to be more or less aggressive. A sanity check was also completed where we made sure that the users we were analysing were not in our known fraudsters database. From that, with each user starting with a fraud score = 0, true and false flags were created, if it followed or did not follow the scoring metric. For a true result +1 was added. The higher the score, the more suspicious this user was.

A final table was then created (see below) with users scoring values in between 6 and 5, which is considered very high. All of them followed a suspicious behavior: aggressive patterns (declines, incomplete flows), avoidance of organic features (P2P) and either very high or very low financial exposure (depending on the strategy used)

USER_ID	450dc7bf-d8d2-4989-97b9-b54669255ef0	febeeef3-19a9-4e9b-8f57-4398f400f3f8	3011a01f-e9b0-477e-9791-9aa5ae6abd86	30e56a5d-ef22-4ee1-93f7-8ffbf134fb9e	df5112c5-11fe-4a59-9e12-cbbab7873b43
Known Fraudster?	No	No	No	No	No
Fraud Risk Score	6	5	5	5	5
High Avg Transaction Amount	True	False	True	False	False
Many Declined Transactions	True	True	False	True	True
Low Card Transactions	True	True	True	True	True
Low P2P Transactions	True	True	True	True	True
Low Completion Rate	True	True	True	True	True
Low Total Transactions	True	True	True	True	True
Average USD per Transaction	382.042575	10.65612	403.672359	44.289686	2.511492
Total Transactions	27	33	12	30	8
Declined Transactions	10	8	7	23	8
Card Payments	6	13	9	29	8
P2P Payments	0	0	0	0	0
Completed Transactions	13	5	5	5	0

We therefore advise for the following users to be flagged and monitored closer for signs of continuous suspicious activity:

- **user ID**: 450dc7bf-d8d2-4989-97b9-b54669255ef0
- **user ID:** febeeef3-19a9-4e9b-8f57-4398f400f3f8
- **user ID:** 3011a01f-e9b0-477e-9791-9aa5ae6abd86
- **user ID:** 30e56a5d-ef22-4ee1-93f7-8ffbf134fb9e
- **user ID:** df5112c5-11fe-4a59-9e12-cbbab7873b43

B* - CODE (Future work & suggestions) - optional

```
# Task B
# Future work & suggestions (not required but still valuable isnight to consider)

import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import matplotlib.ticker as mtick
import pandas as pd

# Ensure datetime format
tx_users["CREATED_DATE_x"] = pd.to_datetime(tx_users["CREATED_DATE_x"])
tx_users["week"] = tx_users["CREATED_DATE_x"].dt.to_period("W").dt.start_time

# Mark each transaction as fraud or not
tx_users["is_fraud_tx"] = tx_users["USER_ID"].isin(fraudsters["user_id"])

# Group by week
weekly = tx_users.groupby("week").agg(
    total_tx=("ID_x", "count"),
    fraud_tx=("is_fraud_tx", "sum")
).reset_index()

# Compute fraud rate
weekly["fraud_rate"] = weekly["fraud_tx"] / weekly["total_tx"]

# Smooth fraud rate with a rolling average
weekly["fraud_rate_smooth"] = weekly["fraud_rate"].rolling(4).mean()

# Optional: remove low-volume weeks to improve clarity
weekly = weekly[weekly["total_tx"] > 200]

# Plot
fig, ax1 = plt.subplots(figsize=(14, 6))
```

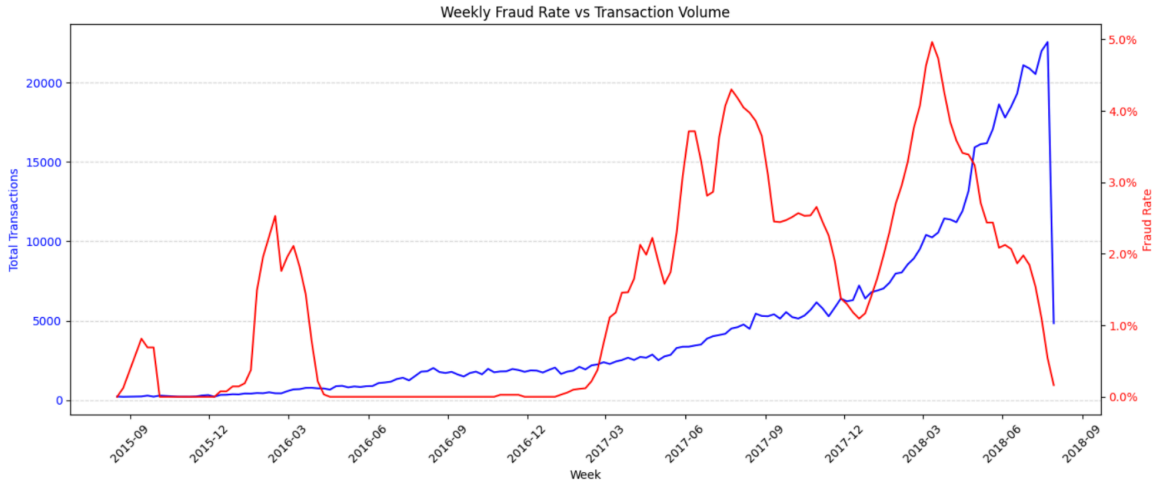
```
# Left axis: total transactions
ax1.plot(weekly["week"], weekly["total_tx"], color="blue", label="Total Transactions")
ax1.set_ylabel("Total Transactions", color="blue")
ax1.tick_params(axis="y", labelcolor="blue")
ax1.set_xlabel("Week")
ax1.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
ax1.xaxis.set_major_locator(mdates.MonthLocator(interval=3))
plt.xticks(rotation=45)
ax1.grid(True, axis="y", linestyle="--", alpha=0.5)

# Right axis: fraud rate
ax2 = ax1.twinx()
ax2.plot(weekly["week"], weekly["fraud_rate_smooth"], color="red", label="Fraud Rate (Smoothed)")
ax2.set_ylabel("Fraud Rate", color="red")
ax2.tick_params(axis="y", labelcolor="red")
ax2.yaxis.set_major_formatter(mtick.PercentFormatter(1.0))

# Title and layout
plt.title("Weekly Fraud Rate vs Transaction Volume")
fig.tight_layout()
plt.show()
```

B* - Explanation (Future work & suggestions) - optional

As an extra and optional task, we decided to dig a bit deeper and try to find interesting insights for the FC team on how the fraud rate (fraudulent transactions/total transactions) change with the number of total transactions over time. The following can be concluded from the table below:



- **Stable growth:** total transaction volume increased consistently from 2015 until mid 2018, indicating strong user and platform growth.
- **Fraud peaks:** Noticeable fraud rate peaks occurred in early 2016, mid-2017, and March 2018.
- **Drop in fraud rate post march 2018:** despite transactions continued to grow, fraud rate sharply declined after March 2018 and remained lower. Potentially suggesting the following:
    - improved fraud detection or intervention (well done team!)
    - fraudster adapting and becoming harder to detect.
    - policy changes (maybe related to new controls, KYC enforcement etc)

We advise the following:

- **Investigate the big drop in fraud rate in 2018-Q1**: check if this was due to a change in platform rules, new security features, or if fraudsters simply changed tactics.
- **Check blind spots:** Confirm that the lower fraud rate reflects actual reduction in fraud and not a decrease in detection effectiveness.
- **Deploy alerts for unusual spikes:** Implement alerts for sudden weekly increases in fraud rate, enabling faster reaction and deeper investigation when needed.