rev_home_task_2

June 20, 2025

1 Revolut Home task - Helping FC Squad

1.1 Introduction

[2]:
```
from sqlalchemy import create_engine, Column, String, Integer, BigInteger,
    Boolean, Date, TIMESTAMP, Float, ForeignKey
from sqlalchemy.dialects.postgresql import UUID
from sqlalchemy.orm import declarative_base
from sqlalchemy.orm import sessionmaker
import uuid
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```python
from sklearn.preprocessing import MinMaxScaler
```

[3]:
```python
Base = declarative_base()
pd.set_option('display.width', 1000)
pd.set_option('display.max_rows', None)
```

1.1.1 Setting up db scheme

[4]:
```python
class Transaction(Base):
    __tablename__ = 'transactions'
    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid.uuid4)
    currency = Column(String(3), nullable=False)
    amount = Column(BigInteger, nullable=False)
    state = Column(String(25), nullable=False)
    created_date = Column(TIMESTAMP, nullable=False)
    merchant_category = Column(String(100))
    merchant_country = Column(String(3))
    entry_method = Column(String(4), nullable=False)
    user_id = Column(UUID(as_uuid=True), nullable=False)
    type = Column(String(20), nullable=False)
    source = Column(String(20), nullable=False)

class User(Base):
    __tablename__ = 'users'
    id = Column(UUID(as_uuid=True), primary_key=True, default=uuid.uuid4)
    has_email = Column(Boolean, nullable=False)
    phone_country = Column(String(300))
    terms_version = Column(Date, nullable=False)
    created_date = Column(TIMESTAMP, nullable=False)
    state = Column(String(25), nullable=False)
    country = Column(String(2))
    birth_year = Column(Integer)
    kyc = Column(String(20))
    failed_sign_in_attempts = Column(Integer)

class FxRate(Base):
    __tablename__ = 'fx_rates'
    base_ccy = Column(String(3), primary_key=True)
    ccy = Column(String(10), primary_key=True)
    rate = Column(Float)

class CurrencyDetail(Base):
    __tablename__ = 'currency_details'
    ccy = Column(String(10), primary_key=True)
    iso_code = Column(Integer)
    exponent = Column(Integer)
    is_crypto = Column(Boolean, nullable=False)
```

```python
class Fraudster(Base):
    __tablename__ = 'fraudsters'
    user_id = Column(UUID(as_uuid=True), primary_key=True, default=uuid.uuid4)
```

1.1.2 Linking to db instance running locally

[5]:
```python
DATABASE_URL = "postgresql+psycopg2://postgres:admin@localhost:5432/postgres"

engine = create_engine(DATABASE_URL)
Session = sessionmaker(bind=engine)
session = Session()
```

[6]:
```python
Base.metadata.create_all(engine)
```

1.2 Data preprocessing

1.2.1 Cleaning and loading Users Data

[7]:
```python
# checking for missing values in Users
df_users = pd.read_csv("data/users.csv")
df_users.isnull().sum()
```

[7]:
```
Unnamed: 0                0
FAILED_SIGN_IN_ATTEMPTS   0
KYC                       0
BIRTH_YEAR                0
COUNTRY                   0
STATE                     0
CREATED_DATE              0
TERMS_VERSION          1710
PHONE_COUNTRY             0
HAS_EMAIL                 0
ID                        0
dtype: int64
```

[8]:
```python
# check if any duplicate data
df_users.duplicated().value_counts()
```

[8]:
```
False    10300
Name: count, dtype: int64
```

[9]:
```python
def load_users_data():
    df_users = pd.read_csv("data/users.csv")

    # replance missing TERMS_VERSION values with None to enable upload in db
    df_users['TERMS_VERSION'] = pd.to_datetime(df_users['TERMS_VERSION'],␣
↪errors='coerce').dt.date
```

```
    df_users['TERMS_VERSION'] = df_users['TERMS_VERSION'].apply(lambda x: x if␣
↪pd.notnull(x) else None)

    for _, row in df_users.iterrows():
        user = User(
            id=row['ID'],
            has_email=row['HAS_EMAIL'],
            phone_country=row.get('PHONE_COUNTRY'),
            terms_version=row.get('TERMS_VERSION'),
            created_date=row['CREATED_DATE'],
            state=row['STATE'],
            country=row.get('COUNTRY'),
            birth_year=row.get('BIRTH_YEAR'),
            kyc=row.get('KYC'),
            failed_sign_in_attempts=row.get('FAILED_SIGN_IN_ATTEMPTS')
        )
        session.add(user)
    session.commit()
```

[10]:
```
# helper to clear users data
session.query(User).delete()
session.commit()
print("All users deleted.")
```

```
All users deleted.
```

[11]:
```
load_users_data()
```

[12]:
```
df_users = pd.read_sql("SELECT * FROM users LIMIT 10;", con=engine)
print(df_users)
```

```
                                     id  has_email   phone_country terms_version
created_date   state country  birth_year     kyc  failed_sign_in_attempts
0  19e904b0-6b2e-41c1-8e86-75178650b515       True              CH          None
2017-05-20 12:20:37.357  ACTIVE      CH        1982  PASSED
0
1  1161adca-2a97-45c5-9931-ac394780addc       True              CZ    2018-09-20
2018-02-03 19:00:02.202  ACTIVE      GB        1990  PASSED
0
2  bc336b6f-7538-45f2-9b47-74bb8cd87bc5       True              FR    2018-09-20
2018-07-26 12:19:46.276  ACTIVE      FR        1992  PASSED
0
3  8f7cb96b-2284-4206-b72b-57b9a6e9dc10       True              MT    2018-05-25
2018-05-02 08:54:34.501  ACTIVE      MT        1990  PASSED
0
4  3db93982-c41d-4fe6-9d51-3e76f126dcc1       True              IE    2018-05-25
2016-08-02 10:05:02.181  ACTIVE      IE        1988  PASSED
0
```

```
5   8ddaadbb-f558-468a-bb33-cfbd0adec6ca       True              LT    2018-05-25
2018-02-03 17:24:15.951  ACTIVE      LT         1978  PASSED
0
6   1872820f-e3ac-4c02-bdc7-727897b60043       True  GB||JE||IM||GG    2018-05-25
2017-08-06 07:33:33.341  ACTIVE      GB         1971  PASSED
0
7   545ff94d-66f8-4bea-b398-84425fb2301e       True  GB||JE||IM||GG    2018-01-01
2017-03-07 10:18:59.427  ACTIVE      GB         1982  PASSED
0
8   10376f1a-a28a-4885-8daa-c8ca496026bb       True              ES    2018-09-20
2018-05-31 04:41:24.672  ACTIVE      ES         1973  PASSED
0
9   fd308db7-0753-4377-879f-6ecf2af14e4f       True              FR    2018-05-25
2018-06-01 17:24:23.852  ACTIVE      FR         1986  PASSED
0
```

[13]:
```python
df_users_count = pd.read_sql("SELECT COUNT(*) FROM users;", con=engine)
print(df_users_count)
```

```
   count
0  10300
```

1.2.2 Cleaning and loading Transactions Data

[14]:
```python
# checking for missing values in Transactions
df_tx = pd.read_csv('data/transactions.csv')
df_tx.isnull().sum()
```

[14]:
```
Unnamed: 0                0
CURRENCY                  0
AMOUNT                    0
STATE                     0
CREATED_DATE              0
MERCHANT_CATEGORY    465586
MERCHANT_COUNTRY     205578
ENTRY_METHOD              0
USER_ID                   0
TYPE                      0
SOURCE                    0
ID                        0
dtype: int64
```

[15]:
```python
print(df_tx[df_tx['CURRENCY'].str.len() > 3])
```

```
Empty DataFrame
Columns: [Unnamed: 0, CURRENCY, AMOUNT, STATE, CREATED_DATE, MERCHANT_CATEGORY,
MERCHANT_COUNTRY, ENTRY_METHOD, USER_ID, TYPE, SOURCE, ID]
Index: []
```

[16]:
```python
# identifiying merchant countries not following convention
print(df_tx[df_tx['MERCHANT_COUNTRY'].str.len() > 3]['MERCHANT_COUNTRY'])
```

```
1889    %20%20%20%20%20%20SERRIS%20%20%20%20%20%20%20%…
1890                        20BELFAST%20%20%20%20%20%20%20GBR
1891                                            E%20%20%20%20%20FL
1898                        PRAHA%201%20%20%20%20%20%20%20CZE
1899                        RIS%20%20%20%20%20%20%20%20%20FRA
1901                                             ADELLA%20DE%20ESP
1902                                  2016%20%20%20%20%20%20FRA
1903                20%20%20PARIS%207%20%20%20%20%20%20%20FRA
1984                      PRAHA%20%20%20%20%20%20%20%20%20CZE
1985                                                 %20-07A%20EGY
1986                             20PHNOM%20PENH%20%20%20%20KHM
1987                               201%20%20%20%20%20%20%20CZE
1992                     0%20SERRIS%20%20%20%20%20%20%20%20FRA
1993                        0%20VIC%20SUR%20AI%20%20%20%20FRA
1995                %20%20PARIS%20%20%20%20%20%20%20%20%20FRA
1996                                              0DE%20SAN%20ESP
1997                                                          I FL
4086    0%%32%30%%32%30%32696%%32%30%%32%30%%32%30SPRI…
Name: MERCHANT_COUNTRY, dtype: object
```

[17]:
```python
def load_transactions_data():
    df_tx = pd.read_csv("data/transactions.csv")

    # replace NaN with None to enable data upload in db
    df_tx = df_tx.where(pd.notnull(df_tx), None)

    # convert UUID fields
    df_tx['ID'] = df_tx['ID'].apply(lambda x: uuid.UUID(x) if x else None)
    df_tx['USER_ID'] = df_tx['USER_ID'].apply(lambda x: uuid.UUID(x) if x else␣
 ↪None)

    # trimming countries that exceed 3 characters
    df_tx['MERCHANT_COUNTRY'] = df_tx['MERCHANT_COUNTRY'].astype(str).str[-3:]

    for _, row in df_tx.iterrows():
        tx = Transaction(
            id=row['ID'],
            currency=row['CURRENCY'],
            amount=row['AMOUNT'],
            state=row['STATE'],
            created_date=row['CREATED_DATE'],
            merchant_category=row.get('MERCHANT_CATEGORY'),
            merchant_country=row.get('MERCHANT_COUNTRY'),
            entry_method=row['ENTRY_METHOD'],
```

```
            user_id=row['USER_ID'],
            type=row['TYPE'],
            source=row['SOURCE']
        )
        session.add(tx)
    session.commit()
```

[18]:
```
# helper to clear transactions data
session.query(Transaction).delete()
session.commit()
print("All transactions deleted.")
```

```
All transactions deleted.
```

[19]:
```
load_transactions_data()
```

[20]:
```
df_tx_10 = pd.read_sql("SELECT * FROM transactions LIMIT 10;", con=engine)
print(df_tx_10)
```

```
                                     id currency  amount      state
created_date merchant_category merchant_country entry_method
user_id          type source
0  f40e4635-ae8e-42c6-a620-a69f631b0f8a      PLN    1226  COMPLETED 2018-07-23
17:11:29.228              None              POL         cont
456bb518-648b-41b1-bdba-91f8fbf9cfe3  CARD_PAYMENT   GAIA
1  d942648d-f386-4cee-be0c-7813588fad91      EUR    1000  COMPLETED 2018-07-24
07:33:24.004              None              one         misc
884981e0-8ccd-4169-8184-c59e68888b83          TOPUP  LIMOS
2  be9d2c47-2df4-4ec6-9024-c1b9b3c55c45      EUR    1397  COMPLETED 2018-07-23
13:31:47.041              None              AUT         cont
f5288349-94cc-450e-915a-79f9d1384885  CARD_PAYMENT   GAIA
3  9077dad1-3e32-4836-beda-db76cab65c8d      CHF     810  COMPLETED 2018-07-23
09:04:39.836              cafe              CHE         cont
cb3126af-c20b-4adb-9286-335c1902251c  CARD_PAYMENT   GAIA
4  ff6a286c-0558-462c-8408-41bca5849ea2      PLN     439  COMPLETED 2018-07-23
19:26:16.486              None              POL         cont
2e9b1d30-801e-45f1-b4c3-649b7a835ed7  CARD_PAYMENT   GAIA
5  6354aa07-273e-475c-838a-7cbb8a70bc85      PLN     350  COMPLETED 2018-07-23
20:42:36.986              None              POL         cont
2e9b1d30-801e-45f1-b4c3-649b7a835ed7  CARD_PAYMENT   GAIA
6  a28bc89d-f4d0-4f38-a163-c4c60b95102a      PLN     339  COMPLETED 2018-07-23
20:02:37.845              None              POL         cont
2e9b1d30-801e-45f1-b4c3-649b7a835ed7  CARD_PAYMENT   GAIA
7  9b1be374-febd-4714-a50f-338321c705a2      PLN    1199  COMPLETED 2018-07-23
14:15:12.636              None              POL         cont
2e9b1d30-801e-45f1-b4c3-649b7a835ed7  CARD_PAYMENT   GAIA
8  bbeae10f-483d-4149-8fa3-d79e07d66f1b      EUR     799  COMPLETED 2018-07-23
14:51:06.783              None              GBR         manu
```

```
0453fe78-b62b-4727-8f79-840ef192028a  CARD_PAYMENT   GAIA
9  2e87b3ef-073a-4245-87c1-a762c2d1d9c1      EUR     350  COMPLETED 2018-07-23
17:47:06.347              None              ESP         cont  440eb1eb-
bdad-4589-9b77-35a08480171a  CARD_PAYMENT   GAIA
```

[21]:
```
df_tx_count = pd.read_sql("SELECT COUNT(*) FROM transactions;", con=engine)
print(df_tx_count)
```

```
    count
0  688651
```

1.2.3 Cleaning and loading FX rates Data

[22]:
```
# checking for missing values in Transactions
df_fx = pd.read_csv('data/fx_rates.csv')
df_fx.isnull().sum()
```

[22]:
```
base_ccy    0
ccy         0
rate        0
dtype: int64
```

[23]:
```
def load_fx_rates_data():
    df_fx = pd.read_csv("data/fx_rates.csv")

    # Replace NaNs with None
    df_fx = df_fx.where(pd.notnull(df_fx), None)

    # Clean string lengths (if needed)
    df_fx['base_ccy'] = df_fx['base_ccy'].astype(str).str[:3].str.upper()
    df_fx['ccy'] = df_fx['ccy'].astype(str).str[:10].str.upper()

    for _, row in df_fx.iterrows():
        fx = FxRate(
            base_ccy=row['base_ccy'],
            ccy=row['ccy'],
            rate=row['rate']
        )
        session.add(fx)
    session.commit()
```

[24]:
```
# helper to clear fx rates data
session.query(FxRate).delete()
session.commit()
print("All fx rates deleted.")
```

```
All fx rates deleted.
```

[25]:
```
load_fx_rates_data()
```

[26]:
```
df_fx_10 = pd.read_sql("SELECT * FROM fx_rates LIMIT 10;", con=engine)
print(df_fx_10)
```

```
  base_ccy  ccy         rate
0      EUR  AED     0.239336
1      EUR  AUD     0.639595
2      EUR  BTC  6617.495728
3      EUR  CAD     0.662312
4      EUR  CHF     0.871317
5      EUR  CZK     0.045900
6      EUR  DKK     0.141097
7      EUR  ETH   370.055188
8      EUR  GBP     1.128280
9      EUR  HKD     0.115652
```

[27]:
```
df_fx_count = pd.read_sql("SELECT COUNT(*) FROM fx_rates;", con=engine)
print(df_fx_count)
```

```
   count
0     84
```

1.2.4 Cleaning and loading Currency Detail Data

[28]:
```
# checking for missing values in Transactions
df_ccy = pd.read_csv('data/currency_details.csv')
df_ccy.isnull().sum()
```

[28]:
```
currency      0
iso_code     39
exponent     24
is_crypto     0
dtype: int64
```

[29]:
```
# indentifying float values that need to be converted to int
print(df_ccy.dtypes)
```

```
currency      object
iso_code     float64
exponent     float64
is_crypto       bool
dtype: object
```

[30]:
```
def load_currency_details_data():
    df_ccy = pd.read_csv("data/currency_details.csv")

    # Replace NaNs with None
    df_ccy = df_ccy.where(pd.notnull(df_ccy), None)

    # Ensure correct types
```

```
    df_ccy['is_crypto'] = df_ccy['is_crypto'].astype(bool)
    df_ccy['iso_code'] = df_ccy['is_crypto'].astype(int)
    df_ccy['exponent'] = df_ccy['is_crypto'].astype(int)


    for _, row in df_ccy.iterrows():
        detail = CurrencyDetail(
            ccy=row['currency'],
            iso_code=row.get('iso_code'),
            exponent=row.get('exponent'),
            is_crypto=row['is_crypto']
        )
        session.add(detail)
    session.commit()
```

[31]:
```
# helper to clear ccy data
session.query(CurrencyDetail).delete()
session.commit()
print("All currency data deleted.")
```

```
All currency data deleted.
```

[32]:
```
load_currency_details_data()
```

[33]:
```
df_ccy_10 = pd.read_sql("SELECT * FROM currency_details LIMIT 10;", con=engine)
print(df_ccy_10)
```

```
   ccy  iso_code  exponent  is_crypto
0  AED         0         0      False
1  AFN         0         0      False
2  ALL         0         0      False
3  AMD         0         0      False
4  ANG         0         0      False
5  AOA         0         0      False
6  ARS         0         0      False
7  AUD         0         0      False
8  AWG         0         0      False
9  AZN         0         0      False
```

[34]:
```
df_ccy_count = pd.read_sql("SELECT COUNT(*) FROM currency_details;", con=engine)
print(df_ccy_count)
```

```
   count
0    208
```

1.2.5 Loading fraudsters IDs

[35]:
```
# checking for missing values in Transactions
df_ccy = pd.read_csv('data/fraudsters.csv')
df_ccy.isnull().sum()
```

[35]:
```
Unnamed: 0    0
user_id       0
dtype: int64
```

[36]:
```
def load_fraudsters_data():
    df_ccy = pd.read_csv("data/fraudsters.csv")

    for _, row in df_ccy.iterrows():
        detail = Fraudster(
            user_id=row['user_id']
        )
        session.add(detail)
    session.commit()
```

[37]:
```
# helper to clear ccy data
session.query(Fraudster).delete()
session.commit()
print("All fraudsters data deleted.")
```

```
All fraudsters data deleted.
```

[38]:
```
load_fraudsters_data()
```

1.3 Solutions

1.4 Task A.1

Task is to provide SQL statement to get a table view that shows the customer id, customer country and transaction amount on the GAIA transaction processing sever.

Customer ID & Transaction amount can be easily extracted from transactions dataset, while customer country is available only in users dataset. So we'll need to merge both datasets on Customer Id to get all informatiion in one view. Let's get to it.

[39]:
```
tx_dash_monitor = pd.read_sql("""
    SELECT
        USER_ID as customer_id,
        COUNTRY as customer_country,
        AMOUNT as transaction_amount
    FROM transactions as tx
    LEFT JOIN users as us ON tx.USER_ID = us.ID
    WHERE SOURCE='GAIA';
    """,
    con=engine)
```

```
tx_dash_monitor.head()
```

[39]:

	customer_id	customer_country	transaction_amount
0	456bb518-648b-41b1-bdba-91f8fbf9cfe3	PL	1226
1	f5288349-94cc-450e-915a-79f9d1384885	DE	1397
2	cb3126af-c20b-4adb-9286-335c1902251c	CH	810
3	2e9b1d30-801e-45f1-b4c3-649b7a835ed7	PL	439
4	2e9b1d30-801e-45f1-b4c3-649b7a835ed7	PL	350

1.5 Task A.2

The task is to calculate Transaction Success Rate KPI (%). KPI = the number of users whose 1st transaction was a successful card payment over $10 USD equivalent multiplied by 100 abd divided by the total N of users.

To solve this problem I'll need to have preprocessed data tables as follows: - All the first transaction by each distict user - Having above I will use the conversion rates to get transaction amount in USD - And the last data table to be used derived from above is users that actually had a first successful transaction above 10 USD done by card payment

Having all above I can simply calculate the % of the qualified transaction.

In total I've got just 40 users that had described experience. Which makes just 0.38% of total number of users.

[40]:
```
tsr = pd.read_sql("""
    WITH first_tx AS (
        SELECT DISTINCT ON (USER_ID)
            USER_ID,
            CURRENCY,
            AMOUNT,
            STATE,
            TYPE,
            CREATED_DATE
        FROM transactions
        ORDER BY USER_ID, CREATED_DATE
    ),
    tx_convert_to_usd AS (
        SELECT
            txs.USER_ID,
            txs.CURRENCY,
            txs.AMOUNT,
            txs.TYPE,
            txs.STATE,
            txs.CREATED_DATE,
            txs.AMOUNT * fx_rates.rate AS AMOUNT_USD
        FROM first_tx AS txs
```

```
        LEFT JOIN fx_rates ON fx_rates.ccy = txs.CURRENCY AND fx_rates.base_ccy␣
 ↪= 'USD'
    ),
    select_users AS (
        SELECT USER_ID
        FROM tx_convert_to_usd
        WHERE STATE='COMPLETED'
            AND TYPE='CARD_PAYMENT'
            AND AMOUNT_USD > 10
    )
    SELECT
        COUNT(*) * 100.0 / (SELECT COUNT(*) FROM users)
        AS transaction_success_rate_per_cent
    FROM select_users
    """,
    con=engine)

tsr.head()
```

[40]:

```
   transaction_success_rate_per_cent
0                            0.38835
```

1.6 Task B

1.6.1 Approach Summary

Here I'm providing a short summary of the analytical approach I chose: 1. Main focus of the approach is to identify patterns of known fraudsters across multiple features. 2. So I explored the dataset first to visualize certain patterns, like avg transaction value per merchant category. 3. Then I have started to build the features that reflect user behavior and anomalies. 4. Based on prepared data I assigned the scores to each user based on defined weights of meaningful features. 5. And lastly based on the score I have selected top 5 user resembling fraudsters.

1.6.2 Exploring data

Check if fraudsters have more transactions in crypto than regular users

[41]:

```
# overview of selected transaction feature and their most common values among␣
 ↪all users
tx_data = pd.read_csv('data/transactions.csv')
fraudsters = pd.read_csv('data/fraudsters.csv')

user_transactions = tx_data[~tx_data['USER_ID'].isin(fraudsters['user_id'])]
{col: user_transactions[col].value_counts()[:10] for col in␣
 ↪user_transactions[['ENTRY_METHOD','STATE', 'TYPE', 'SOURCE',␣
 ↪'MERCHANT_CATEGORY']].columns}
```

[41]:
```
{'ENTRY_METHOD': ENTRY_METHOD
 misc    221621
 chip    180299
 cont    158987
 manu     90435
 mags     22276
 mcon       490
 Name: count, dtype: int64,
 'STATE': STATE
 COMPLETED    574023
 DECLINED      43448
 REVERTED      37478
 FAILED        16104
 PENDING        2471
 CANCELLED       495
 RECORDED         89
 Name: count, dtype: int64,
 'TYPE': TYPE
 CARD_PAYMENT     429721
 TOPUP            128769
 P2P               55902
 ATM               45088
 BANK_TRANSFER     14628
 Name: count, dtype: int64,
 'SOURCE': SOURCE
 GAIA        474746
 HERA        114333
 INTERNAL     55971
 MINOS         8145
 LETO          6794
 CRONUS        6022
 NYX           5146
 OPHION        1275
 LIMOS          908
 APOLLO         767
 Name: count, dtype: int64,
 'MERCHANT_CATEGORY': MERCHANT_CATEGORY
 point_of_interest         37919
 supermarket               30095
 restaurant                23595
 cafe                      13055
 bar                       12920
 atm                        9781
 store                      8804
 convenience_store          6773
 grocery_or_supermarket     6609
 bank                       5938
```

```
Name: count, dtype: int64}
```

[42]:
```
# overview of selected transaction feature and their most common values among
↪fraudsters
fraud_transactions = tx_data[tx_data['USER_ID'].isin(fraudsters['user_id'])]
{col: fraud_transactions[col].value_counts()[:10] for col in
↪fraud_transactions[['ENTRY_METHOD','STATE', 'TYPE', 'SOURCE',
↪'MERCHANT_CATEGORY']].columns}
```

[42]:
```
{'ENTRY_METHOD': ENTRY_METHOD
 misc    5741
 manu    3517
 chip    3363
 cont    1885
 mags      35
 mcon       2
 Name: count, dtype: int64,
 'STATE': STATE
 COMPLETED    10484
 DECLINED      2187
 REVERTED      1185
 FAILED         659
 CANCELLED       20
 PENDING          6
 RECORDED         2
 Name: count, dtype: int64,
 'TYPE': TYPE
 CARD_PAYMENT     6849
 TOPUP            3792
 ATM              2236
 BANK_TRANSFER    1230
 P2P               436
 Name: count, dtype: int64,
 'SOURCE': SOURCE
 GAIA        9085
 MINOS       2911
 HERA        1748
 INTERNAL     436
 LETO         141
 CRONUS       117
 LIMOS         53
 NYX           32
 APOLLO        17
 OPHION         3
 Name: count, dtype: int64,
 'MERCHANT_CATEGORY': MERCHANT_CATEGORY
 atm                    951
```

```
point_of_interest    890
supermarket          451
restaurant           220
bank                 208
convenience_store    204
bar                  112
airport               95
accounting            94
gas_station           87
Name: count, dtype: int64}
```

Short notes based on above outputs comparison: 1. For entry method - manual entry is increasing significantly for fraudsters 2. State counts does not provide meaningful insigts 3. For transaction type - ATM type is increasing significantly for fraudsters 4. Transaction soruce does not provide meaningful insigts. Although worth to note - minos source is increasing for fraudsters. 5. Merchant category also seems to have a valid correlation between regular users and fraudsters

Now let's visualize above correlations

Visualize frequency of entry methods for users and fraudsters

[43]:
```
method_counts_fr = fraud_transactions['ENTRY_METHOD'].
 ↪value_counts(normalize=True)
method_counts_us = user_transactions['ENTRY_METHOD'].
 ↪value_counts(normalize=True)

method_merged = pd.merge(
    method_counts_fr,
    method_counts_us,
    on='ENTRY_METHOD',
    how='inner'
)

method_merged
```

[43]:

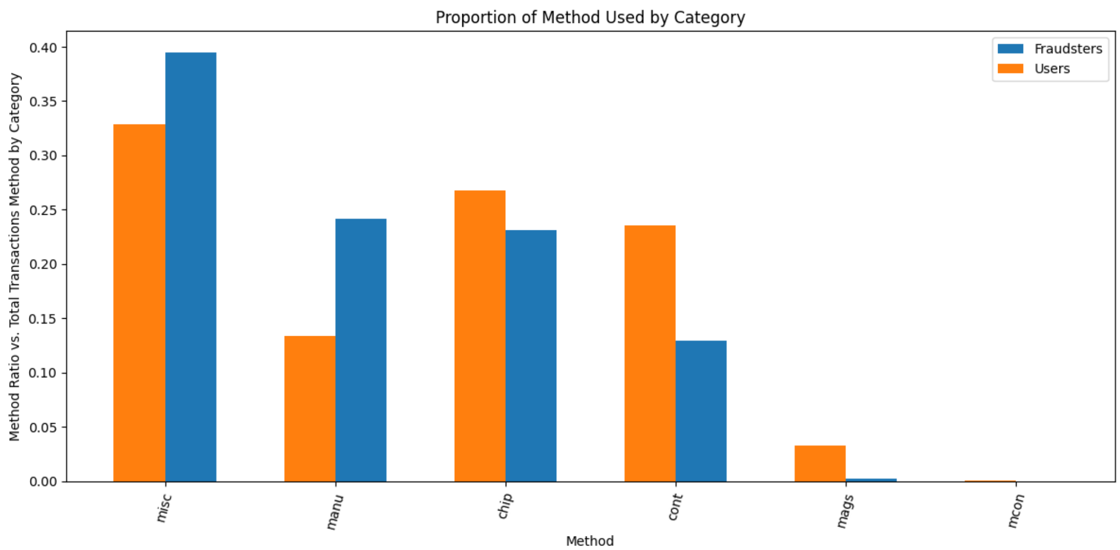
	proportion_x	proportion_y
ENTRY_METHOD		
misc	0.394760	0.328762
manu	0.241835	0.134155
chip	0.231245	0.267463
cont	0.129616	0.235848
mags	0.002407	0.033045
mcon	0.000138	0.000727

[44]:
```
plt.figure(figsize=(12, 6))

indices = np.arange(len(method_merged))
```

```
plt.bar(indices, method_merged['proportion_x'], width=0.3, label='Fraudsters',␣
↪align='edge')
plt.bar(indices - 0.3, method_merged['proportion_y'], width=0.3, label='Users',␣
↪align='edge')

plt.xlabel('Method')
plt.ylabel('Method Ratio vs. Total Transactions Method by Category')
plt.title('Proportion of Method Used by Category')
plt.xticks(rotation=75, ticks=indices, labels=method_merged.index)
plt.legend()
plt.tight_layout()
plt.show()
```



Visualize transaction type among fraudsters and users

[45]:
```
type_counts_fr = fraud_transactions['TYPE'].value_counts(normalize=True)
type_counts_us = user_transactions['TYPE'].value_counts(normalize=True)

type_merged = pd.merge(
    type_counts_fr,
    type_counts_us,
    on='TYPE',
    how='inner'
)

type_merged
```

[45]:
```
                 proportion_x  proportion_y
TYPE
```

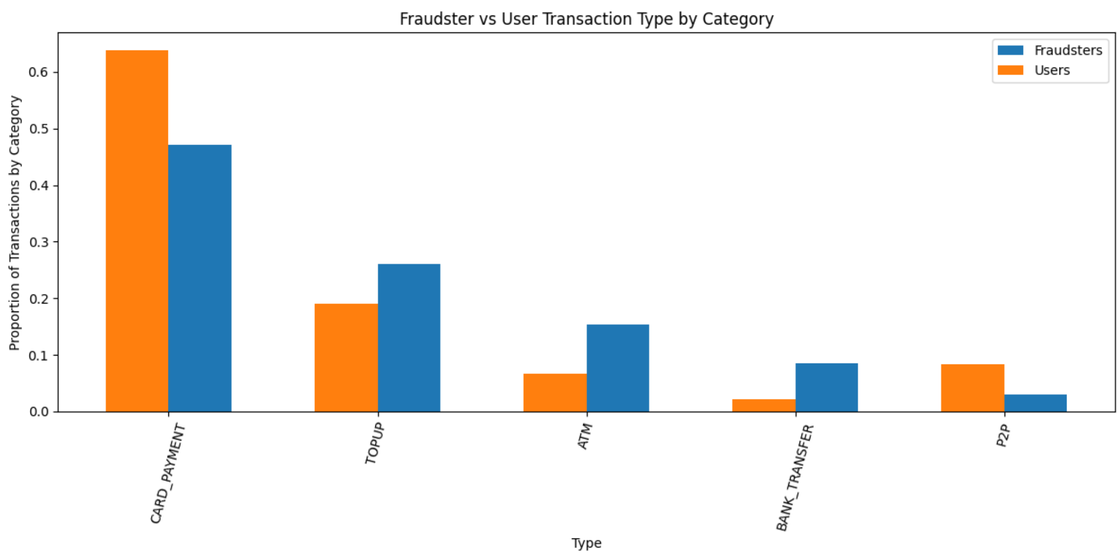
```
CARD_PAYMENT       0.470948     0.637466
TOPUP              0.260744     0.191021
ATM                0.153751     0.066885
BANK_TRANSFER      0.084577     0.021700
P2P                0.029980     0.082927
```

[46]:
```
plt.figure(figsize=(12, 6))

indices = np.arange(len(type_merged))

plt.bar(indices, type_merged['proportion_x'], width=0.3, label='Fraudsters',␣
 ↪align='edge')
plt.bar(indices - 0.3, type_merged['proportion_y'], width=0.3, label='Users',␣
 ↪align='edge')

plt.xlabel('Type')
plt.ylabel('Proportion of Transactions by Category')
plt.title('Fraudster vs User Transaction Type by Category')
plt.xticks(rotation=75, ticks=indices, labels=type_merged.index)
plt.legend()
plt.tight_layout()
plt.show()
```



Visualize transaction amount anomalies based on Merchant Category Compare fraudsters with regular user data

[47]:
```
merch_n_fraudsters = pd.read_sql("""
    WITH tx_users AS (
```

```
        SELECT *
        FROM transactions tx LEFT JOIN users
        ON tx.USER_ID = users.ID
    ),
    preaggregated AS(
        SELECT *,
            tx_users.AMOUNT * fx_rates.rate AS AMOUNT_USD
        FROM tx_users
        LEFT JOIN fx_rates ON fx_rates.ccy = tx_users.CURRENCY
        WHERE fx_rates.ccy = tx_users.CURRENCY AND fx_rates.base_ccy = 'USD'
    ),
    aggregated AS (
        SELECT * FROM preaggregated
        WHERE USER_ID IN (SELECT user_id FROM fraudsters)
    )
    SELECT MERCHANT_CATEGORY, COUNT(*), AVG(AMOUNT_USD), MAX(AMOUNT_USD)
    FROM aggregated
    WHERE MERCHANT_CATEGORY != ''
    GROUP BY MERCHANT_CATEGORY
    LIMIT 20
""", con=engine)
```

[48]:

```
merch_n_users = pd.read_sql("""
    WITH tx_users AS (
        SELECT *
        FROM transactions tx LEFT JOIN users
        ON tx.USER_ID = users.ID
    ),
    preaggregated AS(
        SELECT *,
            tx_users.AMOUNT * fx_rates.rate AS AMOUNT_USD
        FROM tx_users
        LEFT JOIN fx_rates ON fx_rates.ccy = tx_users.CURRENCY
        WHERE fx_rates.ccy = tx_users.CURRENCY AND fx_rates.base_ccy = 'USD'
    ),
    aggregated AS (
        SELECT * FROM preaggregated
        WHERE USER_ID NOT IN (SELECT user_id FROM fraudsters) AND CURRENCY =␣
↪'GBP'
    )
    SELECT
        MERCHANT_CATEGORY,
        COUNT(*),
        AVG(AMOUNT_USD),
        MAX(AMOUNT_USD)
    FROM aggregated
    GROUP BY MERCHANT_CATEGORY
```

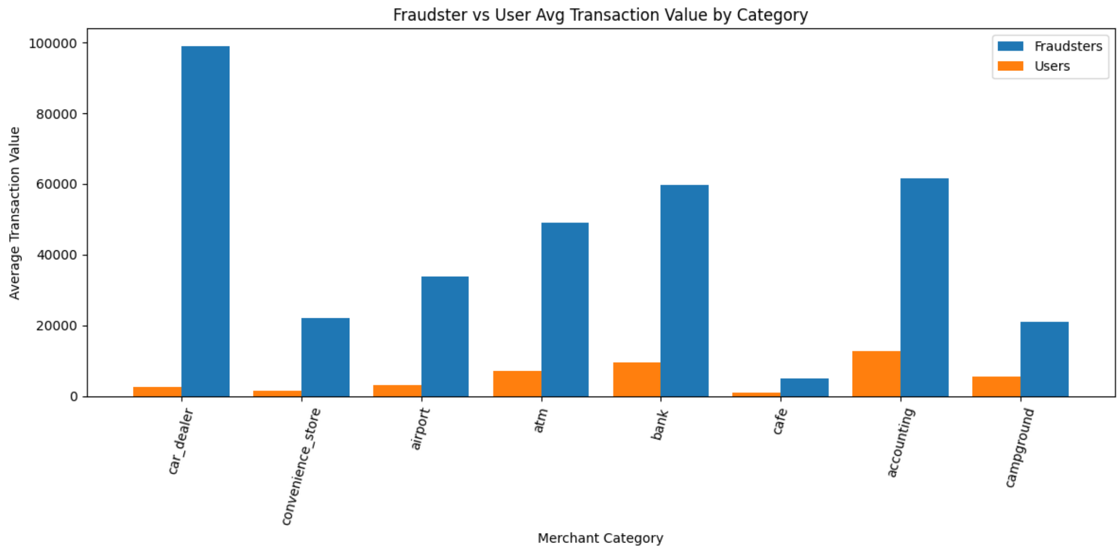
```
""", con=engine)
```

[49]:
```
merch_n_fraudsters['avg'] = merch_n_fraudsters['avg'].astype(int)
merch_n_users['avg'] = merch_n_users['avg'].astype(int)

merch_merged = pd.merge(
    merch_n_fraudsters,
    merch_n_users,
    on='merchant_category',
    how='inner'
)
merch_merged['ratio'] = merch_merged['avg_x'] / merch_merged['avg_y']
merch_merged.sort_values(by='ratio', ascending=False, inplace=True)
top_merch_difs = merch_merged[merch_merged['ratio'] > 3]
```

[50]:
```
plt.figure(figsize=(12, 6))
plt.bar(top_merch_difs['merchant_category'], top_merch_difs['avg_x'], width=0.
 ↪4, label='Fraudsters', align='edge')
plt.bar(top_merch_difs['merchant_category'], top_merch_difs['avg_y'], width=-0.
 ↪4, label='Users', align='edge')

plt.xlabel('Merchant Category')
plt.ylabel('Average Transaction Value')
plt.title('Fraudster vs User Avg Transaction Value by Category')
plt.xticks(rotation=75)
plt.legend()
plt.tight_layout()
plt.show()
```



1.6.3 Data preprocessing

Convert all transaction to USD

[51]:
```
tx_in_usd= pd.read_sql("""
    WITH tx_convert_to_usd AS (
        SELECT
            *,
            txs.AMOUNT * fx_rates.rate AS AMOUNT_USD
        FROM transactions AS txs
        LEFT JOIN fx_rates ON fx_rates.ccy = txs.CURRENCY AND fx_rates.base_ccy␣
 ↪= 'USD'
    )
    SELECT * FROM tx_convert_to_usd
""", con=engine)
```

[52]:
```
tx_in_usd.head()
```

[52]:
```
                                     id currency  amount      state
created_date merchant_category merchant_country entry_method
user_id          type source base_ccy  ccy      rate   amount_usd
0  f40e4635-ae8e-42c6-a620-a69f631b0f8a      PLN    1226  COMPLETED 2018-07-23
17:11:29.228              None              POL          cont
456bb518-648b-41b1-bdba-91f8fbf9cfe3  CARD_PAYMENT   GAIA      USD  PLN
0.281099   344.627969
1  d942648d-f386-4cee-be0c-7813588fad91      EUR    1000  COMPLETED 2018-07-24
07:33:24.004              None              one          misc
884981e0-8ccd-4169-8184-c59e68888b83         TOPUP  LIMOS      USD  EUR
1.177809  1177.809148
2  be9d2c47-2df4-4ec6-9024-c1b9b3c55c45      EUR    1397  COMPLETED 2018-07-23
13:31:47.041              None              AUT          cont
f5288349-94cc-450e-915a-79f9d1384885  CARD_PAYMENT   GAIA      USD  EUR
1.177809  1645.399380
3  9077dad1-3e32-4836-beda-db76cab65c8d      CHF     810  COMPLETED 2018-07-23
09:04:39.836              cafe              CHE          cont
cb3126af-c20b-4adb-9286-335c1902251c  CARD_PAYMENT   GAIA      USD  CHF
1.019113   825.481825
4  ff6a286c-0558-462c-8408-41bca5849ea2      PLN     439  COMPLETED 2018-07-23
19:26:16.486              None              POL          cont
2e9b1d30-801e-45f1-b4c3-649b7a835ed7  CARD_PAYMENT   GAIA      USD  PLN
0.281099   123.402674
```

Merging transaction data with users data. On top we are also adding data from fraudesters to label all users with is_fraud.

[53]:
```
merged_tx_users= pd.read_sql("""
    WITH tx_convert_to_usd AS (
        SELECT
```

```
            *,
            txs.AMOUNT * fx_rates.rate AS AMOUNT_USD
        FROM transactions AS txs
        LEFT JOIN fx_rates ON fx_rates.ccy = txs.CURRENCY AND fx_rates.base_ccy␣
↪= 'USD'
    ),
    merged_data AS (
        SELECT
            tx.*,
            us.kyc AS user_kyc,
            us.birth_year AS user_birth_year,
            us.country AS user_country,
            us.state AS user_state,
            us.has_email AS user_has_email,
            us.terms_version AS user_term_ver,
            us.phone_country AS user_phone_country
        FROM tx_convert_to_usd AS tx
        LEFT JOIN users AS us
        ON tx.USER_ID = us.ID
    )
    SELECT
        merged_data.*,
        CASE WHEN f.user_id IS NOT NULL THEN TRUE ELSE FALSE END AS is_fraud
    FROM merged_data
    LEFT JOIN fraudsters AS f ON merged_data.user_id = f.user_id
""", con=engine)
```

[54]:
```
merged_tx_users.head()
```

[54]:
```
                                     id currency  amount      state
created_date merchant_category merchant_country entry_method
user_id          type  …      rate   amount_usd user_kyc  user_birth_year
user_country user_state  user_has_email user_term_ver user_phone_country
is_fraud
0  f40e4635-ae8e-42c6-a620-a69f631b0f8a      PLN    1226  COMPLETED 2018-07-23
17:11:29.228              None              POL         cont
456bb518-648b-41b1-bdba-91f8fbf9cfe3  CARD_PAYMENT  …  0.281099   344.627969
PASSED              1968              PL      ACTIVE              True     2018-03-20
PL      False
1  d942648d-f386-4cee-be0c-7813588fad91      EUR    1000  COMPLETED 2018-07-24
07:33:24.004              None              one         misc
884981e0-8ccd-4169-8184-c59e68888b83         TOPUP  …  1.177809  1177.809148
PASSED              1956              DE      ACTIVE              True     2018-09-20
DE      False
2  be9d2c47-2df4-4ec6-9024-c1b9b3c55c45      EUR    1397  COMPLETED 2018-07-23
13:31:47.041              None              AUT         cont
f5288349-94cc-450e-915a-79f9d1384885  CARD_PAYMENT  …  1.177809  1645.399380
```

```
PASSED              1985            DE     ACTIVE           True    2018-05-25
DE     False
3  9077dad1-3e32-4836-beda-db76cab65c8d      CHF     810  COMPLETED 2018-07-23
09:04:39.836              cafe              CHE          cont
cb3126af-c20b-4adb-9286-335c1902251c  CARD_PAYMENT  …  1.019113   825.481825
PASSED              1973            CH     ACTIVE           True    2018-05-25
CH     False
4  ff6a286c-0558-462c-8408-41bca5849ea2      PLN     439  COMPLETED 2018-07-23
19:26:16.486              None              POL          cont
2e9b1d30-801e-45f1-b4c3-649b7a835ed7  CARD_PAYMENT  …  0.281099   123.402674
PASSED              1998            PL     ACTIVE           True    2018-03-20
PL     False

[5 rows x 23 columns]
```

1.6.4 Feature building

Creating aggregated features data per user to visualize patterns and potential anomalies. Features selected: 1. Counts of amount of transaction of top 4 states: Completed, Reverted, Declined, Failed. As well % ratio of these states vs. total number of transactions per user. 2. Average and Maximum transactions amount in USD per user. 3. Number of distinct merchant categories, merchant countries and entry methods per user. 4. Available user data: KYC, has email, sign in attempts and if the user is a fraud.

[55]:
```
agg_user_features = pd.read_sql("""
    WITH tx_convert_to_usd AS (
        SELECT
            *,
            txs.AMOUNT * fx_rates.rate AS AMOUNT_USD
        FROM transactions AS txs
        LEFT JOIN fx_rates ON fx_rates.ccy = txs.CURRENCY AND fx_rates.base_ccy␣
↪= 'USD'
    ),
    merged_tx_users AS (
        SELECT
            tx.*,
            us.kyc AS user_kyc,
            us.birth_year AS user_birth_year,
            us.country AS user_country,
            us.state AS user_state,
            us.has_email AS user_has_email,
            us.terms_version AS user_term_ver,
            us.phone_country AS user_phone_country,
            us.failed_sign_in_attempts
        FROM tx_convert_to_usd AS tx
        LEFT JOIN users AS us
        ON tx.USER_ID = us.ID
```

```
    ),
    merged_tx_users_fraud AS (
        SELECT
            merged_tx_users.*,
            CASE WHEN f.user_id IS NOT NULL THEN TRUE ELSE FALSE END AS is_fraud
        FROM merged_tx_users
        LEFT JOIN fraudsters AS f ON merged_tx_users.user_id = f.user_id
    ),
    agg_user_features AS (
        SELECT
            user_id,
            COUNT(*) AS total_tx,
            COUNT(*) FILTER (WHERE state = 'COMPLETED') AS completed_tx,
            COUNT(*) FILTER (WHERE state = 'REVERTED') AS reverted_tx,
            COUNT(*) FILTER (WHERE state = 'DECLINED') AS declined_tx,
            COUNT(*) FILTER (WHERE state = 'FAILED') AS failed_tx,
            ROUND(100.0 * COUNT(*) FILTER (WHERE state = 'COMPLETED') /␣
↪NULLIF(COUNT(*), 0), 2) AS pct_completed,
            ROUND(100.0 * COUNT(*) FILTER (WHERE state = 'REVERTED') /␣
↪NULLIF(COUNT(*), 0), 2) AS pct_reverted,
            ROUND(100.0 * COUNT(*) FILTER (WHERE state = 'DECLINED') /␣
↪NULLIF(COUNT(*), 0), 2) AS pct_declined,
            ROUND(100.0 * COUNT(*) FILTER (WHERE state = 'FAILED') /␣
↪NULLIF(COUNT(*), 0), 2) AS pct_failed,
            AVG(AMOUNT_USD) AS avg_tx_usd,
            MAX(AMOUNT_USD) AS max_tx_usd,
            COUNT(DISTINCT merchant_country) AS country_count,
            COUNT(DISTINCT merchant_category) AS category_count,
            COUNT(DISTINCT entry_method) AS entry_methods,
            MAX(failed_sign_in_attempts) AS failed_sign_in_attempts,
            MAX(user_has_email::int) AS has_email,
            MAX(user_kyc) AS kyc,
            MAX(is_fraud::int)::boolean AS is_fraud
        FROM merged_tx_users_fraud
        GROUP BY user_id
    )
    SELECT * FROM agg_user_features
    ORDER BY pct_reverted DESC NULLS LAST
""", con=engine)
```

[56]:

```
agg_user_features.head()
```

[56]:

```
                                user_id  total_tx  completed_tx  reverted_tx  
declined_tx  failed_tx  pct_completed  pct_reverted  pct_declined  pct_failed  
avg_tx_usd   max_tx_usd  country_count  category_count  entry_methods  
failed_sign_in_attempts  has_email     kyc  is_fraud  
0  efd52c9a-9049-4f0a-a28c-0753b11e9751         1             0            1  
```

```
0           0            0.0          100.0            0.0          0.0    117.780915
117.780915              1               0               1
0           1    NONE     False
1  86a02248-ff68-49ef-87f7-7e90f1ae900f           1             0             1
0           0            0.0          100.0            0.0          0.0    131.990611
131.990611              1               0               1
0           1  FAILED     False
2  552e5758-3209-4bd1-a105-f75f1117f24f           1             0             1
0           0            0.0          100.0            0.0          0.0    117.780915
117.780915              1               0               1
0           1    NONE     False
3  8749917b-a839-49cd-8c56-a69a9f9a6d68           1             0             1
0           0            0.0          100.0            0.0          0.0    131.990611
131.990611              1               0               1
0           0    NONE     False
4  874d449f-773e-4971-937c-dfc2f25541b7           3             0             3
0           0            0.0          100.0            0.0          0.0  3965.290798
5889.045740              1               0               1
0           1  PASSED     False
```

Now to the above data I will also add features that were identified as meaningful on the data exploration stage. Those are created for each user: 1. most_common_entry_method 2. most_common_tx_type 3. top_merchant_category

[57]:
```
agg_user_features_all = pd.read_sql("""
    WITH tx_convert_to_usd AS (
        SELECT
            *,
            txs.AMOUNT * fx_rates.rate AS AMOUNT_USD
        FROM transactions AS txs
        LEFT JOIN fx_rates ON fx_rates.ccy = txs.CURRENCY AND fx_rates.base_ccy␣
↪= 'USD'
    ),
    merged_tx_users AS (
        SELECT
            tx.*,
            us.kyc AS user_kyc,
            us.birth_year AS user_birth_year,
            us.country AS user_country,
            us.state AS user_state,
            us.has_email AS user_has_email,
            us.terms_version AS user_term_ver,
            us.phone_country AS user_phone_country,
            us.failed_sign_in_attempts
        FROM tx_convert_to_usd AS tx
        LEFT JOIN users AS us
        ON tx.USER_ID = us.ID
```

```
    ),
    merged_tx_users_fraud AS (
        SELECT
            merged_tx_users.*,
            CASE WHEN f.user_id IS NOT NULL THEN TRUE ELSE FALSE END AS is_fraud
        FROM merged_tx_users
        LEFT JOIN fraudsters AS f ON merged_tx_users.user_id = f.user_id
    ),
    agg_user_features AS (
        SELECT
            user_id,
            COUNT(*) AS total_tx,
            COUNT(*) FILTER (WHERE state = 'COMPLETED') AS completed_tx,
            COUNT(*) FILTER (WHERE state = 'REVERTED') AS reverted_tx,
            COUNT(*) FILTER (WHERE state = 'DECLINED') AS declined_tx,
            COUNT(*) FILTER (WHERE state = 'FAILED') AS failed_tx,
            ROUND(100.0 * COUNT(*) FILTER (WHERE state = 'COMPLETED') /␣
↪NULLIF(COUNT(*), 0), 2) AS pct_completed,
            ROUND(100.0 * COUNT(*) FILTER (WHERE state = 'REVERTED') /␣
↪NULLIF(COUNT(*), 0), 2) AS pct_reverted,
            ROUND(100.0 * COUNT(*) FILTER (WHERE state = 'DECLINED') /␣
↪NULLIF(COUNT(*), 0), 2) AS pct_declined,
            ROUND(100.0 * COUNT(*) FILTER (WHERE state = 'FAILED') /␣
↪NULLIF(COUNT(*), 0), 2) AS pct_failed,
            AVG(AMOUNT_USD) AS avg_tx_usd,
            MAX(AMOUNT_USD) AS max_tx_usd,
            COUNT(DISTINCT merchant_country) AS country_count,
            COUNT(DISTINCT merchant_category) AS category_count,
            COUNT(DISTINCT entry_method) AS entry_methods,
            MAX(failed_sign_in_attempts) AS failed_sign_in_attempts,
            MAX(user_has_email::int) AS has_email,
            MAX(user_kyc) AS kyc,
            MAX(is_fraud::int)::boolean AS is_fraud
        FROM merged_tx_users_fraud
        GROUP BY user_id
    ),
    most_common_entry_method AS (
        SELECT user_id, entry_method
        FROM (
            SELECT
                user_id,
                entry_method,
                COUNT(*) AS freq,
                ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY COUNT(*) DESC)␣
↪AS row
            FROM merged_tx_users_fraud
            GROUP BY user_id, entry_method
```

```
        )
        WHERE row = 1
    ),
    most_common_tx_type AS (
        SELECT user_id, type
        FROM (
            SELECT
                user_id,
                type,
                COUNT(*) AS freq,
                ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY COUNT(*) DESC)␣
↪AS row
            FROM merged_tx_users_fraud
            GROUP BY user_id, type
        )
        WHERE row = 1
    ),
    top_merchant_category AS (
        SELECT user_id, merchant_category
        FROM (
            SELECT
                user_id,
                merchant_category,
                SUM(amount_usd) AS total_amount,
                ROW_NUMBER() OVER (PARTITION BY user_id ORDER BY␣
↪SUM(amount_usd) DESC) AS row
            FROM merged_tx_users_fraud
            GROUP BY user_id, merchant_category
        )
        WHERE row = 1
    )
    SELECT
        auf.*,
        mcem.entry_method AS most_common_entry_method,
        mct.type AS most_common_type,
        tmc.merchant_category AS top_category_by_amount
        FROM agg_user_features auf
    LEFT JOIN most_common_entry_method mcem ON auf.user_id = mcem.user_id
    LEFT JOIN most_common_tx_type mct ON auf.user_id = mct.user_id
    LEFT JOIN top_merchant_category tmc ON auf.user_id = tmc.user_id
    ORDER BY pct_reverted DESC NULLS LAST;
    """, con=engine)
```

[58]:
```
agg_user_features_all.head()
```

[58]:
```
                                    user_id  total_tx  completed_tx  reverted_tx  
declined_tx  failed_tx  pct_completed  pct_reverted  pct_declined  pct_failed
```

```
…  country_count  category_count  entry_methods  failed_sign_in_attempts
has_email     kyc  is_fraud most_common_entry_method  most_common_type
top_category_by_amount
0  efd52c9a-9049-4f0a-a28c-0753b11e9751         1             0            1
0          0            0.0         100.0           0.0          0.0  …
1              0              1                          0           1    NONE
False                      misc              TOPUP                   None
1  86a02248-ff68-49ef-87f7-7e90f1ae900f         1             0            1
0          0            0.0         100.0           0.0          0.0  …
1              0              1                          0           1  FAILED
False                      misc              TOPUP                   None
2  552e5758-3209-4bd1-a105-f75f1117f24f         1             0            1
0          0            0.0         100.0           0.0          0.0  …
1              0              1                          0           1    NONE
False                      misc              TOPUP                   None
3  8749917b-a839-49cd-8c56-a69a9f9a6d68         1             0            1
0          0            0.0         100.0           0.0          0.0  …
1              0              1                          0           0    NONE
False                      misc              TOPUP                   None
4  874d449f-773e-4971-937c-dfc2f25541b7         3             0            3
0          0            0.0         100.0           0.0          0.0  …
1              0              1                          0           1  PASSED
False                      misc              TOPUP                   None

[5 rows x 22 columns]
```

1.6.5 Looking for patterns/anomalies in aggregated data

[59]:
```python
focus_features = [
    'total_tx', 'completed_tx', 'reverted_tx', 'declined_tx', 'failed_tx',
    'pct_completed', 'pct_reverted', 'pct_declined', 'pct_failed',
    'country_count', 'category_count', 'entry_methods',
    'failed_sign_in_attempts', 'has_email'
]

fraud_stats = agg_user_features.groupby('is_fraud')[focus_features].mean().T
fraud_stats.rename(columns={
    False: 'false',
    True: 'true'
}, inplace=True)
```

[60]:
```python
fraud_stats['ratio'] = fraud_stats[['true']].min(axis=1) /␣
 ↪fraud_stats[['false']].max(axis=1)
fraud_stats.sort_values(by='ratio', ascending=False, inplace=True)
fraud_stats
```

[60]:
```
is_fraud                      false       true     ratio
failed_sign_in_attempts    0.009324   0.016722  1.793478
pct_declined               7.604785  11.764214  1.546949
declined_tx                5.626522   7.314381  1.299983
failed_tx                  2.085470   2.204013  1.056842
has_email                  0.958430   0.996656  1.039883
pct_completed             68.284179  70.878930  1.037999
entry_methods              2.887076   2.608696  0.903577
reverted_tx                4.853406   3.963211  0.816583
pct_failed                 7.045616   5.449833  0.773507
pct_reverted              16.282831  11.582676  0.711343
country_count              4.337995   2.508361  0.578230
total_tx                  87.297073  48.638796  0.557164
category_count             5.819477   3.123746  0.536774
completed_tx              74.336053  35.063545  0.471690
```

Now let's select the most impactful features to distinguish fraudsters

[61]:
```
top_features = fraud_stats.loc[['failed_sign_in_attempts', 'pct_declined',␣
↪'declined_tx', 'failed_tx', 'pct_reverted', 'country_count', 'total_tx',␣
↪'category_count', 'completed_tx']].index.tolist()
```

Normalizing features so that we can use them for fraudster score building & merging with rest of meaningful features

[62]:
```
scoring_df = agg_user_features_all[['user_id', 'is_fraud'] + top_features].
↪copy()

scaler = MinMaxScaler()
scoring_df[top_features] = scaler.fit_transform(scoring_df[top_features])
scoring_df[scoring_df['is_fraud'] == True].head()

meaningful_features = agg_user_features_all[['user_id',␣
↪'most_common_entry_method', 'most_common_type', 'top_category_by_amount']].
↪copy()

scoring_df = pd.merge(
    scoring_df,
    meaningful_features,
    on='user_id',
    how='left'
)

# encoding meaningful features

type_weights = {
    'ATM': 1,
    'BANK_TRANSFER': 1,
```

```
    'P2P': -1
}
scoring_df['most_common_type'] = scoring_df['most_common_type'].
 ↪map(type_weights).fillna(0.0)

entry_method_weights = {
    'manu': 1,
    'cont': -1,
    'mags': -1
}
scoring_df['most_common_entry_method'] = scoring_df['most_common_entry_method'].
 ↪map(entry_method_weights).fillna(0.0)

category_weights = {
    'car_dealer': 1,
    'convenience_store': 1,
    'airport': 1,
    'atm': 1,
    'bank': 1,
    'cafe': 1,
    'accounting': 1,
    'campground': 1
}
scoring_df['top_category_by_amount'] = scoring_df['top_category_by_amount'].
 ↪map(category_weights).fillna(0.0)
```

1.6.6 Assigning scores to users

Now let's design the score to apply for all users, so that we can identify a probability of a user to be a fraud. To define this score I will use selected feautures that have most of the importance and will set for them a coefficient. Coefficients will be based on the ratios identified earlier showing behavior differences between regular users and frauds. If the ratio impacts positively to the side of a fraudster it will be positive, otherwise negative. For the meaningful features the coefficients are selected to be 0.05. Ideally needs to undergo testing for fine tuning.

[63]:
```
# defining dictionary to store weights
score_weights = {
    'completed_tx': fraud_stats.at['completed_tx', 'ratio'],
    'category_count': fraud_stats.at['category_count', 'ratio'],
    'total_tx': fraud_stats.at['total_tx', 'ratio'],
    'failed_sign_in_attempts': fraud_stats.at['failed_sign_in_attempts',␣
 ↪'ratio'],
    'country_count': fraud_stats.at['country_count', 'ratio'],
    'pct_declined': fraud_stats.at['pct_declined', 'ratio'],
    'pct_reverted': fraud_stats.at['pct_reverted', 'ratio'],
    'declined_tx': fraud_stats.at['declined_tx', 'ratio'],
    #'pct_failed': fraud_stats.at['pct_failed', 'ratio'],
```

```
    'most_common_entry_method': 2,
    'most_common_type': 2,
    'top_category_by_amount': 0.5

}
```

[64]:
```
scoring_df['fraud_score'] = sum(
    scoring_df[feature] * weight for feature, weight in score_weights.items()
)

scoring_df.sort_values(by='fraud_score', ascending=False, inplace=True)
scoring_df.reset_index(drop=True, inplace=True)
```

Sense checking solution Higher fraud score -> higher probability of a user to be a fraud.

[65]:
```
scoring_df.head()
```

[65]:
```
                               user_id  is_fraud  failed_sign_in_attempts
pct_declined  declined_tx  failed_tx  pct_reverted  country_count  total_tx
category_count  completed_tx  most_common_entry_method  most_common_type
top_category_by_amount  fraud_score
0  ec4fd825-7167-450b-aa9f-0cbcf681978b     False                      0.0
0.6550     0.526316   0.003876        0.2183        0.127660  0.083700
0.191489       0.011015                       1.0               0.0
1.0      4.581173
1  dc283b17-bbe1-4ae9-a11c-0029d5ae71d9      True                      0.0
0.2731     0.985965   0.019380        0.1273        0.234043  0.377386
0.510638       0.240755                       1.0               0.0
0.0      4.528019
2  72f192ea-d2ff-418b-875d-68424d113a41     False                      0.0
0.9950     0.701754   0.000000        0.0050        0.021277  0.073421
0.021277       0.000000                       1.0               0.0
0.0      4.519671
3  8cdf3d03-5bff-46d0-bc81-53d42c558153     False                      0.0
0.9368     0.312281   0.000000        0.0632        0.106383  0.034508
0.042553       0.000000                       1.0               0.0
1.0      4.503680
4  46172727-471c-4627-b706-1f9881a8e4d2     False                      0.0
0.3033     0.957895   0.077519        0.0856        0.276596  0.330029
0.191489       0.206530                       1.0               0.0
0.0      4.319349
```

[66]:
```
scoring_df[scoring_df['is_fraud'] == True][['fraud_score']].describe()
```

[66]:
```
       fraud_score
count   299.000000
mean      0.920046
```

```
std       1.185202
min      -1.997864
25%       0.178683
50%       0.387731
75%       2.183220
max       4.528019
```

[67]:
```
scoring_df[scoring_df['is_fraud'] == False][['fraud_score']].describe()
```

[67]:
```
       fraud_score
count  7722.000000
mean      0.313741
std       1.160729
min      -1.999814
25%       0.110618
50%       0.314626
75%       0.537115
max       4.581173
```

The outcoume is as expected. The mean bends towards a hiher value for fraudsters. Also 3rd quartile is showing that majority of all users is below mean of fraudsters, so we are actually cutting off the ones that are leaning to fraudsters. Selecting top 5 users based on this data should be a probable set of fraudsters.

Top 5 Fraud Suspects

[68]:
```
scoring_df[scoring_df['is_fraud'] == False][['user_id']][:5]
```

[68]:
```
                                user_id
0  ec4fd825-7167-450b-aa9f-0cbcf681978b
2  72f192ea-d2ff-418b-875d-68424d113a41
3  8cdf3d03-5bff-46d0-bc81-53d42c558153
4  46172727-471c-4627-b706-1f9881a8e4d2
5  62aaa0fb-65ae-44b6-884d-56f38a302b3e
```

The final output highlights the top 5 non-fraud users whose behavior most closely resembles known fraudsters.