

**Important for Governance**

To fully validate and make your model governance-ready:

Document your KIs (description, purpose, thresholds).

Maintain versioning (log rule changes).

Set up periodic monitoring (drift checking, alert volume checking).

Prepare a model limitation statement ("e.g., model assumes available transaction time is accurate").

| \*\*KI ID\*\* | \*\*KI Name\*\* | \*\*Description\*\* | \*\*Trigger Logic\*\* | \*\*Weight\*\* | \*\*Reason for Inclusion\*\* |

| KI01 | High Transaction Amount | Flags transactions with high requested USD amount | `REQUESTED\_AMOUNT\_US\_INTL > 10,000` | 2.0 | Large transactions are more likely to be involved in fraud schemes |

| KI02 | Night-Time Transaction | Flags transactions made between midnight and 5 AM | `hour in [0,1,2,3,4,5]` | 1.0 | Legitimate transactions typically occur during business hours |

| KI03 | New Counterparty Activity | Flags first-time interaction with a counterparty | `first transaction with PARTY\_KEY for ACCOUNT\_KEY` | 1.5 | Fraudsters often create new recipients for laundering funds |

| KI04 | High Transaction Frequency | Flags accounts with more than 3 transactions in 1 hour | `txn\_count\_last\_1h > 3` | 2.0 | Rapid multiple transactions could indicate account takeover or money mule activity |

| KI05 | Large Deviation from Normal Behavior | Flags transactions >2 std deviations from account mean | `abs(amt\_deviation) > 2 \* rolling\_std\_amt` | 1.5 | Significant deviations from typical behavior can be suspicious |

| KI06 | Fast Repeat Transactions | Less than 60 seconds between two transactions | `seconds\_since\_last\_txn < 60` | 1.0 | Fraudsters may send rapid bursts before detection |

| KI07 | Currency Mismatch | Transaction in a new/unusual currency for the account | `CURRENCY\_CD different from historical avg` | 1.0 | Cross-currency laundering or irregular activity |

| KI08 | International High-Value Wire | Large transaction with foreign counterparty | `high amount + international PARTY\_KEY` | 2.5 | International wires are more risky and regulated |

| KI09 | Suspicious Channel Use | High-value transaction via mobile or online channel | `CHANNEL\_offline == 0 and txn > 5000` | 1.5 | High-value via self-service channel could be suspicious |

| KI10 | Same Device, Different Accounts | Multiple accounts accessed from same device fingerprint (if available) | `device\_id linked to >2 ACCOUNT\_KEYs in 1 day` | 2.5 | Sign of mule networks or account takeovers

Documentation Requirements

When you implement your own Actimize-style system:

Keep a full KI Inventory (like above table).

Maintain a version log whenever you:

Add a KI

Modify a KI threshold

Change a KI weight

Explain in the documentation:

Why each KI exists

How thresholds were initially selected

How thresholds are re-calibrated (e.g., based on alert volumes or SAR conversion)

**Great! Below is a ready-to-use Python template that:**

Implements your custom KI library

Applies each KI as a rule-based flag

Aggregates them into a fraud risk score

Flags transactions above a defined risk threshold

**Python Template: Actimize-Style Rule-Based Fraud Scoring Engine**

import pandas as pd

import numpy as np

# Load your transaction data

df = pd.read\_csv("transactions\_aug2024.csv")

# --- Step 1: Feature Engineering (assumes datetime is already parsed) ---

df['TRANSACTION\_LOCAL\_DATE\_TIME'] = pd.to\_datetime(df['TRANSACTION\_LOCAL\_DATE\_TIME'])

df['hour'] = df['TRANSACTION\_LOCAL\_DATE\_TIME'].dt.hour

df.sort\_values(by=['ACCOUNT\_KEY', 'TRANSACTION\_LOCAL\_DATE\_TIME'], inplace=True)

# Required engineered features (add if not yet done)

df['account\_mean\_amt'] = df.groupby('ACCOUNT\_KEY')['REQUESTED\_AMOUNT\_US\_INTL'].transform('mean')

df['rolling\_std\_amt'] = df.groupby('ACCOUNT\_KEY')['REQUESTED\_AMOUNT\_US\_INTL'].rolling(window=5, min\_periods=1).std().reset\_index(0, drop=True).fillna(0)

df['amt\_deviation'] = df['REQUESTED\_AMOUNT\_US\_INTL'] - df['account\_mean\_amt']

df['txn\_count\_last\_1h'] = df.groupby('ACCOUNT\_KEY')['TRANSACTION\_LOCAL\_DATE\_TIME'].rolling('1h', on='TRANSACTION\_LOCAL\_DATE\_TIME').count().reset\_index(0, drop=True)

df['seconds\_since\_last\_txn'] = df.groupby('ACCOUNT\_KEY')['TRANSACTION\_LOCAL\_DATE\_TIME'].diff().dt.total\_seconds().fillna(9999)

df['is\_new\_party'] = ~df.duplicated(subset=['ACCOUNT\_KEY', 'PARTY\_KEY'])

# Optional: Dummy channel encoding (if available)

if 'CHANNEL' in df.columns:

df = pd.get\_dummies(df, columns=['CHANNEL'], drop\_first=True)

# --- Step 2: Apply KIs ---

df['KI01\_high\_amt'] = (df['REQUESTED\_AMOUNT\_US\_INTL'] > 10000).astype(int)

df['KI02\_night\_txn'] = df['hour'].isin([0,1,2,3,4,5]).astype(int)

df['KI03\_new\_party'] = df['is\_new\_party'].astype(int)

df['KI04\_high\_velocity'] = (df['txn\_count\_last\_1h'] > 3).astype(int)

df['KI05\_amt\_deviation'] = (np.abs(df['amt\_deviation']) > 2 \* df['rolling\_std\_amt']).astype(int)

df['KI06\_quick\_repeat'] = (df['seconds\_since\_last\_txn'] < 60).astype(int)

df['KI07\_unusual\_currency'] = (df.groupby('ACCOUNT\_KEY')['CURRENCY\_CD'].transform('nunique') > 1).astype(int)

# Example: Add KI08 if you have flags like 'is\_foreign\_party'

# df['KI08\_intl\_wire'] = (df['REQUESTED\_AMOUNT\_US\_INTL'] > 5000) & (df['COUNTRY\_CODE'] != 'US')

# Optional: Suspicious self-service channel

if 'CHANNEL\_offline' in df.columns:

df['KI09\_self\_service\_high'] = ((df['REQUESTED\_AMOUNT\_US\_INTL'] > 5000) & (df['CHANNEL\_offline'] == 0)).astype(int)

else:

df['KI09\_self\_service\_high'] = 0

# --- Step 3: Scoring Engine (Weighted) ---

df['fraud\_score'] = (

df['KI01\_high\_amt'] \* 2.0 +

df['KI02\_night\_txn'] \* 1.0 +

df['KI03\_new\_party'] \* 1.5 +

df['KI04\_high\_velocity'] \* 2.0 +

df['KI05\_amt\_deviation'] \* 1.5 +

df['KI06\_quick\_repeat'] \* 1.0 +

df['KI07\_unusual\_currency'] \* 1.0 +

df['KI09\_self\_service\_high'] \* 1.5

)

# --- Step 4: Flagging Potential Fraud ---

threshold = 4.0

df['potential\_fraud'] = (df['fraud\_score'] >= threshold).astype(int)

# --- Step 5: Output Results ---

df[['ACCOUNT\_KEY', 'REQUESTED\_AMOUNT\_US\_INTL', 'fraud\_score', 'potential\_fraud'] + [col for col in df.columns if col.startswith('KI')]].to\_excel("rule\_based\_fraud\_flags.xlsx", index=False)

print("Top flagged transactions:")

print(df[df['potential\_fraud'] == 1][['ACCOUNT\_KEY', 'REQUESTED\_AMOUNT\_US\_INTL', 'fraud\_score']].head())

**----**

**Here’s your dashboard output for the rule-based fraud model:**

**1. Fraud Score Distribution**

**Shows how fraud scores are spread across the dataset.**

**Helps you visualize where most of your transactions fall and where alerts cluster.**

**2. Key Indicator (KI) Activation Rates**

**Reveals how frequently each KI is triggered.**

**Useful for identifying overly sensitive (or inactive) rules.**

**Would you like me to help you build a dashboard layout (e.g., KI activation rates, score distribution, or alert drill-down table) next?**

# Re-import necessary libraries after code execution state reset

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import ace\_tools as tools

# Simulate a dataset again

np.random.seed(42)

n = 1000

data = {

'fraud\_score': np.random.choice([0, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 6, 7, 8], size=n, p=[0.05, 0.1, 0.1, 0.1, 0.15, 0.1, 0.1, 0.05, 0.1, 0.05, 0.05, 0.05]),

'KI01\_high\_amt': np.random.binomial(1, 0.2, n),

'KI02\_night\_txn': np.random.binomial(1, 0.1, n),

'KI03\_new\_party': np.random.binomial(1, 0.15, n),

'KI04\_high\_velocity': np.random.binomial(1, 0.05, n),

'KI05\_amt\_deviation': np.random.binomial(1, 0.12, n),

'KI06\_quick\_repeat': np.random.binomial(1, 0.08, n),

'KI07\_unusual\_currency': np.random.binomial(1, 0.1, n),

'KI09\_self\_service\_high': np.random.binomial(1, 0.07, n)

}

df = pd.DataFrame(data)

df['potential\_fraud'] = (df['fraud\_score'] >= 4).astype(int)

# Create activation rate bar chart

activation\_rates = df[[col for col in df.columns if col.startswith('KI')]].mean().sort\_values()

# Display fraud score distribution

plt.figure(figsize=(10, 5))

sns.histplot(df['fraud\_score'], bins=15, kde=True)

plt.title("Fraud Score Distribution")

plt.xlabel("Fraud Score")

plt.ylabel("Transaction Count")

plt.grid(True)

plt.show()

# Display KI activation rates

plt.figure(figsize=(10, 6))

sns.barplot(x=activation\_rates.values, y=activation\_rates.index)

plt.title("Key Indicator (KI) Activation Rates")

plt.xlabel("Activation Rate")

plt.ylabel("KI")

plt.grid(True)

plt.show()

# Create summary table

summary = df.groupby('potential\_fraud').agg(

count=('fraud\_score', 'count'),

avg\_score=('fraud\_score', 'mean')

).reset\_index()

tools.display\_dataframe\_to\_user(name="Fraud Alert Summary", dataframe=summary)

-------

Great — even with only those 7 fields available, you can still run a meaningful rule-based fraud detection engine, as long as you:

**Build or access a historical profile model (from 2024 data)**

Use that model to derive dynamic behavioral features at runtime

Apply your KI logic using engineered fields

Available Fields You Have

Field Use

REQUESTED\_AMOUNT\_US\_INTL Transaction value — used for amount-based rules

CURRENCY\_CD For detecting currency mismatch

ACCOUNT\_KEY Used to retrieve account profile (mean, std, typical currency)

CHANNEL Used to detect suspicious channels (mobile/web)

TRANSACTION\_LOCAL\_DATE\_TIME Used to extract hour, detect night-time txns, velocity, repeat behavior

PARTY\_KEY For new party flag

Currency Duplicate of CURRENCY\_CD (safe to drop)

How to Detect Suspicious Transactions with Only These Fields

**1. Preload Account Profiles**

Before looping through 2025 transactions, load 2024 profiles:

python

Copy

Edit

import pandas as pd

# Load 2024 transaction data

df\_2024 = pd.read\_csv("transactions\_2024.csv") # or .xlsx using read\_excel

df\_2024['TRANSACTION\_LOCAL\_DATE\_TIME'] = pd.to\_datetime(df\_2024['TRANSACTION\_LOCAL\_DATE\_TIME'])

df\_2024.sort\_values(by=['ACCOUNT\_KEY', 'TRANSACTION\_LOCAL\_DATE\_TIME'], inplace=True)

# Compute baseline statistics per account

account\_profiles = df\_2024.groupby('ACCOUNT\_KEY').agg({

'REQUESTED\_AMOUNT\_US\_INTL': ['mean', 'std'],

'CURRENCY\_CD': lambda x: x.mode()[0] if not x.mode().empty else 'USD'

})

account\_profiles.columns = ['mean\_amt', 'std\_amt', 'typical\_currency']

account\_profiles.reset\_index(inplace=True)

# Save this for later use

account\_profiles.to\_csv("account\_profiles\_model.csv", index=False)

account\_profiles = pd.read\_csv("account\_profiles\_model.csv")

# Columns: ACCOUNT\_KEY, mean\_amt, std\_amt, typical\_currency

**2. Maintain Context for Velocity and Repeat Detection**

Use a dictionary to track for each account:

last\_txn\_time

past\_parties used

Rolling window of timestamps for txn\_count\_last\_1h

**3. Loop Through Each Transaction**

Here’s a simplified Python function to evaluate one transaction at a time:

from datetime import datetime, timedelta

# Sample transaction record (replace with real values)

txn = {

'REQUESTED\_AMOUNT\_US\_INTL': 12000,

'CURRENCY\_CD': 'EUR',

'ACCOUNT\_KEY': 'A123',

'CHANNEL': 'mobile',

'TRANSACTION\_LOCAL\_DATE\_TIME': '2025-01-10T02:35:00',

'PARTY\_KEY': 'P987'

}

# Context to track past activity (simulate database/memory)

account\_context = {

'A123': {

'last\_txn\_time': datetime(2025, 1, 10, 1, 50),

'parties\_seen': {'P111', 'P222'},

'txn\_times': [datetime(2025, 1, 10, 1, 20), datetime(2025, 1, 10, 1, 50)]

}

}

# Account profiles loaded from 2024

account\_profiles = {

'A123': {'mean\_amt': 3200, 'std\_amt': 800, 'typical\_currency': 'USD'}

}

# Evaluate

def score\_transaction(txn, context, profiles):

score = 0

klist = []

acct = txn['ACCOUNT\_KEY']

amount = txn['REQUESTED\_AMOUNT\_US\_INTL']

hour = pd.to\_datetime(txn['TRANSACTION\_LOCAL\_DATE\_TIME']).hour

now = pd.to\_datetime(txn['TRANSACTION\_LOCAL\_DATE\_TIME'])

# Profile stats

mean\_amt = profiles.get(acct, {}).get('mean\_amt', 0)

std\_amt = profiles.get(acct, {}).get('std\_amt', 1)

typical\_curr = profiles.get(acct, {}).get('typical\_currency', txn['CURRENCY\_CD'])

# Deviation

if abs(amount - mean\_amt) > 2 \* std\_amt:

score += 1.5

klist.append("KI05\_amt\_deviation")

# High amount

if amount > 10000:

score += 2.0

klist.append("KI01\_high\_amt")

# Night-time

if hour in [0, 1, 2, 3, 4, 5]:

score += 1.0

klist.append("KI02\_night\_txn")

# Channel flag

if txn['CHANNEL'].lower() in ['mobile', 'web'] and amount > 5000:

score += 1.5

klist.append("KI09\_self\_service\_high")

# New party

if txn['PARTY\_KEY'] not in context[acct]['parties\_seen']:

score += 1.5

klist.append("KI03\_new\_party")

# Seconds since last transaction

last\_time = context[acct]['last\_txn\_time']

seconds\_diff = (now - last\_time).total\_seconds()

if seconds\_diff < 60:

score += 1.0

klist.append("KI06\_quick\_repeat")

# Velocity (txn count in last 1 hour)

recent\_txns = [t for t in context[acct]['txn\_times'] if now - t <= timedelta(hours=1)]

if len(recent\_txns) > 3:

score += 2.0

klist.append("KI04\_high\_velocity")

# Currency mismatch

if txn['CURRENCY\_CD'] != typical\_curr:

score += 1.0

klist.append("KI07\_unusual\_currency")

# Final

fraud\_flag = int(score >= 4.0)

return {

'score': score,

'fraud\_flag': fraud\_flag,

'triggered\_KIs': klist

}

result = score\_transaction(txn, account\_context, account\_profiles)

print(result)

----

import pandas as pd

from datetime import datetime, timedelta

# Load account behavior profiles from 2024

account\_profiles = pd.read\_csv("account\_profiles\_model.csv")

account\_profiles = account\_profiles.set\_index("ACCOUNT\_KEY").to\_dict(orient="index")

# Load 2025 transaction data

df = pd.read\_excel("transactions\_2025.xlsx")

df['TRANSACTION\_LOCAL\_DATE\_TIME'] = pd.to\_datetime(df['TRANSACTION\_LOCAL\_DATE\_TIME'])

df.sort\_values(by=['ACCOUNT\_KEY', 'TRANSACTION\_LOCAL\_DATE\_TIME'], inplace=True)

# Initialize context

account\_context = {}

results = []

for idx, txn in df.iterrows():

acct = txn['ACCOUNT\_KEY']

now = txn['TRANSACTION\_LOCAL\_DATE\_TIME']

amount = txn['REQUESTED\_AMOUNT\_US\_INTL']

hour = now.hour

party = txn['PARTY\_KEY']

channel = txn['CHANNEL']

currency = txn['CURRENCY\_CD']

# Prepare context

if acct not in account\_context:

account\_context[acct] = {

'last\_txn\_time': now - timedelta(hours=2),

'parties\_seen': set(),

'txn\_times': []

}

context = account\_context[acct]

profile = account\_profiles.get(acct, {'mean\_amt': 0, 'std\_amt': 1, 'typical\_currency': currency})

score = 0

klist = []

# Rule-based checks

if abs(amount - profile['mean\_amt']) > 2 \* profile['std\_amt']:

score += 1.5

klist.append("KI05\_amt\_deviation")

if amount > 10000:

score += 2.0

klist.append("KI01\_high\_amt")

if hour in [0, 1, 2, 3, 4, 5]:

score += 1.0

klist.append("KI02\_night\_txn")

if channel.lower() in ['mobile', 'web'] and amount > 5000:

score += 1.5

klist.append("KI09\_self\_service\_high")

if party not in context['parties\_seen']:

score += 1.5

klist.append("KI03\_new\_party")

seconds\_diff = (now - context['last\_txn\_time']).total\_seconds()

if seconds\_diff < 60:

score += 1.0

klist.append("KI06\_quick\_repeat")

recent\_txns = [t for t in context['txn\_times'] if now - t <= timedelta(hours=1)]

if len(recent\_txns) > 3:

score += 2.0

klist.append("KI04\_high\_velocity")

if currency != profile['typical\_currency']:

score += 1.0

klist.append("KI07\_unusual\_currency")

fraud\_flag = int(score >= 4.0)

# Store results

results.append({

'ACCOUNT\_KEY': acct,

'PARTY\_KEY': party,

'REQUESTED\_AMOUNT\_US\_INTL': amount,

'TRANSACTION\_LOCAL\_DATE\_TIME': now,

'fraud\_score': score,

'fraud\_flag': fraud\_flag,

'triggered\_KIs': ", ".join(klist)

})

# Update context

context['last\_txn\_time'] = now

context['txn\_times'].append(now)

context['parties\_seen'].add(party)

# Save results to Excel

results\_df = pd.DataFrame(results)

results\_df.to\_excel("flagged\_transactions\_2025.xlsx", index=False)

print("Done: Output written to flagged\_transactions\_2025.xlsx")

# Save to file

script\_path = "/mnt/data/fraud\_detection\_demo\_2025.py"

with open(script\_path, "w") as f:

f.write(demo\_script)

script\_path

# Create a cleaned, modular version of the fraud detection script

cleaned\_script = '''

import pandas as pd

from datetime import datetime, timedelta

# === Load Account Profiles (2024 behavior model) ===

account\_profiles = pd.read\_csv("account\_profiles\_model.csv")

account\_profiles = account\_profiles.set\_index("ACCOUNT\_KEY").to\_dict(orient="index")

# === Load Transactions for 2025 ===

df = pd.read\_excel("transactions\_2025.xlsx")

df['TRANSACTION\_LOCAL\_DATE\_TIME'] = pd.to\_datetime(df['TRANSACTION\_LOCAL\_DATE\_TIME'])

df.sort\_values(by=['ACCOUNT\_KEY', 'TRANSACTION\_LOCAL\_DATE\_TIME'], inplace=True)

# === Initialize Context Tracking ===

account\_context = {}

results = []

# === KI Check Functions ===

def ki01\_high\_amt(amount):

return 2.0 if amount > 10000 else 0, "KI01\_high\_amt" if amount > 10000 else None

def ki02\_night\_txn(hour):

return 1.0 if hour in [0, 1, 2, 3, 4, 5] else 0, "KI02\_night\_txn" if hour in [0, 1, 2, 3, 4, 5] else None

def ki03\_new\_party(party, known\_parties):

return (1.5, "KI03\_new\_party") if party not in known\_parties else (0, None)

def ki04\_high\_velocity(txn\_times, now):

recent = [t for t in txn\_times if now - t <= timedelta(hours=1)]

return (2.0, "KI04\_high\_velocity") if len(recent) > 3 else (0, None)

def ki05\_amt\_deviation(amount, mean, std):

return (1.5, "KI05\_amt\_deviation") if abs(amount - mean) > 2 \* std else (0, None)

def ki06\_quick\_repeat(now, last\_time):

return (1.0, "KI06\_quick\_repeat") if (now - last\_time).total\_seconds() < 60 else (0, None)

def ki07\_currency\_mismatch(curr, typical\_curr):

return (1.0, "KI07\_unusual\_currency") if curr != typical\_curr else (0, None)

def ki09\_self\_service(channel, amount):

return (1.5, "KI09\_self\_service\_high") if channel.lower() in ['mobile', 'web'] and amount > 5000 else (0, None)

# === Loop Through Transactions and Score ===

for \_, txn in df.iterrows():

acct = txn['ACCOUNT\_KEY']

now = txn['TRANSACTION\_LOCAL\_DATE\_TIME']

hour = now.hour

party = txn['PARTY\_KEY']

amount = txn['REQUESTED\_AMOUNT\_US\_INTL']

channel = txn['CHANNEL']

currency = txn['CURRENCY\_CD']

# Init context if first transaction for this account

if acct not in account\_context:

account\_context[acct] = {

'last\_txn\_time': now - timedelta(hours=2),

'parties\_seen': set(),

'txn\_times': []

}

context = account\_context[acct]

profile = account\_profiles.get(acct, {'mean\_amt': 0, 'std\_amt': 1, 'typical\_currency': currency})

score = 0

triggered\_kis = []

# Apply all KIs

for ki\_func in [ki01\_high\_amt, ki02\_night\_txn, ki03\_new\_party, ki04\_high\_velocity,

ki05\_amt\_deviation, ki06\_quick\_repeat, ki07\_currency\_mismatch, ki09\_self\_service]:

args = {

ki01\_high\_amt: (amount,),

ki02\_night\_txn: (hour,),

ki03\_new\_party: (party, context['parties\_seen']),

ki04\_high\_velocity: (context['txn\_times'], now),

ki05\_amt\_deviation: (amount, profile['mean\_amt'], profile['std\_amt']),

ki06\_quick\_repeat: (now, context['last\_txn\_time']),

ki07\_currency\_mismatch: (currency, profile['typical\_currency']),

ki09\_self\_service: (channel, amount)

}[ki\_func]

pts, name = ki\_func(\*args)

score += pts

if name:

triggered\_kis.append(name)

fraud\_flag = int(score >= 4.0)

results.append({

'ACCOUNT\_KEY': acct,

'PARTY\_KEY': party,

'REQUESTED\_AMOUNT\_US\_INTL': amount,

'TRANSACTION\_LOCAL\_DATE\_TIME': now,

'fraud\_score': score,

'fraud\_flag': fraud\_flag,

'triggered\_KIs': ", ".join(triggered\_kis)

})

# Update context

context['last\_txn\_time'] = now

context['txn\_times'].append(now)

context['parties\_seen'].add(party)

# === Save Final Results ===

results\_df = pd.DataFrame(results)

results\_df.to\_excel("flagged\_transactions\_2025.xlsx", index=False)

print("Scoring complete. Output saved to flagged\_transactions\_2025.xlsx.")

'''

# Save the cleaned script

script\_path = "/mnt/data/fraud\_detection\_cleaned\_modular.py"

with open(script\_path, "w") as f:

f.write(cleaned\_script)

script\_path