```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd


file_path = '/content/drive/MyDrive/gnn_fraud_project
/PS_20174392719_1491204439457_log.csv'


# STEP 1 — Colab Setup


!pip install pandas numpy matplotlib seaborn tqdm scikit-learn
```

```
Requirement already satisfied: pandas in
/usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: seaborn in
/usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-
packages (4.67.3)
Requirement already satisfied: scikit-learn in
/usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.12/dist-packages (from pandas) (2025.3)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (26.0)
Requirement already satisfied: pillow>=8 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.12/dist-packages (from matplotlib) (3.3.2)
Requirement already satisfied: scipy>=1.6.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.3)
```

```
Requirement already satisfied: joblib>=1.2.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in
/usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2-
>pandas) (1.17.0)
```

```python
# STEP 2 — Load Dataset

import pandas as pd
import numpy as np


file_path = '/content/drive/MyDrive/gnn_fraud_project
/PS_20174392719_1491204439457_log.csv'

df = pd.read_csv(file_path)

print("Shape:", df.shape)
df.head()
```

```
Shape: (6362620, 11)
```

{"type":"dataframe","variable_name":"df"}

```python
# STEP 3 — Basic Audit (Very Important)

# Check fraud ratio
fraud_ratio = df['isFraud'].mean()
print("Fraud Ratio:", fraud_ratio)

# Unique accounts
print("Unique Origin Accounts:", df['nameOrig'].nunique())
print("Unique Destination Accounts:", df['nameDest'].nunique())

# Time range
print("Min step:", df['step'].min())
print("Max step:", df['step'].max())
```

```
Fraud Ratio: 0.001290820448180152
Unique Origin Accounts: 6353307
Unique Destination Accounts: 2722362
Min step: 1
Max step: 743
```

```python
# STEP 4 — Memory Optimization (CRITICAL)

# Convert types
df['type'] = df['type'].astype('category')

for col in ['amount', 'oldbalanceOrg', 'newbalanceOrig',
            'oldbalanceDest', 'newbalanceDest']:
    df[col] = df[col].astype('float32')

for col in ['step', 'isFraud', 'isFlaggedFraud']:
    df[col] = df[col].astype('int32')

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype   
---  ------          -----   
 0   step            int32   
 1   type            category
 2   amount          float32 
 3   nameOrig        object  
 4   oldbalanceOrg   float32 
 5   newbalanceOrig  float32 
 6   nameDest        object  
 7   oldbalanceDest  float32 
 8   newbalanceDest  float32 
 9   isFraud         int32   
 10  isFlaggedFraud  int32   
dtypes: category(1), float32(5), int32(3), object(2)
memory usage: 297.3+ MB
None
```

```
# STEP 5 – Temporal Split

train_df = df[df['step'] <= 500].copy()
val_df = df[(df['step'] > 500) & (df['step'] <= 600)]
test_df = df[df['step'] > 600]

print(train_df.shape, val_df.shape, test_df.shape)
```

```
(6061807, 11) (197240, 11) (103573, 11)
```

```
# STEP 6 – Basic Feature Engineering (Transaction-Level)

# Transaction velocity per origin account
train_df['tx_count'] = train_df.groupby('nameOrig')
['nameOrig'].transform('count')

# Avg amount per origin
train_df['avg_amt_orig'] = train_df.groupby('nameOrig')
['amount'].transform('mean')

# Encode transaction type
train_df = pd.get_dummies(train_df, columns=['type'], drop_first=True)
```

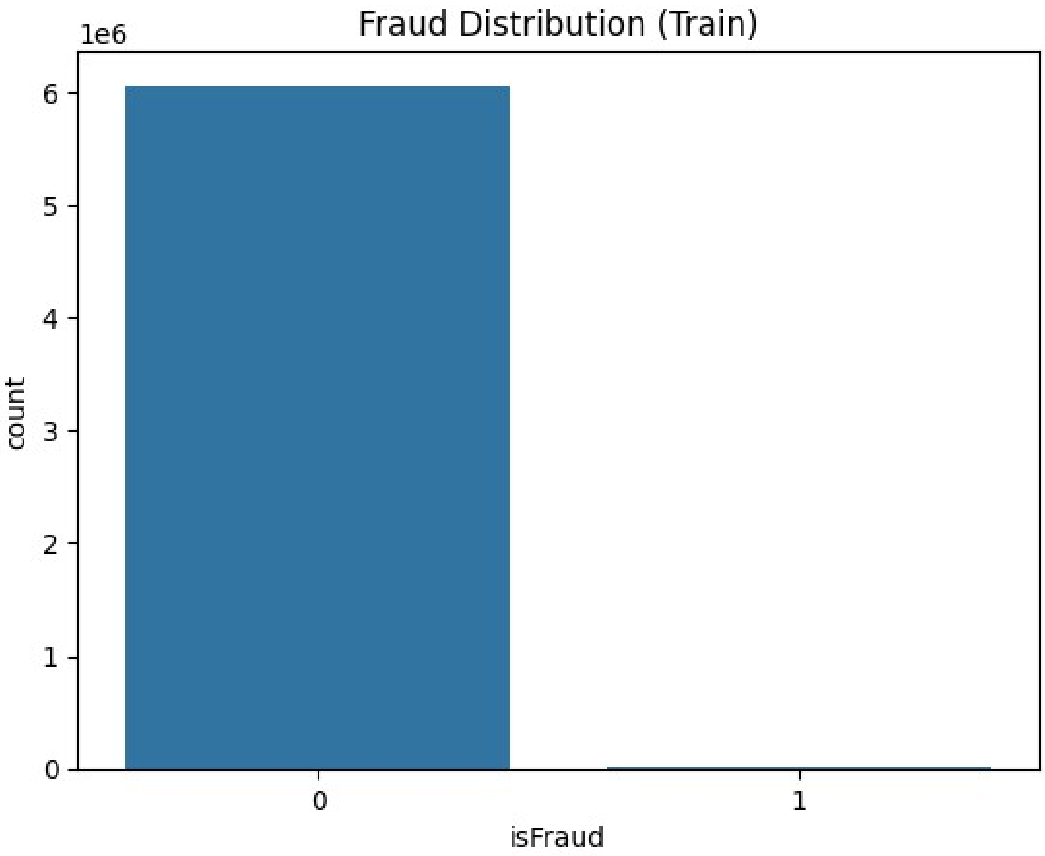
```
train_df.head()
```

```
{"type":"dataframe","variable_name":"train_df"}
```

```
# STEP 7 — Fraud Distribution Visualization

import matplotlib.pyplot as plt
import seaborn as sns

sns.countplot(x='isFraud', data=train_df)
plt.title("Fraud Distribution (Train)")
plt.show()
```



```
# WEEK 2 — TABULAR BASELINE (NO GNN YET)

# STEP 0 — Data Leakage Fix (Very Important)

train_df = df[df['step'] <= 500].copy()
val_df = df[(df['step'] > 500) & (df['step'] <= 600)].copy()
test_df = df[df['step'] > 600].copy()

# STEP 1 — Feature Engineering (Safe Version)
# Historical stats from TRAIN ONLY
orig_tx_count =
train_df.groupby('nameOrig').size().rename("orig_tx_count")
orig_avg_amt = train_df.groupby('nameOrig')
['amount'].mean().rename("orig_avg_amt")

# Merge into train
train_df = train_df.merge(orig_tx_count, on='nameOrig', how='left')
train_df = train_df.merge(orig_avg_amt, on='nameOrig', how='left')

# Val/Test me leakage-free merge
val_df = val_df.merge(orig_tx_count, on='nameOrig', how='left')
val_df = val_df.merge(orig_avg_amt, on='nameOrig', how='left')

test_df = test_df.merge(orig_tx_count, on='nameOrig', how='left')
test_df = test_df.merge(orig_avg_amt, on='nameOrig', how='left')

----------------------------------------------------------------------
-----
NameError                                  Traceback (most recent call
last)
/tmp/ipython-input-824243144.py in <cell line: 0>()
      1 # Val/Test me leakage-free merge
----> 2 val_df = val_df.merge(orig_tx_count, on='nameOrig',
how='left')
      3 val_df = val_df.merge(orig_avg_amt, on='nameOrig', how='left')
      4
      5 test_df = test_df.merge(orig_tx_count, on='nameOrig',
how='left')

NameError: name 'val_df' is not defined

# Missing values handle karo (cold start accounts)
for df_ in [train_df, val_df, test_df]:
    df_['orig_tx_count'] = df_['orig_tx_count'].fillna(0)
    df_['orig_avg_amt'] = df_['orig_avg_amt'].fillna(0)

# STEP 2 — Encode Transaction Type

train_df = pd.get_dummies(train_df, columns=['type'], drop_first=True)
val_df = pd.get_dummies(val_df, columns=['type'], drop_first=True)
test_df = pd.get_dummies(test_df, columns=['type'], drop_first=True)

val_df = val_df.reindex(columns=train_df.columns, fill_value=0)
test_df = test_df.reindex(columns=train_df.columns, fill_value=0)

# STEP 3 — Feature Selection

drop_cols = ['nameOrig', 'nameDest', 'isFraud', 'isFlaggedFraud']

X_train = train_df.drop(columns=drop_cols)
```

```python
y_train = train_df['isFraud']

X_val = val_df.drop(columns=drop_cols)
y_val = val_df['isFraud']

X_test = test_df.drop(columns=drop_cols)
y_test = test_df['isFraud']

# STEP 4 — Logistic Regression Baseline

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import average_precision_score

model = LogisticRegression(max_iter=1000, class_weight='balanced',
n_jobs=-1)
model.fit(X_train, y_train)

val_probs = model.predict_proba(X_val)[:, 1]

pr_auc = average_precision_score(y_val, val_probs)
print("Validation PR-AUC:", pr_auc)
```

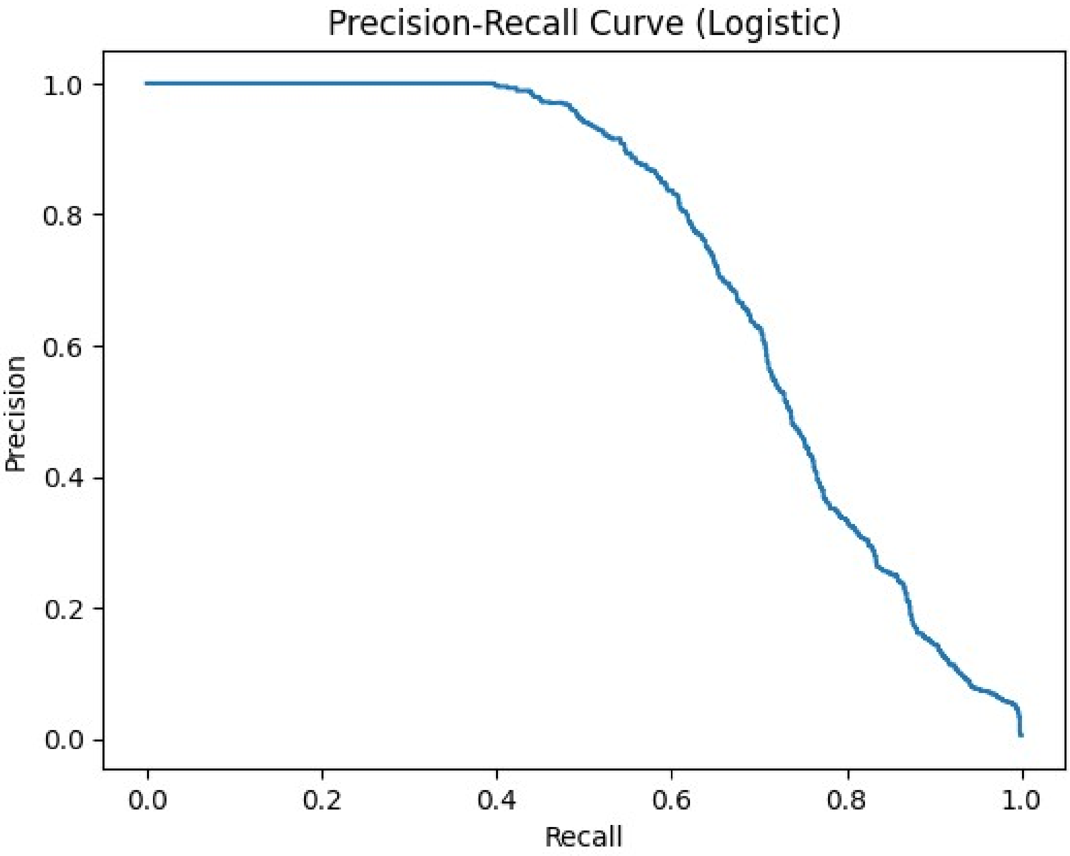
```
Validation PR-AUC: 0.7386225254621741
```

```python
# STEP 5 — Precision-Recall Curve

from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

precision, recall, _ = precision_recall_curve(y_val, val_probs)

plt.plot(recall, precision)
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve (Logistic)")
plt.show()
```



```
# WEEK 2.5 — DIAGNOSTIC EXPERIMENT (Critical)

# STEP 1 — Balance Columns Remove

balance_cols = [
    'oldbalanceOrg',
    'newbalanceOrig',
    'oldbalanceDest',
    'newbalanceDest'
]

X_train_diag = X_train.drop(columns=balance_cols)
X_val_diag = X_val.drop(columns=balance_cols)
X_test_diag = X_test.drop(columns=balance_cols)

# STEP 2 — Logistic Dobara Train

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import average_precision_score

diag_model = LogisticRegression(max_iter=1000,
class_weight='balanced', n_jobs=-1)
diag_model.fit(X_train_diag, y_train)
```

```python
val_probs_diag = diag_model.predict_proba(X_val_diag)[:, 1]

pr_auc_diag = average_precision_score(y_val, val_probs_diag)
print("Validation PR-AUC (No Balance Features):", pr_auc_diag)
```

```
Validation PR-AUC (No Balance Features): 0.07248820815534474
```

```python
# (Week 2 Final Step)
```

```python
# Step 1 – Clean Feature Set Define
```

```python
clean_drop_cols = [
    'nameOrig',
    'nameDest',
    'isFraud',
    'isFlaggedFraud',
    'oldbalanceOrg',
    'newbalanceOrig',
    'oldbalanceDest',
    'newbalanceDest'
]

X_train_clean = train_df.drop(columns=clean_drop_cols)
y_train = train_df['isFraud']

X_val_clean = val_df.drop(columns=clean_drop_cols)
y_val = val_df['isFraud']

X_test_clean = test_df.drop(columns=clean_drop_cols)
y_test = test_df['isFraud']
```

```python
# Step 2 – XGBoost Train
```

```python
!pip install xgboost
```

```
Requirement already satisfied: xgboost in
/usr/local/lib/python3.12/dist-packages (3.2.0)
Requirement already satisfied: numpy in
/usr/local/lib/python3.12/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.12/dist-packages (from xgboost) (2.27.5)
Requirement already satisfied: scipy in
/usr/local/lib/python3.12/dist-packages (from xgboost) (1.16.3)
```

```python
from xgboost import XGBClassifier
from sklearn.metrics import average_precision_score

xgb_model = XGBClassifier(
    n_estimators=200,
    max_depth=6,
    learning_rate=0.1,
    scale_pos_weight = (len(y_train) - y_train.sum()) / y_train.sum(),
    tree_method='hist',
    n_jobs=-1
)

xgb_model.fit(X_train_clean, y_train)

val_probs_xgb = xgb_model.predict_proba(X_val_clean)[:, 1]

pr_auc_xgb = average_precision_score(y_val, val_probs_xgb)

print("Validation PR-AUC (XGBoost Clean):", pr_auc_xgb)
```

```
Validation PR-AUC (XGBoost Clean): 0.043466269007156166
```

```
# WEEK 3 — ADVANCED BEHAVIORAL FEATURES

# Step 1 — Time Since Last Transaction (Per nameOrig)

# Make sure sorted
train_df = train_df.sort_values(['nameOrig', 'step'])
val_df = val_df.sort_values(['nameOrig', 'step'])
test_df = test_df.sort_values(['nameOrig', 'step'])

# Time since last transaction
train_df['prev_step'] = train_df.groupby('nameOrig')['step'].shift(1)
train_df['time_delta'] = train_df['step'] - train_df['prev_step']

# First transaction ke liye fill
train_df['time_delta'] = train_df['time_delta'].fillna(-1)

# Val/Test me leakage-safe logic

last_train_step = train_df.groupby('nameOrig')
['step'].max().rename("last_train_step")

# Merge into val:

val_df = val_df.merge(last_train_step, on='nameOrig', how='left')

val_df['time_delta'] = val_df['step'] - val_df['last_train_step']
val_df['time_delta'] = val_df['time_delta'].fillna(-1)

# Same for test:

test_df = test_df.merge(last_train_step, on='nameOrig', how='left')

test_df['time_delta'] = test_df['step'] - test_df['last_train_step']
test_df['time_delta'] = test_df['time_delta'].fillna(-1)

print(train_df[['time_delta','isFraud']].describe())
```

```
         time_delta       isFraud
count  6.061807e+06  6.061807e+06
```

```
mean  -8.056085e-01  9.173832e-04
std    6.419069e+00  3.027444e-02
min   -1.000000e+00  0.000000e+00
25%   -1.000000e+00  0.000000e+00
50%   -1.000000e+00  0.000000e+00
75%   -1.000000e+00  0.000000e+00
max    4.870000e+02  1.000000e+00
```

```
train_df.groupby('isFraud')['time_delta'].mean()
```

```
isFraud
0   -0.805829
1   -0.565006
Name: time_delta, dtype: float64
```

```
# STEP 2 — Destination Transaction Count (Train Only)

# Destination transaction count (train only)
dest_tx_count =
train_df.groupby('nameDest').size().rename("dest_tx_count")

train_df = train_df.merge(dest_tx_count, on='nameDest', how='left')
val_df = val_df.merge(dest_tx_count, on='nameDest', how='left')
test_df = test_df.merge(dest_tx_count, on='nameDest', how='left')

# Cold start handling
for df_ in [train_df, val_df, test_df]:
    df_['dest_tx_count'] = df_['dest_tx_count'].fillna(0)

# STEP 3 — Destination Fraud Rate (Train Only)

dest_fraud_rate = train_df.groupby('nameDest')
['isFraud'].mean().rename("dest_fraud_rate")

train_df = train_df.merge(dest_fraud_rate, on='nameDest', how='left')
val_df = val_df.merge(dest_fraud_rate, on='nameDest', how='left')
test_df = test_df.merge(dest_fraud_rate, on='nameDest', how='left')

for df_ in [train_df, val_df, test_df]:
    df_['dest_fraud_rate'] = df_['dest_fraud_rate'].fillna(0)

# STEP 4 — Destination Velocity (Time-Aware)

# Train
train_df['dest_prev_step'] = train_df.groupby('nameDest')
['step'].shift(1)
train_df['dest_time_delta'] = train_df['step'] -
train_df['dest_prev_step']
train_df['dest_time_delta'] = train_df['dest_time_delta'].fillna(-1)

last_train_dest_step = train_df.groupby('nameDest')
['step'].max().rename("last_train_dest_step")
```

```
val_df = val_df.merge(last_train_dest_step, on='nameDest', how='left')
val_df['dest_time_delta'] = val_df['step'] -
val_df['last_train_dest_step']
val_df['dest_time_delta'] = val_df['dest_time_delta'].fillna(-1)

test_df = test_df.merge(last_train_dest_step, on='nameDest',
how='left')
test_df['dest_time_delta'] = test_df['step'] -
test_df['last_train_dest_step']
test_df['dest_time_delta'] = test_df['dest_time_delta'].fillna(-1)

# Quick Sanity Check

train_df.groupby('isFraud')
[['dest_tx_count','dest_fraud_rate','dest_time_delta']].mean()
```

```
{"summary":"{\n  \"name\": \"train_df\",\n  \"rows\": 2,\n
\"fields\": [\n    {\n      \"column\": \"isFraud\",\n
\"properties\": {\n        \"dtype\": \"int32\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          1,\n
0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"dest_tx_count\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.9991318986435913,\n        \"min\":
9.788347419528861,\n        \"max\": 11.201333301190209,\n
\"num_unique_values\": 2,\n        \"samples\": [\n
9.788347419528861,\n          11.201333301190209\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"dest_fraud_rate\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.2515840155912176,\n        \"min\": 0.0005909842059913715,\n
\"max\": 0.3563845111313756,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.3563845111313756,\n
0.0005909842059913715\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"dest_time_delta\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 23.503772446415123,\n        \"min\": -
33.631720913504765,\n        \"max\": -0.39236715285343426,\n
\"num_unique_values\": 2,\n        \"samples\": [\n          -
33.631720913504765,\n          -0.39236715285343426\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```

```
# STEP 5 — Rebuild Clean Feature Matrix

final_drop_cols = [
    'nameOrig',
    'nameDest',
    'isFraud',
    'isFlaggedFraud',
```

```python
    'oldbalanceOrg',
    'newbalanceOrig',
    'oldbalanceDest',
    'newbalanceDest',
    'prev_step',
    'dest_prev_step',
    'last_train_step',
    'last_train_dest_step'
]

X_train_final = train_df.drop(columns=final_drop_cols,
errors='ignore')
X_val_final = val_df.drop(columns=final_drop_cols, errors='ignore')

X_val_final = X_val_final.reindex(columns=X_train_final.columns,
fill_value=0)

# STEP 6 — XGBoost Re-Run

from xgboost import XGBClassifier
from sklearn.metrics import average_precision_score

xgb_model2 = XGBClassifier(
    n_estimators=300,
    max_depth=6,
    learning_rate=0.05,
    scale_pos_weight=(len(y_train) - y_train.sum()) / y_train.sum(),
    tree_method='hist',
    n_jobs=-1
)

xgb_model2.fit(X_train_final, y_train)

val_probs2 = xgb_model2.predict_proba(X_val_final)[:, 1]
pr_auc2 = average_precision_score(y_val, val_probs2)

print("Validation PR-AUC (With Destination Features):", pr_auc2)
```

```
Validation PR-AUC (With Destination Features): 0.010323163129808157
```

```python
# WEEK 4 — SYNTHETIC FRAUD RING INJECTION

# STEP 1 — Fraud Subset Identify

fraud_train = train_df[train_df['isFraud'] == 1].copy()
nonfraud_train = train_df[train_df['isFraud'] == 0].copy()

print("Original Fraud Count:", len(fraud_train))
```

```
Original Fraud Count: 5561
```

```python
# STEP 2 — Select Injection Seed Accounts
```

```python
import numpy as np

np.random.seed(42)

seed_frauds = fraud_train.sample(200).copy()

# STEP 3 — Create Synthetic Mule Destinations

num_mules = 20

mule_accounts = [f"SYN_MULE_{i}" for i in range(num_mules)]

# STEP 4 — Inject Multi-Origin Fraud Ring

synthetic_rows = []

for i, mule in enumerate(mule_accounts):
    origins_subset = seed_frauds.iloc[i*10:(i+1)*10]

    for _, row in origins_subset.iterrows():
        new_row = row.copy()
        new_row['nameDest'] = mule
        new_row['amount'] = row['amount'] * 1.1
        new_row['step'] = row['step'] + 1  # temporal burst

        synthetic_rows.append(new_row)

synthetic_df = pd.DataFrame(synthetic_rows)

# STEP 5 — Append to Train

train_df_aug = pd.concat([train_df, synthetic_df], ignore_index=True)

print("Train size before:", len(train_df))
print("Train size after injection:", len(train_df_aug))
```

```
Train size before: 6061807
Train size after injection: 6062007
```

```python
# STEP 6 — Recompute Destination Features

train_df = train_df_aug.copy()

# Destination transaction count
dest_tx_count =
train_df.groupby('nameDest').size().rename("dest_tx_count")

train_df = train_df.merge(dest_tx_count, on='nameDest', how='left')
val_df = val_df.merge(dest_tx_count, on='nameDest', how='left')
test_df = test_df.merge(dest_tx_count, on='nameDest', how='left')
```

```
for df_ in [train_df, val_df, test_df]:
    df_['dest_tx_count'] = df_['dest_tx_count'].fillna(0)

# Destination Fraud Rate Recompute

dest_fraud_rate = train_df.groupby('nameDest')
['isFraud'].mean().rename("dest_fraud_rate")

train_df = train_df.merge(dest_fraud_rate, on='nameDest', how='left')
val_df = val_df.merge(dest_fraud_rate, on='nameDest', how='left')
test_df = test_df.merge(dest_fraud_rate, on='nameDest', how='left')

for df_ in [train_df, val_df, test_df]:
    df_['dest_fraud_rate'] = df_['dest_fraud_rate'].fillna(0)

# Important Observation

train_df[train_df['nameDest'].str.contains("SYN_MULE")][
    ['dest_tx_count','dest_fraud_rate']
].head()
```

```
{"summary":"{\n  \"name\": \"]\",\n  \"rows\": 5,\n  \"fields\": [\n
{\n      \"column\": \"dest_tx_count\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0,\n        \"min\": 10,\n
\"max\": 10,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          10\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"dest_fraud_rate\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 1.0,\n
\"max\": 1.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```

```
# Rebuild Baseline After Injection

# Final Feature Builder

def build_feature_matrix(df, feature_cols):
    """
    Returns feature matrix aligned to given feature_cols.
    """
    X = df[feature_cols].copy()
    return X

# STEP 2 — Define Drop Columns (Helper + Leakage + Balance)

final_drop_cols = [
    'nameOrig',
    'nameDest',
    'isFraud',
    'isFlaggedFraud',
    'oldbalanceOrg',
```

```python
    'newbalanceOrig',
    'oldbalanceDest',
    'newbalanceDest',
    'prev_step',
    'dest_prev_step',
    'last_train_step',
    'last_train_dest_step'
]

# STEP 3 – Build Clean Train Matrix

# Drop unwanted columns
train_model_df = train_df.drop(columns=final_drop_cols,
errors='ignore')
val_model_df   = val_df.drop(columns=final_drop_cols, errors='ignore')

# Separate labels
y_train = train_df['isFraud'].values
y_val   = val_df['isFraud'].values

# Save feature list from train only
feature_cols = train_model_df.columns.tolist()

# Align validation to train
val_model_df = val_model_df.reindex(columns=feature_cols,
fill_value=0)

# Convert to numpy (memory + speed)
X_train_final = train_model_df.values
X_val_final   = val_model_df.values

# STEP 4 – Optimized XGBoost Config (Colab Friendly)

from xgboost import XGBClassifier
from sklearn.metrics import average_precision_score
import numpy as np

scale_weight = (len(y_train) - np.sum(y_train)) / np.sum(y_train)

xgb_model2 = XGBClassifier(
    n_estimators=400,
    max_depth=5,
    learning_rate=0.05,
    subsample=0.8,
    colsample_bytree=0.8,
    scale_pos_weight=scale_weight,
    tree_method='hist',
    eval_metric='logloss',
    random_state=42,
    n_jobs=-1
)
```

```
xgb_model2.fit(X_train_final, y_train)
```