Synthetic Financial Datasets For Fraud Detection - Project

In this project, I tackle the challenge of fraud detection in mobile-money ecosystems using PaySim - a large, high-fidelity synthetic dataset that mirrors real transaction flows and injected malicious behavior. The dataset can be found at https://www.kaggle.com/datasets/ealaxi/paysim1.

Headers

step - maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).

type - CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.

amount - amount of the transaction in local currency.

nameOrig - customer who started the transaction

oldbalanceOrg - initial balance before the transaction

newbalanceOrig - new balance after the transaction.

nameDest - customer who is the recipient of the transaction

oldbalanceDest - initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).

newbalanceDest - new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).

isFraud - This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers accounts and try to empty the funds by transferring to another account and then cashing out of the system.

isFlaggedFraud - The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction.

NOTE: Transactions which are detected as fraud are cancelled, so for fraud detection these columns (oldbalanceOrg, newbalanceOrig, oldbalanceDest, newbalanceDest) must not be used.

Importing dependencies

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns
from scipy.stats import pearsonr
from scipy.stats import shapiro
from scipy.stats import spearmanr
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from lightgbm.sklearn import LGBMClassifier
import xgboost as xgb
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, roc_auc_score, roc_curve
import joblib
```

Data Extraction and Unpacking

```
import zipfile

with zipfile.ZipFile('Synthetic Financial Datasets For Fraud Detection.zip', 'r') as zip:
    zip.extractall()
```

Reading CSV

```
df_financial = pd.read_csv('PS_20174392719_1491204439457_log.csv')
df_financial.head(5)
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalance
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	

Dataset Overview and Structure

```
df_financial.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   nameOrig        object
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
 10  isFlaggedFraud  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

Data Cleaning - Missing Values and Duplicates

```
print("Missing Values:")
print(df_financial.isnull().sum())
print("Null 'amount' Values:")
print((df_financial['amount'] == 0).sum())
print('Number of Duplicates: ',df_financial.duplicated().sum())
```

```
Missing Values:
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
Null 'amount' Values:
16
Number of Duplicates:  0
```

Exploratory Data Analysis (EDA)

Let's take a look at the 16 transactions with null 'amount' values.

```
df_financial.loc[df_financial['amount'] == 0]
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbal
2736447	212	CASH_OUT	0.0	C1510987794	0.0	0.0	C1696624817	0.00	
3247298	250	CASH_OUT	0.0	C521393327	0.0	0.0	C480398193	0.00	
3760289	279	CASH_OUT	0.0	C539112012	0.0	0.0	C1106468520	538547.63	[illegible]
5563714	387	CASH_OUT	0.0	C1294472700	0.0	0.0	C1325541393	7970766.57	7
5996408	425	CASH_OUT	0.0	C832555372	0.0	0.0	C1462759334	76759.90	
5996410	425	CASH_OUT	0.0	C69493310	0.0	0.0	C719711728	2921531.34	2
6168500	554	CASH_OUT	0.0	C10965156	0.0	0.0	C1493336195	230289.66	[illegible]
6205440	586	CASH_OUT	0.0	C1303719003	0.0	0.0	C900608348	1328472.86	1
6266414	617	CASH_OUT	0.0	C1971175979	0.0	0.0	C1352345416	0.00	
6281483	646	CASH_OUT	0.0	C2060908932	0.0	0.0	C1587892888	0.00	
6281485	646	CASH_OUT	0.0	C1997645312	0.0	0.0	C601248796	0.00	
6296015	671	CASH_OUT	0.0	C1960007029	0.0	0.0	C459118517	27938.72	
6351226	702	CASH_OUT	0.0	C1461113533	0.0	0.0	C1382150537	107777.02	
6362461	730	CASH_OUT	0.0	C729003789	0.0	0.0	C1388096959	1008609.53	1
6362463	730	CASH_OUT	0.0	C2088151490	0.0	0.0	C1156763710	0.00	
6362585	741	CASH_OUT	0.0	C312737633	0.0	0.0	C1400061387	267522.87	[illegible]

Insight: Transactions with null 'amount' values are both fraudulent and withdrawals.

But is every fraudulent transaction a withdrawal with a null 'amount'?

```
df_financial.loc[df_financial['isFraud'] == 1]
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	ne
2	1	TRANSFER	181.00	C1305486145	181.00	0.0	C553264065	0.00	
3	1	CASH_OUT	181.00	C840083671	181.00	0.0	C38997010	21182.00	
251	1	TRANSFER	2806.00	C1420196421	2806.00	0.0	C972765878	0.00	
252	1	CASH_OUT	2806.00	C2101527076	2806.00	0.0	C1007251739	26202.00	
680	1	TRANSFER	20128.00	C137533655	20128.00	0.0	C1848415041	0.00	
...	...	...	...	...	...	...	...	...	
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C776919290	0.00	
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C1881841831	0.00	
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C1365125890	68488.84	
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C2080388513	0.00	
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C873221189	6510099.11	

8213 rows × 11 columns

No.

Insight: Not every fraudulent transaction has a null 'amount'.

Let us examine the descriptive statistics of the dataset.

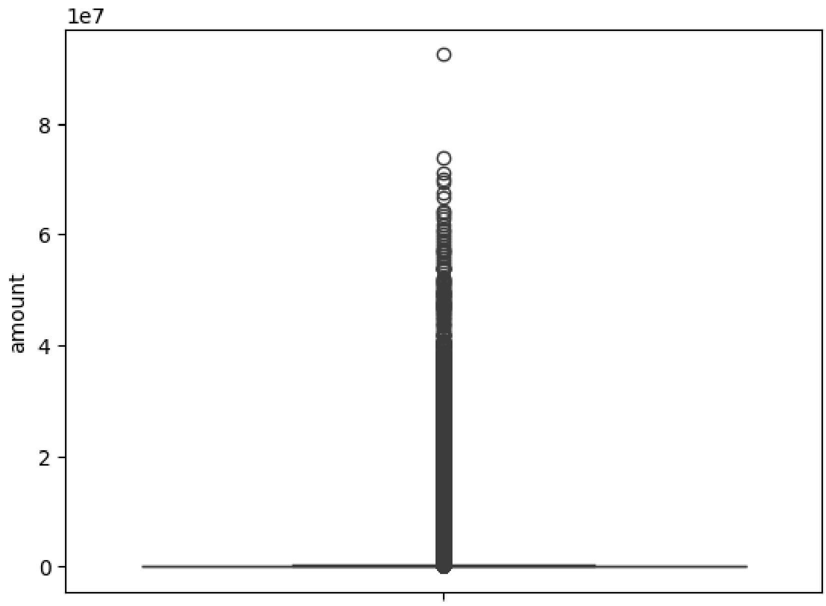
```
df_financial.describe()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isF
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224996e+06	1.290820e-03	2
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+06	3.590480e-02	[illegible]
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146614e+05	0.000000e+00	0
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111909e+06	0.000000e+00	0
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561793e+08	1.000000e+00	1

It seems there are some outliers in the 'amount' statistics, possibly indicating fraudulent transactions. Let's examine the boxplot chart.

```
sns.boxplot(df_financial['amount'])
```

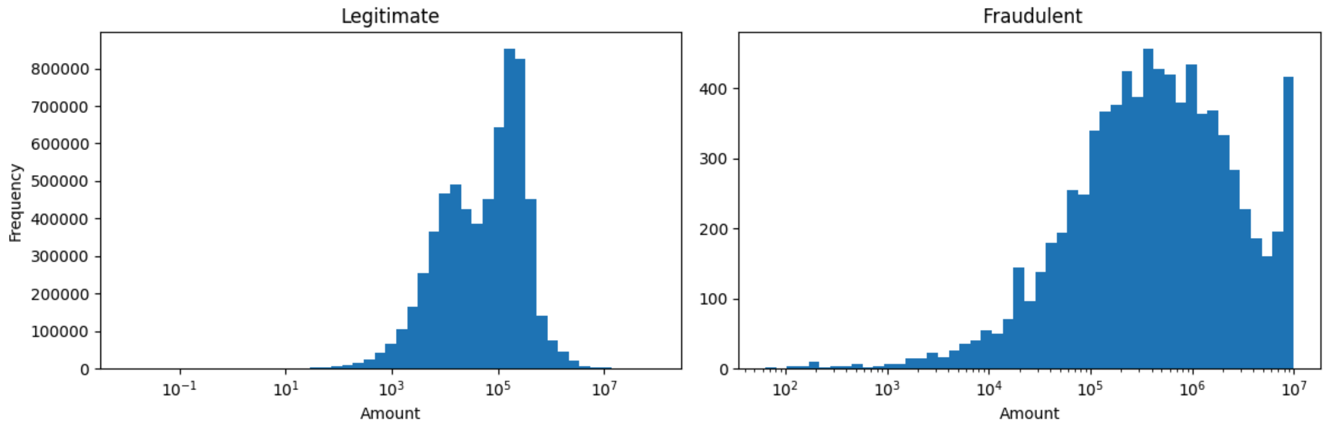
```
<Axes: ylabel='amount'>
```



Since the linear scale is not appropriate, let's plot the 'amount' histogram for both legitimate and fraudulent transactions.

```
# separate values
fraud = df_financial.loc[df_financial['isFraud']==1, 'amount']
nonfraud = df_financial.loc[df_financial['isFraud']==0, 'amount']

# bins (log-scale, values > 0)
bins_nf = np.logspace(np.log10(nonfraud[nonfraud>0].min()),
                      np.log10(nonfraud.max()), 50)
bins_f  = np.logspace(np.log10(fraud[fraud>0].min()),
                      np.log10(fraud.max()), 50)

# plot side-by-side with 1 row and 2 columns
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12,4))

ax1.hist(nonfraud, bins=bins_nf)
ax1.set_xscale('log')
ax1.set_title('Legitimate')
ax1.set_xlabel('Amount')
ax1.set_ylabel('Frequency')
```

```
ax2.hist(fraud, bins=bins_f)
ax2.set_xscale('log')
ax2.set_title(' Fraudulent')
ax2.set_xlabel('Amount')

plt.tight_layout()
plt.show()
```



Insight: Fraudulent transactions tend to have higher values more frequently compared to legitimate transactions.

Let's examine the statistical differences between the two.

```
df_financial.groupby("isFraud").describe()
```

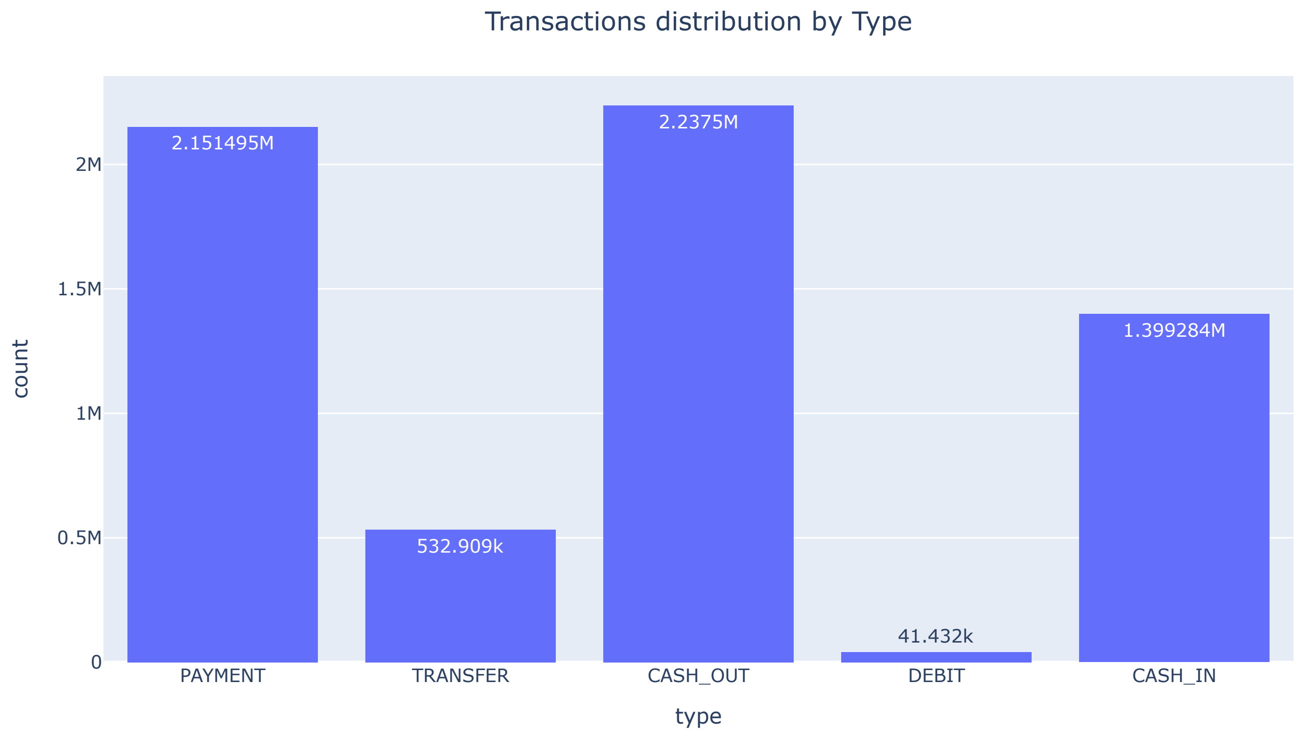
	step								amount		...	newbalanceDe
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%
isFraud												
0	6354407.0	243.235663	142.140194	1.0	156.0	239.0	334.0	718.0	6354407.0	1.781970e+05	...	1111975.345
1	8213.0	368.413856	216.388690	1.0	181.0	367.0	558.0	743.0	8213.0	1.467967e+06	...	1058725.220

2 rows × 56 columns

Insight: There are 6,354,407 legitimate and 8,213 fraudulent transactions.

Examining graphically transactions by type:

```
fig = px.histogram(df_financial, x='type', text_auto=True)
fig.update_layout(
    title='Transactions distribution by Type',
    title_x=0.5
)
fig.show()
```



Breaking down into legitimate and fraudulent transactions:

```
df_financial.loc[df_financial["isFraud"] == 0, 'type'].value_counts()
```

type	count
CASH_OUT	2233384
PAYMENT	2151495
CASH_IN	1399284
TRANSFER	528812
DEBIT	41432

dtype: int64

```
df_financial.loc[df_financial["isFraud"] == 1, 'type'].value_counts()
```

type	count
CASH_OUT	4116
TRANSFER	4097

dtype: int64

Insight: Fraudulent transactions are either withdrawals or transfers, and occur at approximately the same frequency. A clearer view is provided in the crosstab below.

```
fraud_pct = pd.crosstab(
    df_financial['type'],
    df_financial['isFraud'],
    normalize='columns'
) * 100

fraud_pct.columns = ['Legitimate (%)', 'Fraudulent (%)']
```

```
fraud_pct.columns = ['Legitimate (%)', 'Fraudulent (%)']

print(fraud_pct)
```

```
          Legitimate (%)  Fraudulent (%)
type
CASH_IN        22.020686         0.00000
CASH_OUT       35.147009        50.11567
DEBIT           0.652020         0.00000
PAYMENT        33.858313         0.00000
TRANSFER        8.321972        49.88433
```

Let's take a look at fraudulent and withdrawal transactions.

```
df_financial.loc[(df_financial['isFraud'] == 1) & (df_financial['type'] == 'CASH_OUT')]
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	ne
3	1	CASH_OUT	181.00	C840083671	181.00	0.0	C38997010	21182.00	
252	1	CASH_OUT	2806.00	C2101527076	2806.00	0.0	C1007251739	26202.00	
681	1	CASH_OUT	20128.00	C1118430673	20128.00	0.0	C339924917	6268.00	
724	1	CASH_OUT	416001.33	C749981943	0.00	0.0	C667346055	102.00	
970	1	CASH_OUT	1277212.77	C467632528	1277212.77	0.0	C716083600	0.00	
...	...	...	...	...	...	...	...	...	
6362611	742	CASH_OUT	63416.99	C994950684	63416.99	0.0	C1662241365	276433.18	
6362613	743	CASH_OUT	1258818.82	C1436118706	1258818.82	0.0	C1240760502	503464.50	
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C776919290	0.00	
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C1365125890	68488.84	
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C873221189	6510099.11	

4116 rows × 11 columns

In CASH_OUT transactions, the recipient is an external entity (typically an ATM or merchant) whose name always starts with the letter 'M'. Here, we see that some CASH_OUT transactions start with the letter 'C', which technically is not a cash withdrawal — it would be a TRANSFER between customers. Let's check if there are any CASH_OUT transactions, whether fraudulent or legitimate, where the recipient starts with 'M'.

```
df_financial.loc[df_financial.type == 'CASH_OUT'].nameDest.str.contains('M').any()
```

```
np.False_
```

Insight: There is no 'merchant' or ATM as the destination for CASH_OUT transactions, which is incongruent.

Feature Engineering

Let's classify the transactions by type: Customer to Customer (C2C), Customer to Merchant (C2M), Merchant to Customer (M2C), and Merchant to Merchant (M2M).

```
df_financial['trans_direction'] = 'Unknown'
```

```
df_financial.loc[df_financial.nameOrig.str.startswith('C') & df_financial.nameDest.str.startswith('C'), 'trans_directi
df_financial.loc[df_financial.nameOrig.str.startswith('C') & df_financial.nameDest.str.startswith('M'), 'trans_directi
df_financial.loc[df_financial.nameOrig.str.startswith('M') & df_financial.nameDest.str.startswith('C'), 'trans_directi
df_financial.loc[df_financial.nameOrig.str.startswith('M') & df_financial.nameDest.str.startswith('M'), 'trans_directi
df_financial.head(5)
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalance
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	

Removing the origin and destination identifier columns to avoid potential overfitting in the ML model.

```
df_financial.drop(columns = ['nameOrig','nameDest'], axis = 'columns', inplace = True)
```

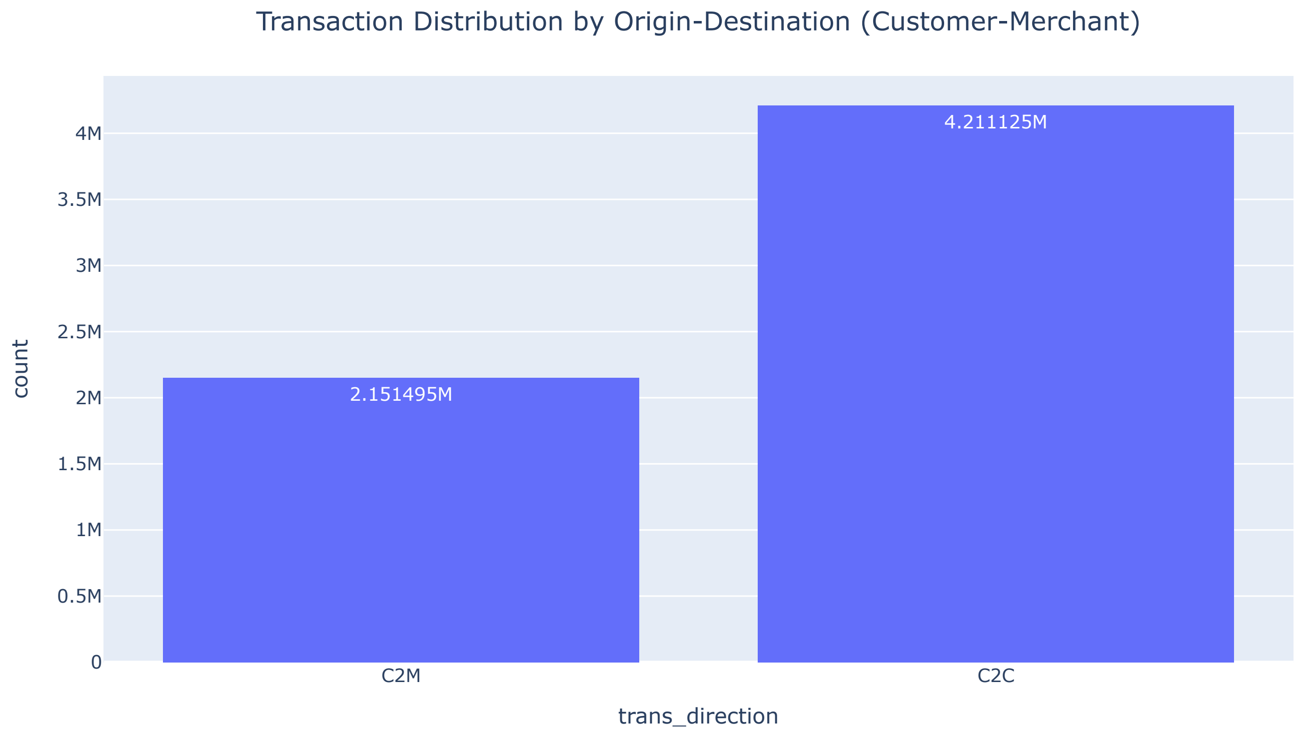
Adjusting the position of the 'trans_direction' column to improve table readability.

```
col = df_financial.pop('trans_direction')

df_financial.insert(loc=1, column='trans_direction', value=col)
df_financial.head(5)
```

	step	trans_direction	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFr
0	1	C2M	PAYMENT	9839.64	170136.0	160296.36	0.0	0.0	
1	1	C2M	PAYMENT	1864.28	21249.0	19384.72	0.0	0.0	
2	1	C2C	TRANSFER	181.00	181.0	0.00	0.0	0.0	
3	1	C2C	CASH_OUT	181.00	181.0	0.00	21182.0	0.0	
4	1	C2M	PAYMENT	11668.14	41554.0	29885.86	0.0	0.0	

Let's look at the number of transactions by type.

```
fig = px.histogram(df_financial, x='trans_direction', text_auto=True)
fig.update_layout(
    title='Transaction Distribution by Origin-Destination (Customer-Merchant)',
    title_x=0.5
)
fig.show()
```



Evaluating the amount of fraudulent and legitimate transactions by origin-destination type.

```
print('Number of fraud transactions according to trans_direction:\n', df_financial.loc[df_financial['isFraud'] == 1, 'tı
print('Number of legitimate transactions according to trans_direction:\n', df_financial.loc[df_financial['isFraud'] == (
```

```
Number of fraud transactions according to trans_direction:
 trans_direction
C2C    8213
Name: count, dtype: int64

Number of legitimate transactions according to trans_direction:
 trans_direction
C2C    4202912
C2M    2151495
Name: count, dtype: int64
```

Insight: Fraudulent transactions are either withdrawals or transfers and occur between Customer-to-Customer (C2C) accounts.

When a transaction is identified as fraud in the simulated system, it is not actually completed and is immediately canceled. Therefore, the balances before and after the transaction (the columns oldbalanceOrg, newbalanceOrig, oldbalanceDest and newbalanceDest) do not reflect any real movement for these cases. Therefore, to avoid data leakage, we will exclude these columns.

```
df_financial.drop(columns = ['oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest'], axis = 'columns', in
df_financial.head(5)
```

	step	trans_direction	type	amount	isFraud	isFlaggedFraud
0	1	C2M	PAYMENT	9839.64	0	0
1	1	C2M	PAYMENT	1864.28	0	0
2	1	C2C	TRANSFER	181.00	1	0
3	1	C2C	CASH_OUT	181.00	1	0
4	1	C2M	PAYMENT	11668.14	0	0

Since categorical data does not have an inherent ordinal relationship, we will use one-hot encoding.

```
df_financial = pd.get_dummies(df_financial, prefix = ['trans_direction', 'type'], drop_first = True)
df_financial.head(5)
```

	step	amount	isFraud	isFlaggedFraud	trans_direction_C2M	type_CASH_OUT	type_DEBIT	type_PAYMENT	type_TRAN
0	1	9839.64	0	0	True	False	False	True	F
1	1	1864.28	0	0	True	False	False	True	F
2	1	181.00	1	0	False	False	False	False	
3	1	181.00	1	0	False	True	False	False	F
4	1	11668.14	0	0	True	False	False	True	F

Splitting the dataset into training and testing parts.

```
x = df_financial.drop('isFraud', axis=1)
y = df_financial.isFraud

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.30, stratify = df_financial.isFraud)
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

Model building

We will now train four machine learning models and then add each one to a list.

```
rfc = RandomForestClassifier(
    n_estimators=15,    # number of trees in the forest
    n_jobs=-1,          # uses all available CPU cores
    random_state=42     # seed for reproducibility
)
lgbm = LGBMClassifier(
    boosting_type='gbdt',  # Gradient Boosting Decision Tree (LightGBM default)
    objective='binary',    # binary classification problem
    random_state=8888      # different seed, to vary the pseudo-random
)
xgbr = xgb.XGBClassifier(
    max_depth=3,         # maximum depth of each tree
    n_jobs=-1,           # uses all cores for training
    random_state=42,     # reproducibility
    learning_rate=0.1    # boosting learning rate (shrinkage)
)
logreg = LogisticRegression(
    solver='liblinear',  # optimization algorithm (good for smaller datasets)
    random_state=42      # seed for regularization/algorithm
)

rfc.fit(x_train, y_train)
lgbm.fit(x_train, y_train)
xgbr.fit(x_train, y_train)
logreg.fit(x_train, y_train)

classifiers = []
classifiers.append(rfc)
classifiers.append(lgbm)
classifiers.append(xgbr)
classifiers.append(logreg)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.

[LightGBM] [Info] Number of positive: 5749, number of negative: 4448085
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.221884 seconds.
```

```
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 527
[LightGBM] [Info] Number of data points in the train set: 4453834, number of used features: 8
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001291 -> initscore=-6.651203
[LightGBM] [Info] Start training from score -6.651203
```

Model Evaluation

```
results = []
for clf in classifiers:
    name = clf.__class__.__name__
    # class and probability prediction
    y_pred = clf.predict(x_test)
    y_proba = clf.predict_proba(x_test)[:,1]

    # metrics
    print(f"\n=== {name} ===")
    print(classification_report(y_test, y_pred, digits=4))
    auc = roc_auc_score(y_test, y_proba)
    print(f"ROC AUC: {auc:.4f}")

    results.append((name, auc))

# 2) ROC curves side by side
plt.figure(figsize=(8,6))
for clf in classifiers:
    y_proba = clf.predict_proba(x_test)[:,1]
    fpr, tpr, _ = roc_curve(y_test, y_proba)
    auc = roc_auc_score(y_test, y_proba)
    plt.plot(fpr, tpr, label=f"{clf.__class__.__name__} (AUC = {auc:.2f})")

plt.plot([0,1],[0,1], 'k--', lw=1)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves Comparativas")
plt.legend(loc="lower right")
plt.tight_layout()
plt.show()

# 3) Identify and save the best model
best_name, best_auc = max(results, key=lambda x: x[1])
best_model = next(m for m in classifiers if m.__class__.__name__ == best_name)
print(f"\nBest model: {best_name} with AUC = {best_auc:.4f}")
joblib.dump(best_model, "best_fraud_model.pkl")
```

```
=== RandomForestClassifier ===
              precision    recall  f1-score   support

           0     0.9992    0.9998    0.9995   1906322
           1     0.7203    0.3470    0.4684      2464

    accuracy                         0.9990   1908786
   macro avg     0.8597    0.6734    0.7339   1908786
weighted avg     0.9988    0.9990    0.9988   1908786

ROC AUC: 0.7936
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.

/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.


=== LGBMClassifier ===
              precision    recall  f1-score   support

           0     0.9990    0.9998    0.9994   1906322
           1     0.5860    0.1976    0.2956      2464

    accuracy                         0.9988   1908786
   macro avg     0.7925    0.5987    0.6475   1908786
weighted avg     0.9984    0.9988    0.9985   1908786

ROC AUC: 0.8855

=== XGBClassifier ===
              precision    recall  f1-score   support

           0     0.9989    1.0000    0.9995   1906322
           1     0.9227    0.1745    0.2935      2464

    accuracy                         0.9989   1908786
   macro avg     0.9608    0.5872    0.6465   1908786
weighted avg     0.9988    0.9989    0.9985   1908786

ROC AUC: 0.9556

=== LogisticRegression ===
              precision    recall  f1-score   support

           0     0.9987    1.0000    0.9993   1906322
           1     0.1818    0.0024    0.0048      2464

    accuracy                         0.9987   1908786
   macro avg     0.5903    0.5012    0.5021   1908786
weighted avg     0.9977    0.9987    0.9981   1908786

ROC AUC: 0.[illegible]
```