🛡️ Proactive Fraud Detection in Financial Transactions using Machine Learning

Importing Libraries

In [1]:
```
import pandas
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Suppress warnings and set plot style
import warnings
warnings.filterwarnings("ignore")
sns.set(style = "whitegrid")
```

Loading Data

In [3]:
```
df = pd.read_csv("Fraud.csv")
```

Adding flagged fraud based on business rule

In [4]:
```
# Flag fraud where type == 'TRANSFER' and amount > 200
df["isFlaggedFraud"] = df.apply(
    lambda row: 1 if row["type"] == "TRANSFER" and row["amount"] > 200 else 0,
    axis=1
)
```

Exploratory Analysis

In [5]:
```
df.head()
```

Out[5]:

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

In [6]:
```
df.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
```

In [7]:
```
df.columns
```

Out[7]:
```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud',
       'isFlaggedFraud'],
      dtype='object')
```

In [8]:
```
df["isFraud"].value_counts()
```

Out[8]:
```
isFraud
0    6354407
1       8213
Name: count, dtype: int64
```

In [9]:
```
df["isFlaggedFraud"].value_counts()
```

Out[9]:
```
isFlaggedFraud
0    5829829
1     532791
Name: count, dtype: int64
```

In [10]:
```
df.isnull().sum().sum()
```

Out[10]:
```
np.int64(0)
```

In [11]:
```
df.shape[0]
```

Out[11]:
```
6362620
```

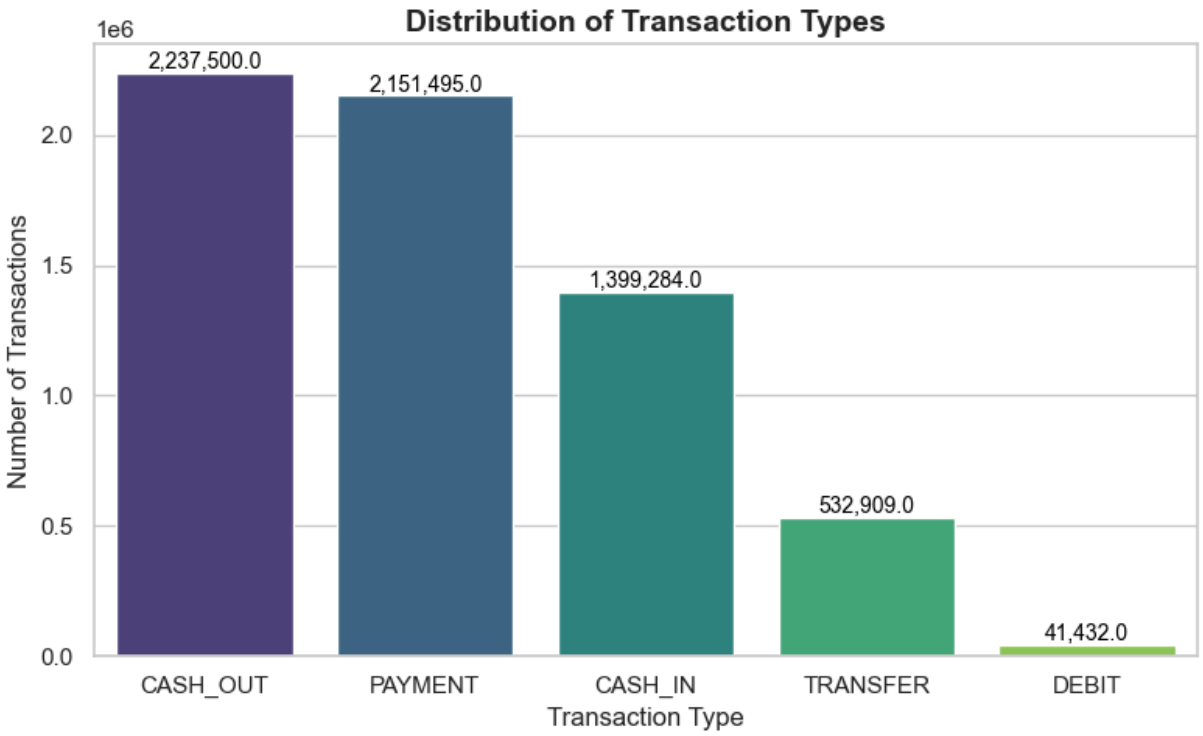
In [12]:
```
round((df["isFraud"].value_counts()[1]/df.shape[0])*100,2)
```

Out[12]:
```
np.float64(0.13)
```

Visualization

Transaction type count

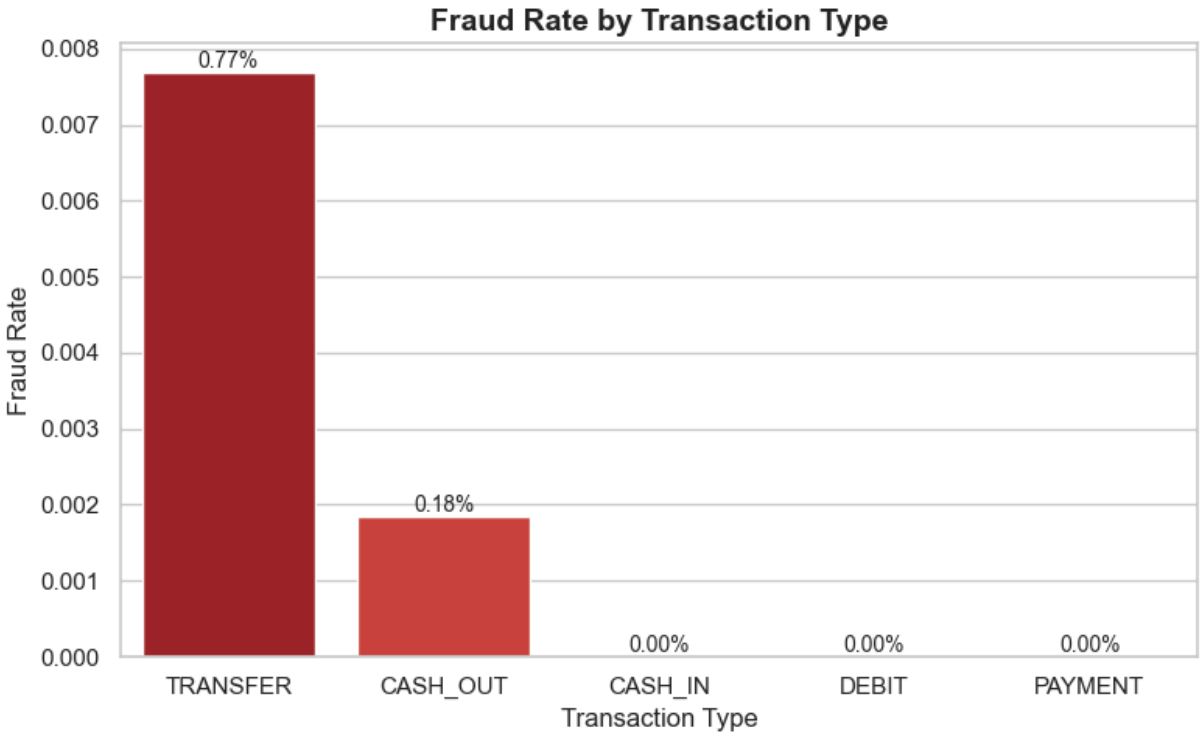
In [13]:
```
sns.set(style="whitegrid")
plt.figure(figsize=(8, 5))

# Plot with seaborn
sns.countplot(data=df, x="type", order=df["type"].value_counts().index, palette="vi
plt.title("Distribution of Transaction Types", fontsize=14, fontweight='bold')
plt.xlabel("Transaction Type", fontsize=12)
plt.ylabel("Number of Transactions", fontsize=12)

# Annotate bars with values
for p in plt.gca().patches:
    height = p.get_height()
    plt.gca().annotate(f'{height:,}', (p.get_x() + p.get_width() / 2., height),
                       ha='center', va='bottom', fontsize=10, color='black')

# Show plot
plt.tight_layout()
plt.show()
```



Fraud rate by type

In [14]:
```
fraud_by_type = df.groupby("type")["isFraud"].mean().sort_values(ascending=False)

# Convert to DataFrame for seaborn
fraud_df = fraud_by_type.reset_index()

sns.set(style="whitegrid")
plt.figure(figsize=(8, 5))
```

```
# Barplot with seaborn
sns.barplot(data=fraud_df, x ="type", y= "isFraud", palette= "Reds_r")
plt.title("Fraud Rate by Transaction Type", fontsize =14, fontweight= 'bold')
plt.xlabel("Transaction Type", fontsize=12)
plt.ylabel("Fraud Rate", fontsize=12)

# Annotate values on bars
for p in plt.gca().patches:
    height = p.get_height()
    plt.gca().annotate(f'{height:.2%}',
                       (p.get_x() + p.get_width() / 2., height),
                       ha='center', va='bottom', fontsize=10)

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



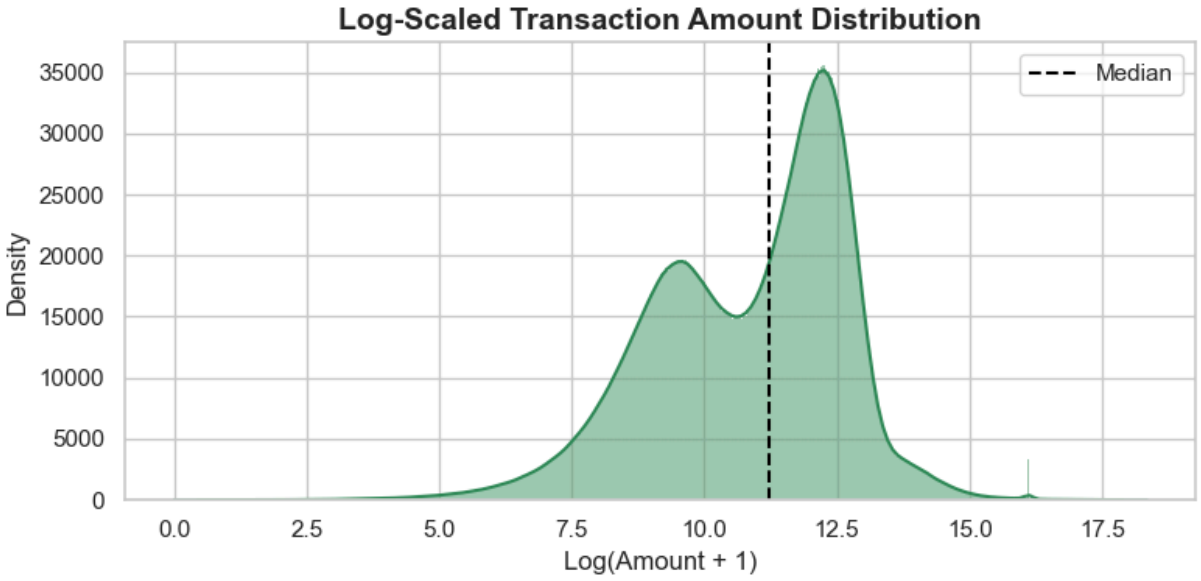
In [15]:
```
df["amount"].describe().astype(int)
```

Out[15]:
```
count     6362620
mean       179861
std        603858
min             0
25%         13389
50%         74871
75%        208721
max      92445516
Name: amount, dtype: int64
```

Outliers

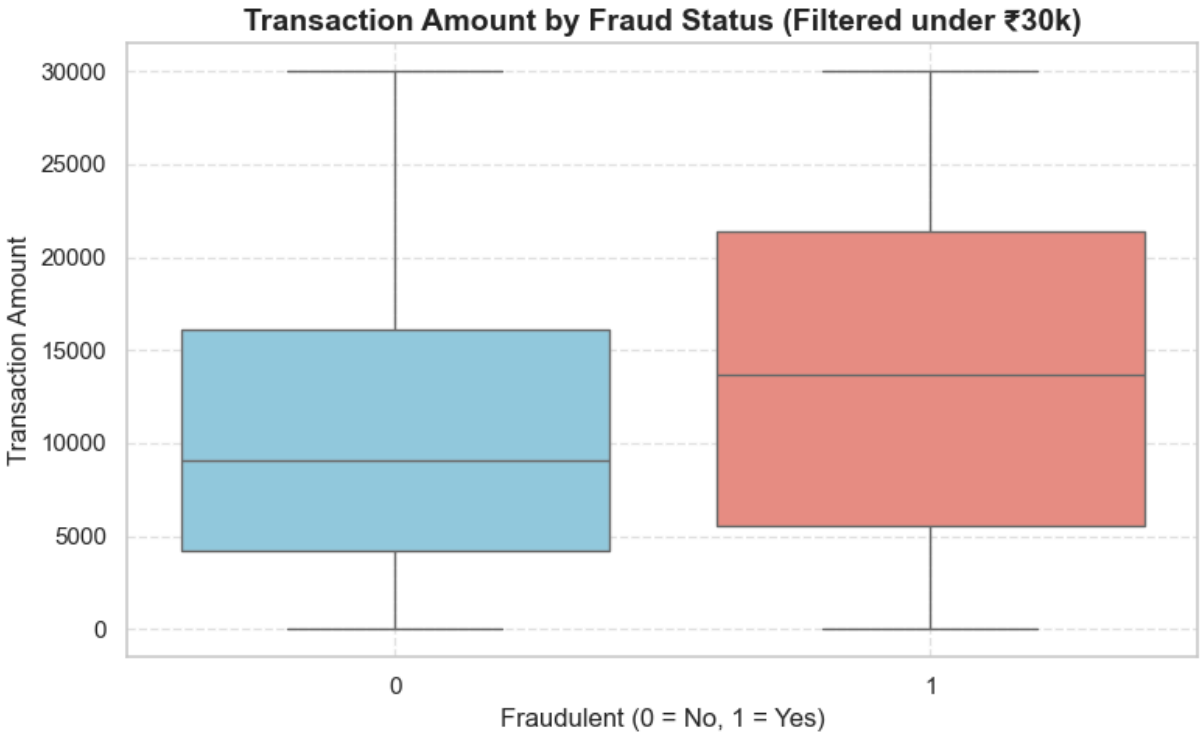
In [16]:
```
import numpy as np
sns.set(style="whitegrid")
```

```
plt.figure(figsize=(8, 4))

#histogram
sns.histplot(np.log1p(df["amount"]), bins=1000, kde=True, color = "seagreen", edgec
plt.title("Log-Scaled Transaction Amount Distribution", fontsize=14, fontweight='bo
plt.xlabel("Log(Amount + 1)", fontsize=12)
plt.ylabel("Density", fontsize=12)
median_log = np.log1p(df["amount"].median())
plt.axvline(median_log, color='black', linestyle='--', linewidth=1.5, label='Median
plt.legend()

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



Boxplot of fraud vs amount (filtered under 30,000)

In [17]:
```
plt.figure(figsize=(8, 5))
sns.set(style="whitegrid")

# Boxplot
sns.boxplot(data=df[df["amount"] < 30000], x="isFraud", y="amount", palette=["skybl
plt.title("Transaction Amount by Fraud Status (Filtered under ₹30k)", fontsize=14,
plt.xlabel("Fraudulent (0 = No, 1 = Yes)", fontsize=12)
plt.ylabel("Transaction Amount", fontsize=12)

plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()

plt.show()
```



Feature Engineering

In [18]:
```
df["balanceDiffOrig"] = df ["oldbalanceOrg"] - df["newbalanceOrig"]
df["balanceDiffDest"] = df["newbalanceDest"] =  df["oldbalanceDest"]
```

In [19]:
```
(df["balanceDiffOrig"] <0).sum()
```

Out[19]: np.int64(1399253)

In [20]:
```
(df["balanceDiffDest"] <0).sum()
```

Out[20]: np.int64(0)

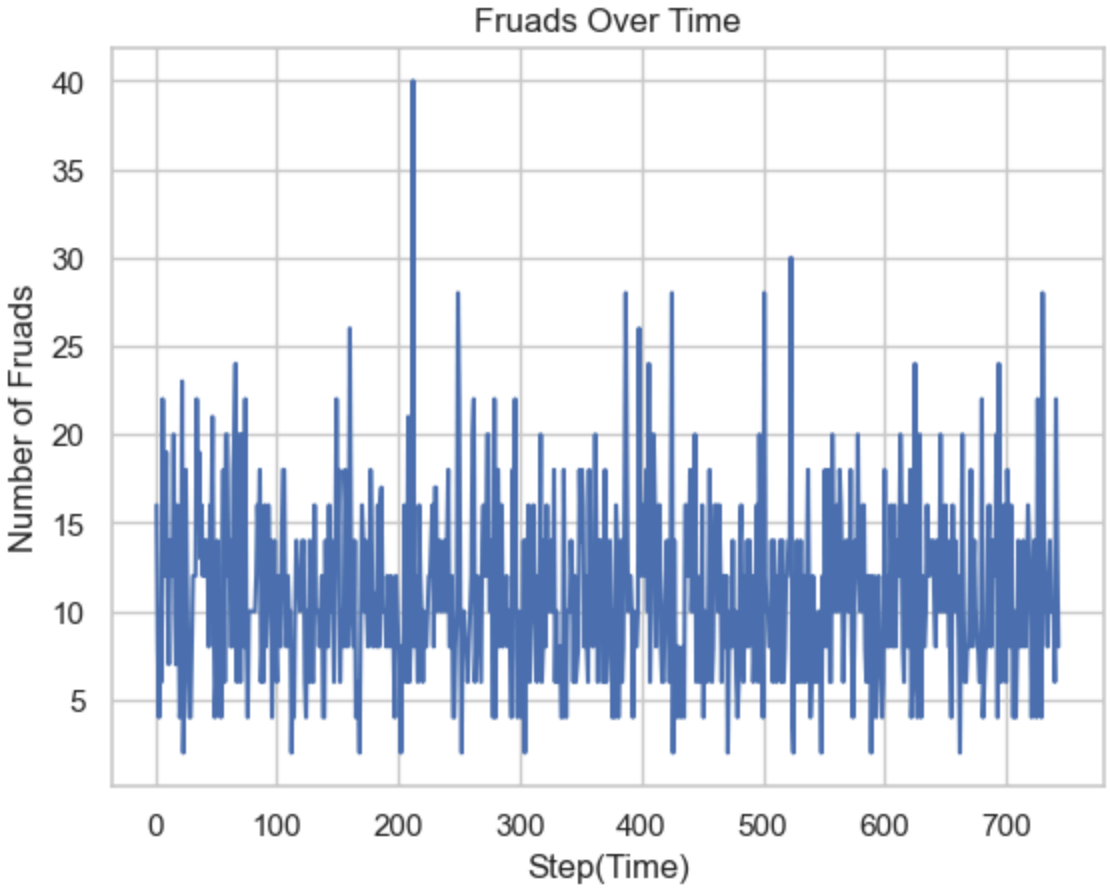
In [21]:
```
df.head(5)
```

Out[21]:

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

Fraud Over Time

In [22]:
```
frauds_per_step = df[df["isFraud"] ==1]["step"].value_counts().sort_index()
plt.plot(frauds_per_step.index, frauds_per_step.values, label = "Frauds per Step")
plt.xlabel("Step(Time)")
plt.ylabel("Number of Fruads")
plt.title("Fruads Over Time")
plt.grid(True)
plt.show()
```



In [23]:
```
df.drop(columns = "step",inplace=True)
```

In [24]:
```
df.head()
```

Out[24]:

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

In [25]:
```
top_senders = df["nameOrig"].value_counts().head(10)
```

In [26]:
```
print("Top Senders:\n", top_senders)
```

```
Top Senders:
 nameOrig
C1677795071    3
C1999539787    3
C724452879     3
C1976208114    3
C400299098     3
C1784010646    3
C1530544995    3
C1065307291    3
C545315117     3
C1902386530    3
Name: count, dtype: int64
```

In [27]:
```
top_receivers = df["nameDest"].value_counts().head(10)
```

In [28]:
```
print("Top Receivers:\n", top_receivers)
```

```
Top Receivers:
 nameDest
C1286084959    113
C985934102     109
C665576141     105
C2083562754    102
C248609774     101
C1590550415    101
C1789550256     99
C451111351      99
C1360767589     98
C1023714065     97
Name: count, dtype: int64
```

In [29]:
```
fraud_users = df[df["isFraud"]==1] ["nameOrig"].value_counts().head(10)
```

In [30]:
```
print("Top Fraudulent Users:\n", fraud_users)
```

```
Top Fraudulent Users:
 nameOrig
C1280323807    1
C1305486145    1
C840083671     1
C1420196421    1
C2101527076    1
C1039979813    1
C2089752665    1
C1614818636    1
C40604503      1
C1970706589    1
Name: count, dtype: int64
```

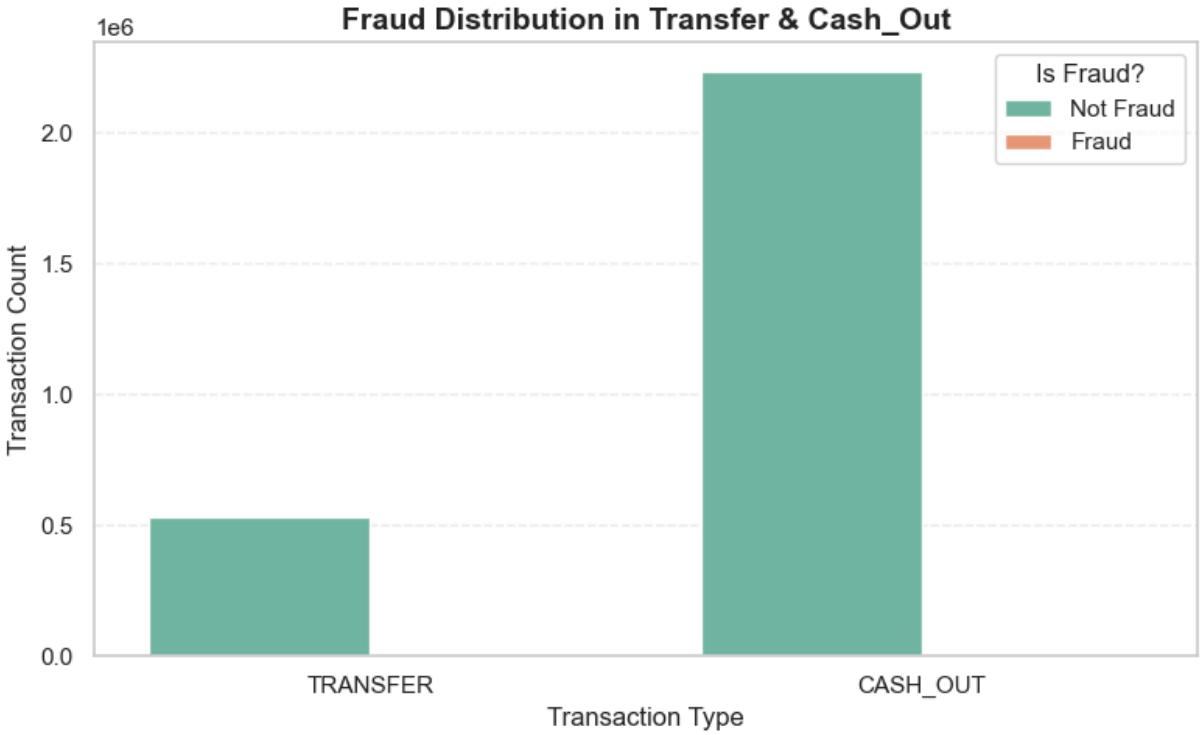
Focus on transfer to cash out

In [31]:
```
fraud_types = df[df["type"].isin(["TRANSFER","CASH_OUT"])]
```

In [32]:
```
fraud_types["type"].value_counts()
```

Out[32]:
```
type
CASH_OUT    2237500
TRANSFER     532909
Name: count, dtype: int64
```

In [33]:
```
plt.figure(figsize=(8, 5))
sns.countplot(data = fraud_types, x="type", hue= "isFraud", palette ="Set2")

plt.title("Fraud Distribution in Transfer & Cash_Out", fontsize= 14, fontweight ='b
plt.xlabel("Transaction Type", fontsize= 12)
plt.ylabel("Transaction Count", fontsize= 12)
plt.legend(title="Is Fraud?", labels=["Not Fraud", "Fraud"])
plt.grid(axis='y', linestyle='--', alpha=0.3)
plt.tight_layout()
plt.show()
```



In [34]:
```
corr = df[['amount', 'oldbalanceOrg', 'newbalanceOrig','oldbalanceDest', 'newbalanc
```

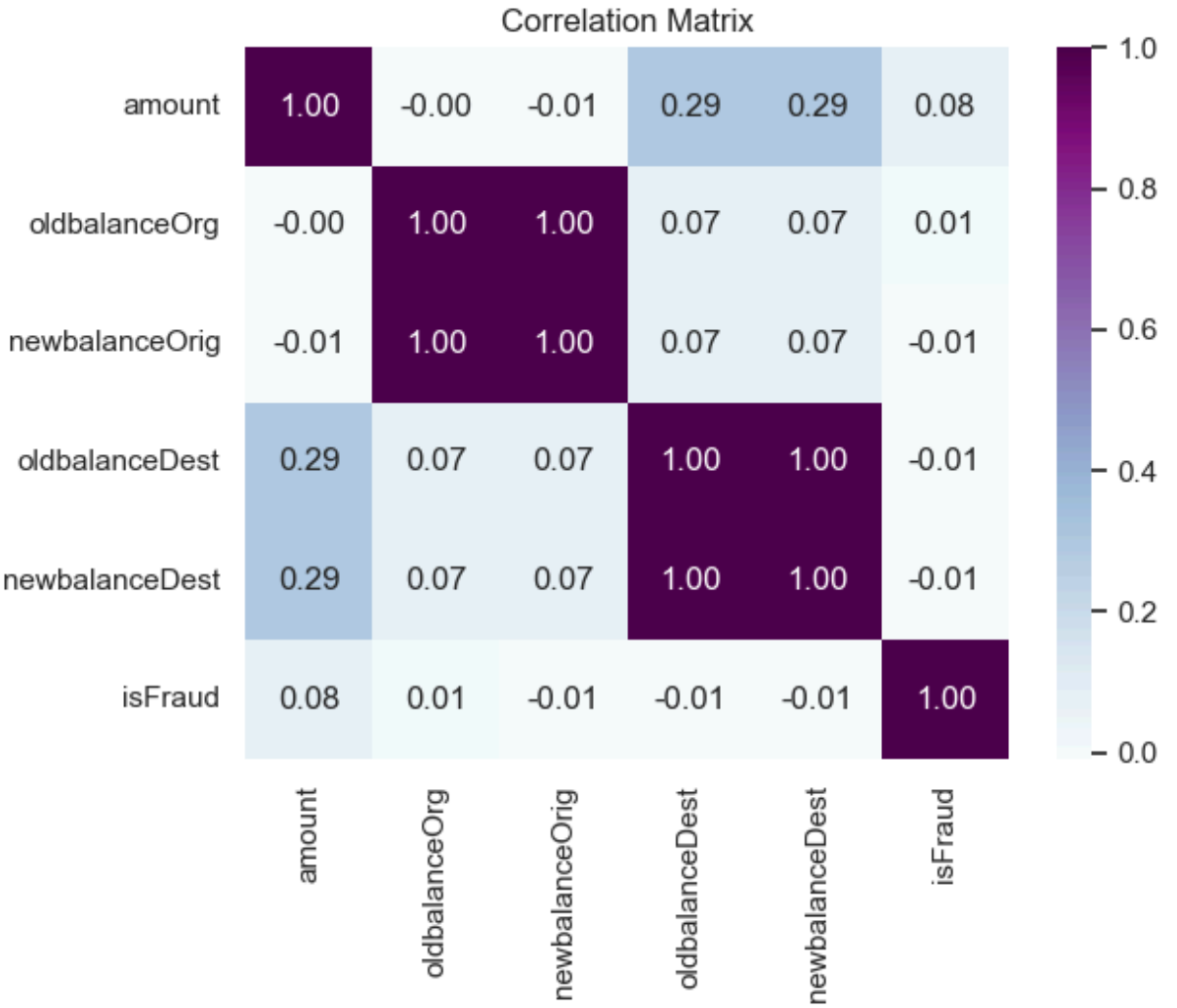
In [35]:
```
corr
```

Out[35]:

	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDe
amount	1.000000	-0.002762	-0.007861	0.294137	0.2941
oldbalanceOrg	-0.002762	1.000000	0.998803	0.066243	0.0662
newbalanceOrig	-0.007861	0.998803	1.000000	0.067812	0.0678
oldbalanceDest	0.294137	0.066243	0.067812	1.000000	1.0000
newbalanceDest	0.294137	0.066243	0.067812	1.000000	1.0000
isFraud	0.076688	0.010154	-0.008148	-0.005885	-0.0058

Correlation Heatmap

In [36]:
```
sns.heatmap(corr, annot = True, cmap = "BuPu", fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



Zero Balance After Transfer

In [37]:
```
zero_after_transfer = df[
    (df["oldbalanceOrg"]>0)&
    (df["newbalanceOrig"] == 0)&
    (df["type"].isin(["TRANSFER", "CASH_OUT"]))
]
```

In [38]:
```
len(zero_after_transfer)
```

Out[38]:
```
1188074
```

In [39]:
```
zero_after_transfer.head()
```

Out[39]:

	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	old
2	TRANSFER	181.00	C1305486145	181.0	0.0	C553264065	
3	CASH_OUT	181.00	C840083671	181.0	0.0	C38997010	
15	CASH_OUT	229133.94	C905080434	15325.0	0.0	C476402209	
19	TRANSFER	215310.30	C1670993182	705.0	0.0	C1100439041	
24	TRANSFER	311685.89	C1984094095	10835.0	0.0	C932583850	

In [40]:
```
df["isFraud"].value_counts()
```

Out[40]:
```
isFraud
0    6354407
1       8213
Name: count, dtype: int64
```

Machine Learning Modeling

In [41]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
```

In [42]:
```
df.head()
```

Out[42]:

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

In [43]:
```
df_model = df.drop(["nameOrig", "nameDest", "isFlaggedFraud"], axis =1)
```

In [44]:
```
df_model.head()
```

Out[44]:

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

In [45]:
```
categorical = ["type"]
numeric = ["amount","oldbalanceOrg", "newbalanceOrig", "oldbalanceDest","newbalance
```

Training & Testing

In [46]:
```
# Target and features
y = df_model["isFraud"]
X = df_model.drop("isFraud", axis = 1)
```

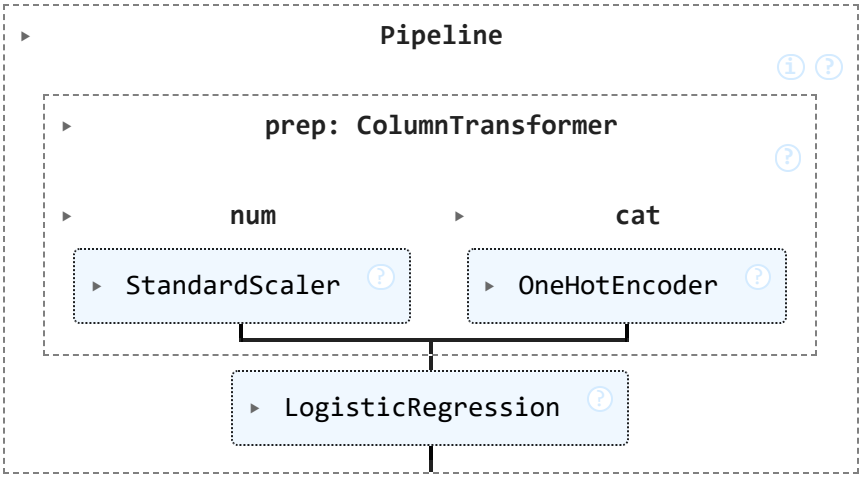
In [47]:
```
# Train-test split with stratification
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size = 0.3, stratify=
```

In [48]:
```
# Preprocessing pipeline
preprocessor = ColumnTransformer(
    transformers= [
        ("num", StandardScaler(), numeric),
        ("cat", OneHotEncoder(drop="first"),categorical)
    ],
    remainder = "drop"
)
```

In [49]:
```
# Preprocessing pipeline
pipeline = Pipeline([
    ("prep",preprocessor),
    ("clf", LogisticRegression(class_weight= "balanced", max_iter=1000))
])
```

In [50]:
```
# Train the model
pipeline.fit(X_train, y_train)
```

Out[50]:



In [51]:
```
# Predictions
y_pred = pipeline.predict(X_test)
```

In [52]:
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.94      0.97   1906322
           1       0.02      0.92      0.04      2464

    accuracy                           0.94   1908786
   macro avg       0.51      0.93      0.50   1908786
weighted avg       1.00      0.94      0.97   1908786
```

In [53]:
```
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
 [[1793611  112711]
 [    200    2264]]
```

In [54]:
```
pipeline.score(X_test, y_test)*100
```

Out[54]:
```
94.08466952293237
```

In [55]:
```
import joblib
```

In [56]:
```
joblib.dump(pipeline,"Fraud_detection_pipeline.pkl")
```

Out[56]:
```
['Fraud_detection_pipeline.pkl']
```

In [ ]: