```
warnings.filterwarnings("ignore")
```

In [25]:
```
df=pd.read_csv(r"PS_20174392719_1491204439457_log.csv")
```

🔍 Dataset Overview

A typical dataset for this project contains:

Transaction ID: Unique identifier for each transaction Timestamp: Date and time of the transaction Amount: Transaction value Payment Method: Credit card, debit card, PayPal, etc. User Information: Customer ID, location, device type Transaction Status: Fraudulent or Genuine

In [26]:
```
df.head()
```

Out[27]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	

In [28]:
```
df.info()
```

Out[30]:
```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
isFlaggedFraud    0
dtype: int64
```

In [31]:
```
df.nunique()
```

Out[32]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	ne
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	

In [33]:
```
df.duplicated().sum()
```

Out[33]:
```
0
```

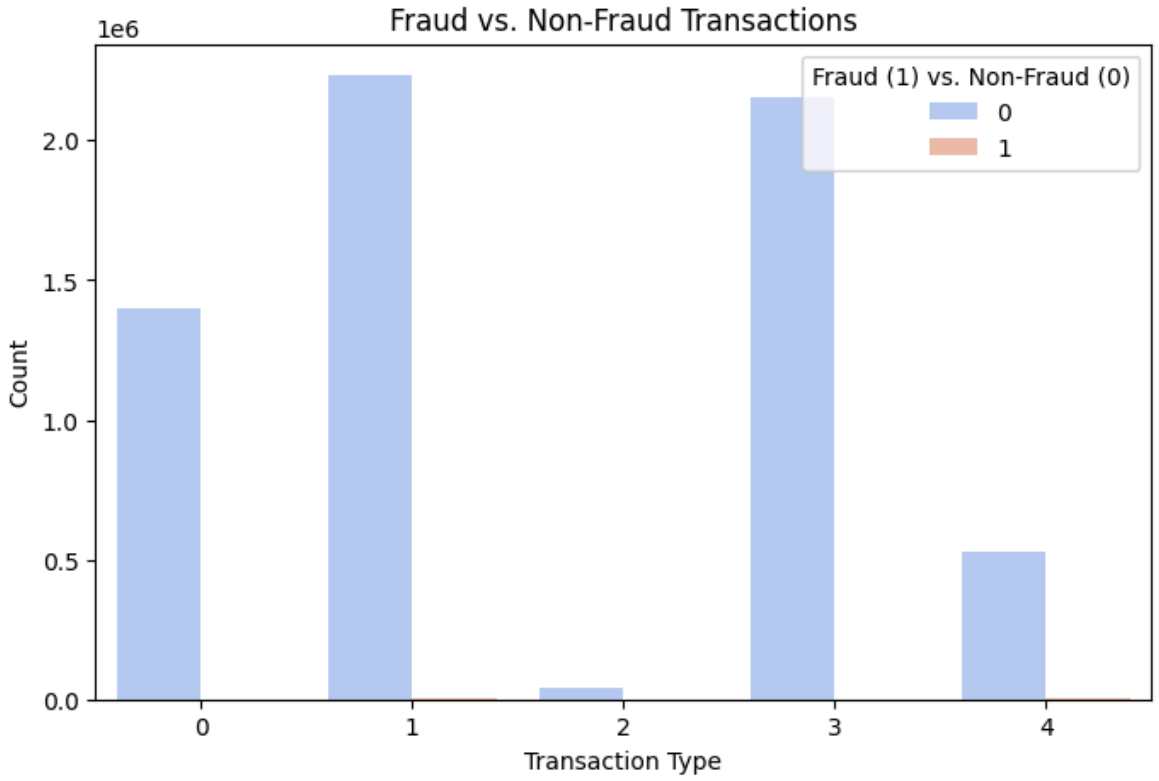
In [34]:
```
df['type'].count
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            int32
 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), int32(1), int64(3), object(2)
memory usage: 509.7+ MB
```

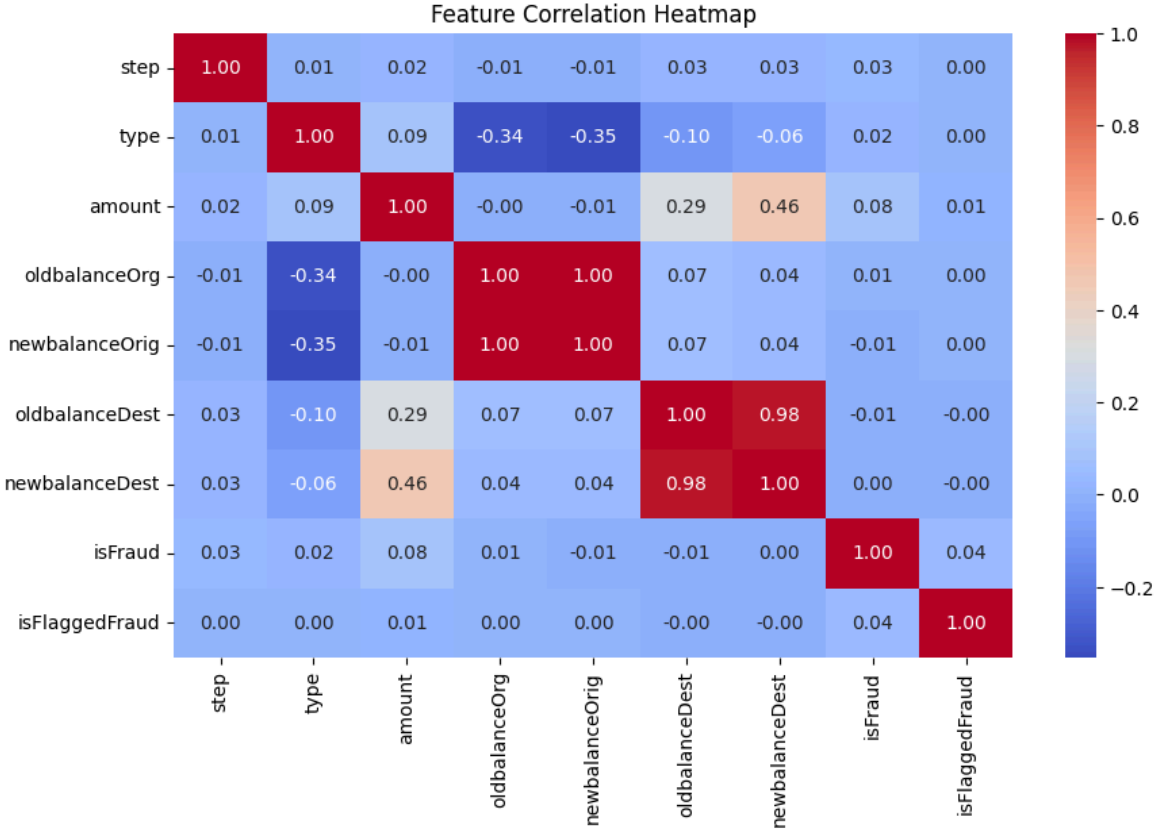
In [38]:
```
# Normalize numerical columns (for better model performance)
scaler = StandardScaler()
num_cols = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newb
df[num_cols] = scaler.fit_transform(df[num_cols])
```

In [39]:
```
# Save cleaned dataset for further analysis
#df.to_csv("cleaned_fraud_dataset.csv", index=False)
```

In [41]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```



In [42]:
```
# Select only numeric columns for correlation
numeric_df = df.select_dtypes(include=['number'])
```


Feature Correlation Heatmap
step 1.00 0.01 0.02 -0.01 -0.01 0.03 0.03 0.03 0.00
type 0.01 1.00 0.09 -0.34 -0.35 -0.10 -0.06 0.02 0.00
amount 0.02 0.09 1.00 -0.00 -0.01 0.29 0.46 0.08 0.01
oldbalanceOrg -0.01 -0.34 -0.00 1.00 1.00 0.07 0.04 0.01 0.00
newbalanceOrig -0.01 -0.35 -0.01 1.00 1.00 0.07 0.04 -0.01 0.00
oldbalanceDest 0.03 -0.10 0.29 0.07 0.07 1.00 0.98 -0.01 -0.00
newbalanceDest 0.03 -0.06 0.46 0.04 0.04 0.98 1.00 0.00 -0.00
isFraud 0.03 0.02 0.08 0.01 -0.01 -0.01 0.00 1.00 0.04
isFlaggedFraud 0.00 0.00 0.01 0.00 0.00 -0.00 -0.00 0.04 1.00
step
type
amount
oldbalanceOrg
newbalanceOrig
oldbalanceDest
newbalanceDest
isFraud
isFlaggedFraud
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

In [44]:
```
from sklearn.model_selection import train_test_split

# Define features and target variable
X = df.drop(columns=['isFraud'])
y = df['isFraud']

# Split dataset into train & test sets (80-20)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

Apply ML Algorithms for Fraud Detection

Logistic Regression

In [45]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Train Logistic Regression
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)

# Predict on test data
y_pred = lr_model.predict(X_test)
```

```
# Evaluate
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

XGBoost (Best for Large Datasets)

In [ ]:
```
import xgboost as xgb

# Train XGBoost model
xgb_model = xgb.XGBClassifier(n_estimators=100, learning_rate=0.1)
xgb_model.fit(X_train, y_train)

# Predict
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate
print("XGBoost Accuracy:", accuracy_score(y_test, y_pred_xgb))
print(classification_report(y_test, y_pred_xgb))
```

Fine-Tune Hyperparameters for Best Performance

In [ ]:
```
from sklearn.model_selection import GridSearchCV

# Define parameter grid for RandomForest
```

In [ ]: