1. Mount your Google Drive in Colab

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

2. Access the dataset from your Google Drive

```
import pandas as pd
file_path = '/content/drive/MyDrive/fraud_det/historical_dataset.csv'
df = pd.read_csv(file_path)
```

3. Basic exploration to understand the data

```
# Show basic info
print("🔍 Dataset Info:")
print(df.info())

# Check for missing values
print("\n🧼 Null Values:")
print(df.isnull().sum())

# Show class distribution
print("\n⚖️ Class Distribution (isFraud):")
print(df['isFraud'].value_counts(normalize=True))

# Preview the dataset
print("\n🔍 First 5 Rows:")
print(df.head())
```

```
🔍 Dataset 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
None

🧼 Null 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

⚖️ Class Distribution (isFraud):
isFraud
0    0.998709
1    0.001291
Name: proportion, dtype: float64

🔍 First 5 Rows:
   step      type    amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
0     1   PAYMENT   9839.64  C1231006815       170136.0       160296.36
1     1   PAYMENT   1864.28  C1666544295        21249.0        19384.72
2     1  TRANSFER    181.00  C1305486145          181.0            0.00
3     1  CASH_OUT    181.00   C840083671          181.0            0.00
```

```
4     1   PAYMENT  11668.14  C2048537720        41554.0        29885.86

      nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
0  M1979787155             0.0             0.0        0               0
1  M2044282225             0.0             0.0        0               0
2   C553264065             0.0             0.0        1               0
3    C38997010         21182.0             0.0        1               0
4  M1230701703             0.0             0.0        0               0
```

4. Handle missing values

```
df.loc[:, 'amount'] = df['amount'].fillna(df['amount'].mean())
print(df.isnull().sum())  # Should show 0 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
```

5. **Data preprocessing** :

- Drop unnecessary columns (nameOrig, nameDest) — these are IDs and not useful for prediction.
- Convert the type column (categorical) into numerical using one-hot encoding.
- Prepare the feature matrix X and the target y.

```
# Drop ID columns
df_clean = df.drop(['nameOrig', 'nameDest'], axis=1)

# One-hot encode the 'type' column
df_clean = pd.get_dummies(df_clean, columns=['type'])

# Separate features and target
X = df_clean.drop(['isFraud'], axis=1)
y = df_clean['isFraud']

print("✅ Preprocessing complete.")
print(f"Feature matrix shape: {X.shape}")
print(f"Target distribution:\n{y.value_counts(normalize=True)}")
```

```
✅ Preprocessing complete.
Feature matrix shape: (6362620, 12)
Target distribution:
isFraud
0    0.998709
1    0.001291
Name: proportion, dtype: float64
```

6. **Stratified sampling**

```
# 🧪 Stratified sampling: Keep all frauds, sample non-frauds
fraud_df = df_clean[df_clean['isFraud'] == 1]
non_fraud_df = df_clean[df_clean['isFraud'] == 0].sample(n=100000, random_state=42)  # Adjust n if needed

# 🔄 Combine and shuffle
sampled_df = pd.concat([fraud_df, non_fraud_df]).sample(frac=1, random_state=42).reset_index(drop=True)

# 🧠 Split into features and target
X = sampled_df.drop('isFraud', axis=1)
y = sampled_df['isFraud']
print(f"✅ Sampled dataset shape: {X.shape}")
print(f"🎯 Sampled class distribution:\n{y.value_counts(normalize=True)}")
```

```
✅ Sampled dataset shape: (108213, 12)
🎯 Sampled class distribution:
isFraud
0    0.924103
1    0.075897
Name: proportion, dtype: float64
```

7. **Train-test split**

```
from sklearn.model_selection import train_test_split

# 📊 Stratified split for fair class balance in both sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

print("✅ Train-test split done.")
print(f"Train shape: {X_train.shape}, Test shape: {X_test.shape}")
```

```
✅ Train-test split done.
Train shape: (86570, 12), Test shape: (21643, 12)
```

5. **Feature scaling**

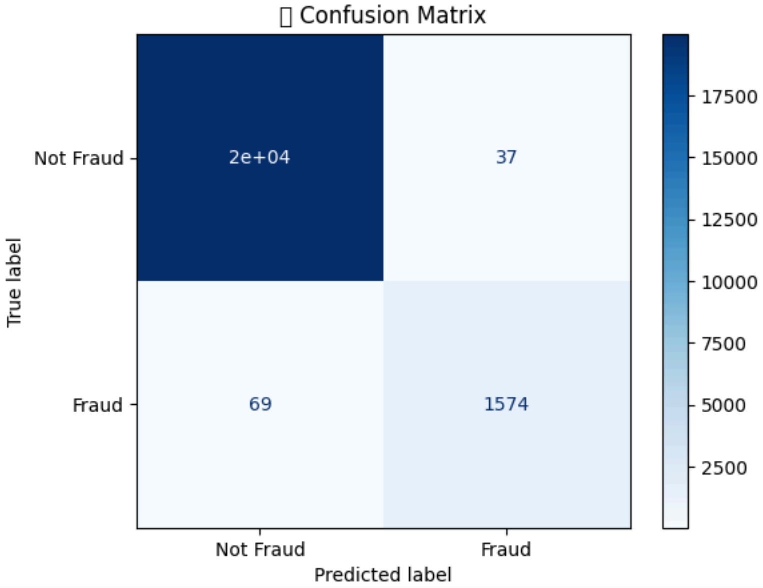
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("✅ Feature scaling complete.")
```

```
✅ Feature scaling complete.
```

6. **Model Training + Evaluation (Random Forest Classifier)**

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# 🎯 Train Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')
rf_model.fit(X_train_scaled, y_train)

# 🔍 Predict on test set
y_pred = rf_model.predict(X_test_scaled)
y_proba = rf_model.predict_proba(X_test_scaled)[:, 1]  # For ROC-AUC

# 📊 Evaluation
print("📈 Classification Report:")
print(classification_report(y_test, y_pred))

print("🎯 ROC AUC Score:", roc_auc_score(y_test, y_proba))

# 🧾 Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Not Fraud", "Fraud"])
disp.plot(cmap="Blues")
plt.title("🧾 Confusion Matrix")
plt.show()
```

```
📈 Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     20000
           1       0.98      0.96      0.97      1643

    accuracy                           1.00     21643
   macro avg       0.99      0.98      0.98     21643
weighted avg       1.00      1.00      1.00     21643

🎯 ROC AUC Score: 0.9990947352404138
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 129534 (\N{RECEIPT}) missing from font(s
  fig.canvas.print_figure(bytes_io, **kw)
```



7. Applying XGBoost

```
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score, ConfusionMatrixDisplay

# Calculate scale_pos_weight = (non-fraud cases / fraud cases) in train set
scale_pos_weight = (y_train == 0).sum() / (y_train == 1).sum()
print(f"📊 scale_pos_weight: {scale_pos_weight:.2f}")

# Initialize and train the model
xgb_model = XGBClassifier(
    n_estimators=100,
    max_depth=5,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8,
    scale_pos_weight=scale_pos_weight,
    use_label_encoder=False,
    eval_metric='logloss',
    tree_method='hist'  # Fastest for CPU or 'gpu_hist' if using GPU
)

xgb_model.fit(X_train_scaled, y_train)
print("✅ XGBoost model trained.")
```

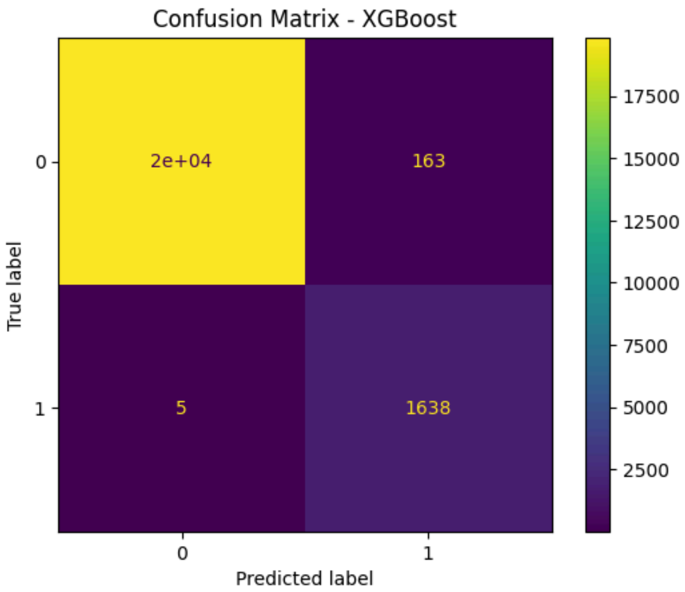
```
📊 scale_pos_weight: 12.18
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [17:11:16] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
✅ XGBoost model trained.
```

8. Evaluate the Tuned Model

```
# Predict and evaluate
y_pred_xgb = xgb_model.predict(X_test_scaled)
y_proba_xgb = xgb_model.predict_proba(X_test_scaled)[:, 1]

print("📋 Classification Report (XGBoost):")
print(classification_report(y_test, y_pred_xgb))

ConfusionMatrixDisplay.from_predictions(y_test, y_pred_xgb)
```

```
plt.title("Confusion Matrix - XGBoost")
plt.show()

roc_auc_xgb = roc_auc_score(y_test, y_proba_xgb)
print(f"🧠 ROC-AUC Score (XGBoost): {roc_auc_xgb:.4f}")
```

```
📋 Classification Report (XGBoost):
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     20000
           1       0.91      1.00      0.95      1643

    accuracy                           0.99     21643
   macro avg       0.95      0.99      0.97     21643
weighted avg       0.99      0.99      0.99     21643
```



```
🧠 ROC-AUC Score (XGBoost): 0.9996
```

9. Save the model

```
import joblib
from sklearn.preprocessing import StandardScaler

# Assuming you have already trained your XGBoost model
# For example: xgb_model.fit(X_train, y_train)

# If you are using a StandardScaler for scaling the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)  # Scale the training data

# Fit your model on the scaled data (replace with your actual model training step)
xgb_model.fit(X_train_scaled, y_train)

# Save the trained model to a .pkl file
model_path = "/content/drive/MyDrive/fraud_det/xgb_model.pkl"
joblib.dump(xgb_model, model_path)
print(f"✅ Model saved to: {model_path}")

# Save the scaler to a .pkl file
scaler_path = "/content/drive/MyDrive/fraud_det/scaler.pkl"
joblib.dump(scaler, scaler_path)
print(f"✅ Scaler saved to: {scaler_path}")
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning: [17:27:19] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
✅ Model saved to: /content/drive/MyDrive/fraud_det/xgb_model.pkl
✅ Scaler saved to: /content/drive/MyDrive/fraud_det/scaler.pkl
```

```
# Load the saved XGBoost model and scaler
xgb_model = joblib.load('/content/drive/MyDrive/fraud_det/xgb_model.pkl')
scaler = joblib.load('/content/drive/MyDrive/fraud_det/scaler.pkl')

# Now you can use the model and scaler to make predictions, as shown earlier
```

10. Running the model on a sample transaction

```
# Load model and scaler
import joblib
xgb_model = joblib.load("/content/drive/MyDrive/fraud_det/xgb_model.pkl")
scaler = joblib.load("/content/drive/MyDrive/fraud_det/scaler.pkl")

# Simulated data with correct order and columns
simulated_data = pd.DataFrame({
    'step': [50, 100],
    'amount': [250000, 1000000],  # Very large amounts
    'oldbalanceOrg': [250000, 1000000],  # Original sender balance
    'newbalanceOrig': [0, 0],  # Balance drops to zero
    'oldbalanceDest': [0, 10],  # Receiver had almost no balance
    'newbalanceDest': [250000, 1000010],  # Receiver suddenly gains a large amount
    'isFlaggedFraud': [1, 1],  # System thinks it's suspicious
    'type_CASH_IN': [0, 0],
    'type_CASH_OUT': [0, 1],  # One is CASH_OUT
    'type_DEBIT': [0, 0],
    'type_PAYMENT': [0, 0],
    'type_TRANSFER': [1, 0]  # Other is TRANSFER
})

# Scale simulated data
simulated_data_scaled = scaler.transform(simulated_data)

# Predict probabilities
probs = xgb_model.predict_proba(simulated_data_scaled)

# Apply custom threshold
threshold = 0.15
predictions = (probs[:, 1] >= threshold).astype(int)

# Display results
print("📊 Probabilities of fraud:", probs[:, 1])
print("🚨 Final Predictions with threshold:", predictions)
```

```
📊 Probabilities of fraud: [0.96058345 0.99930894]
🚨 Final Predictions with threshold: [1 1]
```

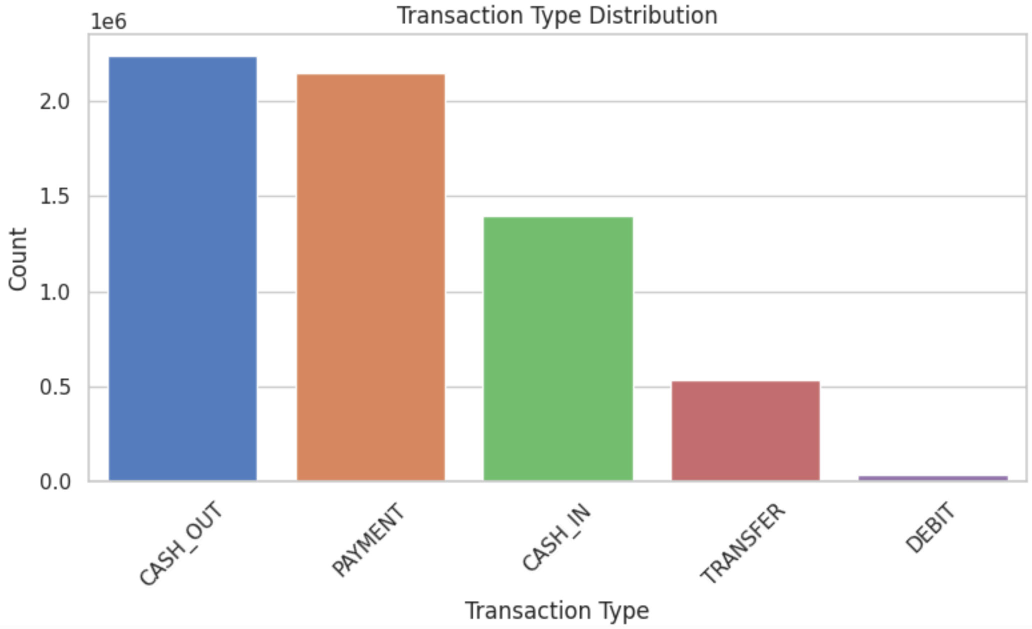
11. Visualization

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
sns.set(style="whitegrid")
%matplotlib inline
```

Transaction type distribution

```
plt.figure(figsize=(8, 5))
sns.countplot(data=df, x='type', order=df['type'].value_counts().index, palette='muted')
plt.title('Transaction Type Distribution')
plt.xlabel('Transaction Type')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

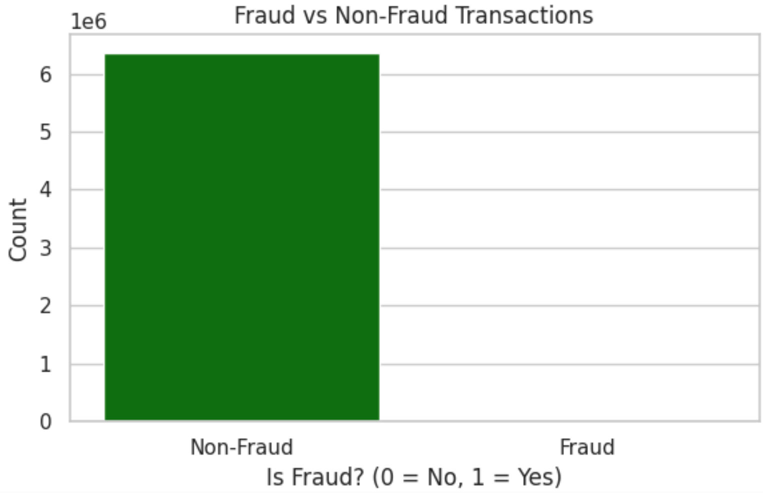
```
<ipython-input-36-c147d27fdda8>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.countplot(data=df, x='type', order=df['type'].value_counts().index, palette='muted')
```



Fraud v/s Non-fraud count

```
plt.figure(figsize=(6, 4))
sns.countplot(data=df, x='isFraud', palette=['green', 'red'])
plt.title('Fraud vs Non-Fraud Transactions')
plt.xlabel('Is Fraud? (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.xticks([0, 1], ['Non-Fraud', 'Fraud'])
plt.tight_layout()
plt.show()
```

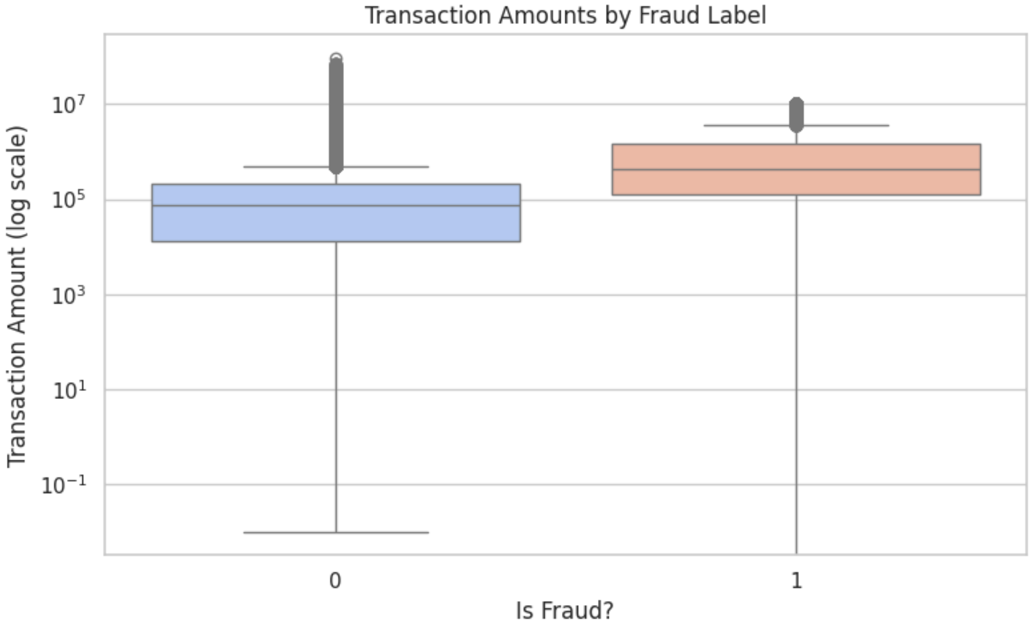
```
<ipython-input-37-4877512cbb36>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.countplot(data=df, x='isFraud', palette=['green', 'red'])
```



Boxplot of Amounts by Fraud Label

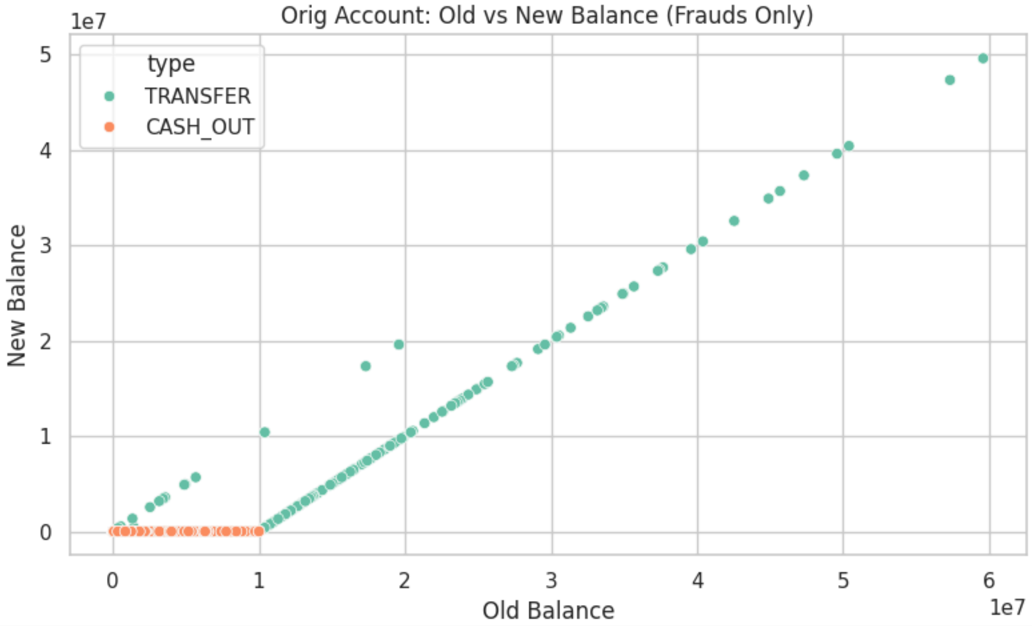
```
plt.figure(figsize=(8, 5))
sns.boxplot(data=df, x='isFraud', y='amount', palette='coolwarm')
plt.yscale('log')  # Amounts can be very large
plt.title('Transaction Amounts by Fraud Label')
plt.xlabel('Is Fraud?')
plt.ylabel('Transaction Amount (log scale)')
plt.tight_layout()
plt.show()
```

```
<ipython-input-38-148209421c8d>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le

  sns.boxplot(data=df, x='isFraud', y='amount', palette='coolwarm')
```



Old vs New Balance (Orig Account) – Only for Frauds

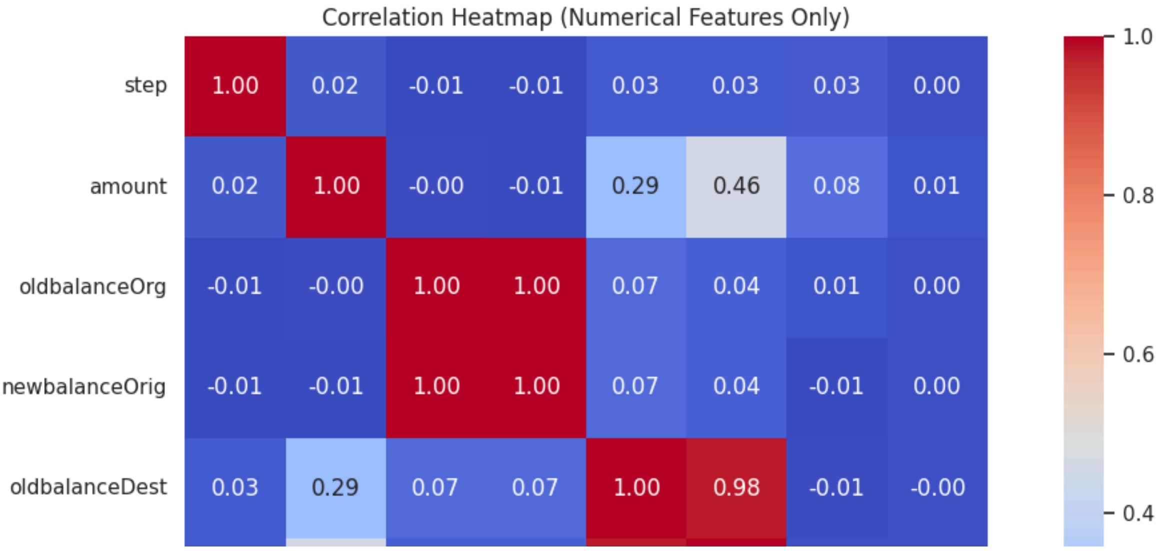
```
fraud_df = df[df['isFraud'] == 1]

plt.figure(figsize=(8, 5))
sns.scatterplot(data=fraud_df, x='oldbalanceOrg', y='newbalanceOrig', hue='type', palette='Set2')
plt.title('Orig Account: Old vs New Balance (Frauds Only)')
plt.xlabel('Old Balance')
plt.ylabel('New Balance')
plt.tight_layout()
plt.show()
```



Correlation Heatmap

```
# Drop non-numeric columns before correlation
numeric_df = df.select_dtypes(include='number')

# Plot the correlation heatmap
```

```
plt.figure(figsize=(12, 8))
corr = numeric_df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title('Correlation Heatmap (Numerical Features Only)')
plt.tight_layout()
plt.show()
```



Correlation Heatmap (Numerical Features Only)