

## AI4D LAB TRAINING

---

[Zephania Reuben \(https://nsoma.me\)](https://nsoma.me)

July 18, 2023

---

## FRAUD DETECTION

---

# ANALYSIS AND VISUALIZATION

### Data Sources

Due to the private nature of financial data, there is a lack of publicly available datasets that can be used for analysis. In this project, a synthetic dataset, publicly available on Kaggle, generated using a simulator called PaySim is used. The dataset was generated using aggregated metrics from the private dataset of a multinational mobile financial services company, and then malicious entries were injected. The dataset contains 10 columns of information for ~6 million rows of data. The key columns available are –

- Type of transactions
- Amount transacted
- Customer ID and Recipient ID
- Old and New balance of Customer and Recipient
- Time step of the transaction
- Whether the transaction was fraudulent or not

## Data Description

The data used for this analysis is a synthetically generated digital transactions dataset using a simulator called PaySim. PaySim simulates mobile money transactions based on a sample of real transactions extracted from one month of financial logs from a mobile money service implemented in an African country. It aggregates anonymized data from the private dataset to generate a synthetic dataset and then injects fraudulent transactions. The dataset has over 6 million transactions and 10 variables. There is a variable named 'isFraud' that indicates actual fraud status of the transaction. This is the class variable for our analysis. The columns in the dataset are described as follows:

Name of the variable	Description
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
oldbalanceOrig	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.

```
In [2]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
sb.set_style('whitegrid')
```

Let's start by importing the data and examining it.

```
In [3]: df = pd.read_csv("../data/transactions_train.csv")
```

In [4]: `df.head()`

Out[4]:

	step	type	amount	nameOrig	oldbalanceOrig	newbalanceOrig	nameDest
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M197978715!
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M204428222!
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C55326406!
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C3899701!
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M123070170!

In [5]: `df.columns`

Out[5]: Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrig', 'newbalanceOrig', 'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud'], dtype='object')

Few column names are wrong, let's fix those by renaming.

In [6]: `df = df.rename(columns={'oldbalanceOrig':'oldBalanceOrig', 'newbalanceOrig':'newBalanceOrig', 'oldbalanceDest': 'oldBalanceDest', 'newbalanceDest': 'newBalanceDest'})`  
`df.head()`

Out[6]:

	step	type	amount	nameOrig	oldBalanceOrig	newBalanceOrig	nameDest
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M197978715
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M204428222
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C55326406
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C3899701
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M123070170

Let's also check for duplicate data

In [7]: `df.duplicated().sum()`

Out[7]: 0

In [8]: `len(df)-len(df.drop_duplicates())`

Out[8]: 0

Test if there any missing values in DataFrame. It turns out there are no obvious missing values but, as we will see below, this does not rule out proxies by a numerical value like 0.

```
In [9]: df.isnull().values.any()
```

```
Out[9]: False
```

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

```
Out[10]: step          0
         type          0
         amount        0
         nameOrig       0
         oldBalanceOrig 0
         newBalanceOrig 0
         nameDest       0
         oldBalanceDest 0
         newBalanceDest 0
         isFraud        0
         dtype: int64
```

Since it is necessary that all columns in the data are of appropriate type for analysis, let's check there is any need for type conversion. Following are the initial types of the columns read by python.

```
In [11]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6351193 entries, 0 to 6351192
Data columns (total 10 columns):
 #   Column          Dtype
---  -
 0   step            int64
 1   type            object
 2   amount          float64
 3   nameOrig        object
 4   oldBalanceOrig  float64
 5   newBalanceOrig  float64
 6   nameDest        object
 7   oldBalanceDest  float64
 8   newBalanceDest  float64
 9   isFraud         int64
dtypes: float64(5), int64(2), object(3)
memory usage: 484.6+ MB
```

```
In [12]: nan_sum = df['isFraud'].isna().sum()
         print(nan_sum)
```

```
0
```

Before proceeding with the analysis, let's present the summary statistics of the variables. In case of numeric variables, let's evaluate the mean, standard deviation and the range of values at different percentiles.

```
In [13]: #Summary of Statistics of Numeric Variables
df.describe()
```

Out[13]:

	step	amount	oldBalanceOrig	newBalanceOrig	oldBalanceDest	newl
<b>count</b>	6.351193e+06	6.351193e+06	6.351193e+06	6.351193e+06	6.351193e+06	6
<b>mean</b>	2.425553e+02	1.798155e+05	8.347957e+05	8.561696e+05	1.101043e+06	1
<b>std</b>	1.410676e+02	6.036310e+05	2.889959e+06	2.926073e+06	3.398924e+06	3
<b>min</b>	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0
<b>25%</b>	1.550000e+02	1.338829e+04	0.000000e+00	0.000000e+00	0.000000e+00	0
<b>50%</b>	2.380000e+02	7.486483e+04	1.415300e+04	0.000000e+00	1.330865e+05	2
<b>75%</b>	3.340000e+02	2.087152e+05	1.073460e+05	1.443651e+05	9.438661e+05	1
<b>max</b>	6.990000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3

In case of categorivcal variables, let's evaluate only the number of unique categories, the most frequency category and it's frequency.

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

```
In [14]: # Summary of Statistics of Categorical Variables
df.describe(include=['object'])
```

Out[14]:

	type	nameOrig	nameDest
<b>count</b>	6351193	6351193	6351193
<b>unique</b>	5	6341907	2716810
<b>top</b>	CASH_OUT	C1832548028	C1286084959
<b>freq</b>	2233369	3	113

### Exploratory Data Analysis (EDA)

In this section, we will do EDA to understand the data more.

## Class Imbalance

In this exploratory analysis, we assess the class imbalance in the dataset. The class imbalance is defined as a percentage of the total number of transactions presented in the isFraud column.

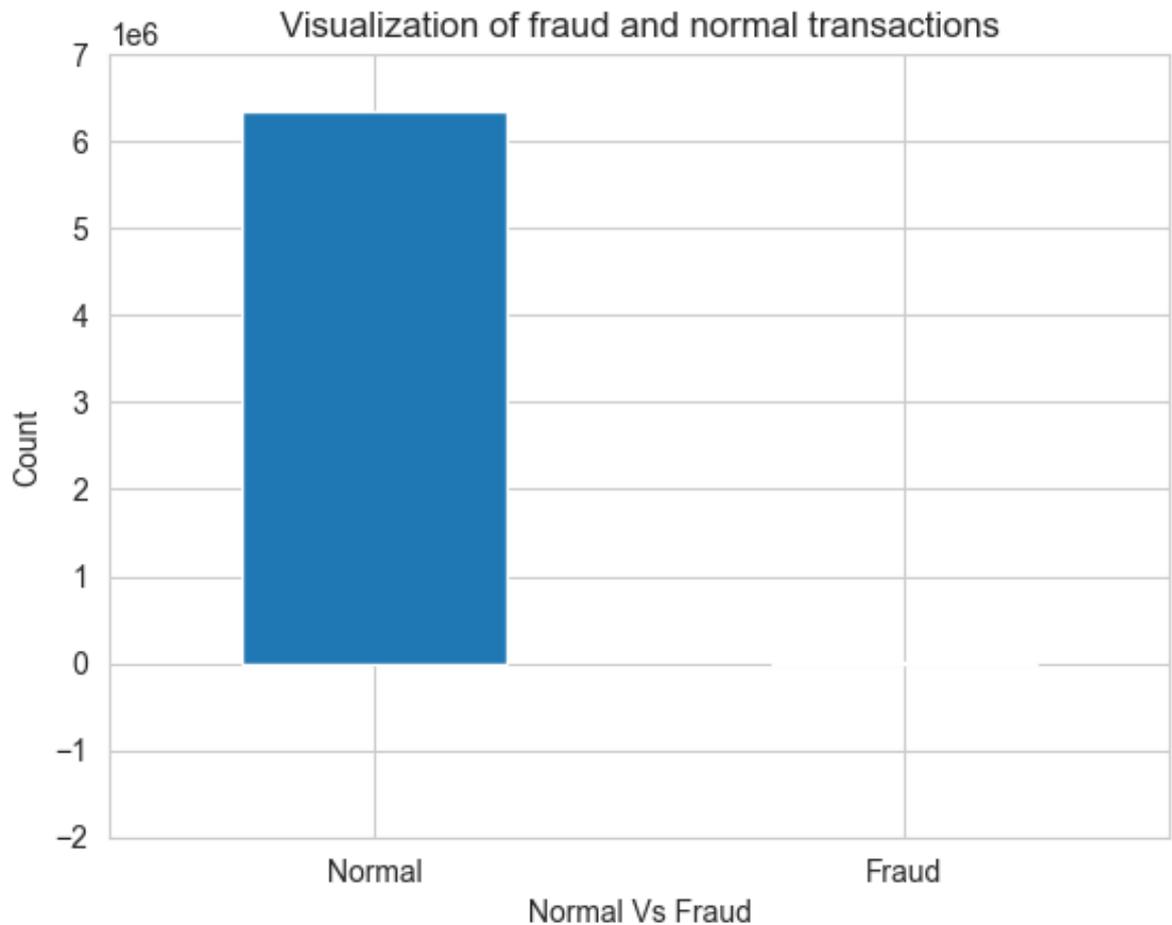
```
In [16]: Total_transactions = len(df)
normal = len(df[df.isFraud == 0])
fraudulent = len(df[df['isFraud'] == 1])
fraud_percentage = round(fraudulent/Total_transactions*100, 2)
normal_percentage = round(normal/Total_transactions*100, 2)
print('Total number of Transactions are {}'.format(Total_transactions))
print('Number of Normal Transactions are {}'.format(normal))
print('Number of fraudulent transactions are {}'.format(fraudulent))
print('Percentage of Fraud Transactions is {}'.format(fraud_percentage))
print('Percentage of Normal Transactions is {}'.format(normal_percentage))
```

```
Total number of Transactions are 6351193
Number of Normal Transactions are 6343476
Number of fraudulent transactions are 7717
Percentage of Fraud Transactions is 0.12
Percentage of Normal Transactions is 99.88
```

```
In [17]: normal_percentage = round(normal/Total_transactions*100, 2)
print('Percentage of Normal Transactions is {}'.format(normal_percentage))
```

```
Percentage of Normal Transactions is 99.88
```

```
In [18]: # Visualize
labels = ["Normal", "Fraud"]
count_classes = df.value_counts(df['isFraud'], sort=True)
count_classes.plot(kind = "bar", rot = 0)
plt.title("Visualization of fraud and normal transactions")
plt.ylabel("Count")
plt.xlabel("Normal Vs Fraud")
plt.ylim([-2e6, 7e6])
plt.xticks(range(2), labels)
plt.figure(figsize=(100,100))
plt.show()
```



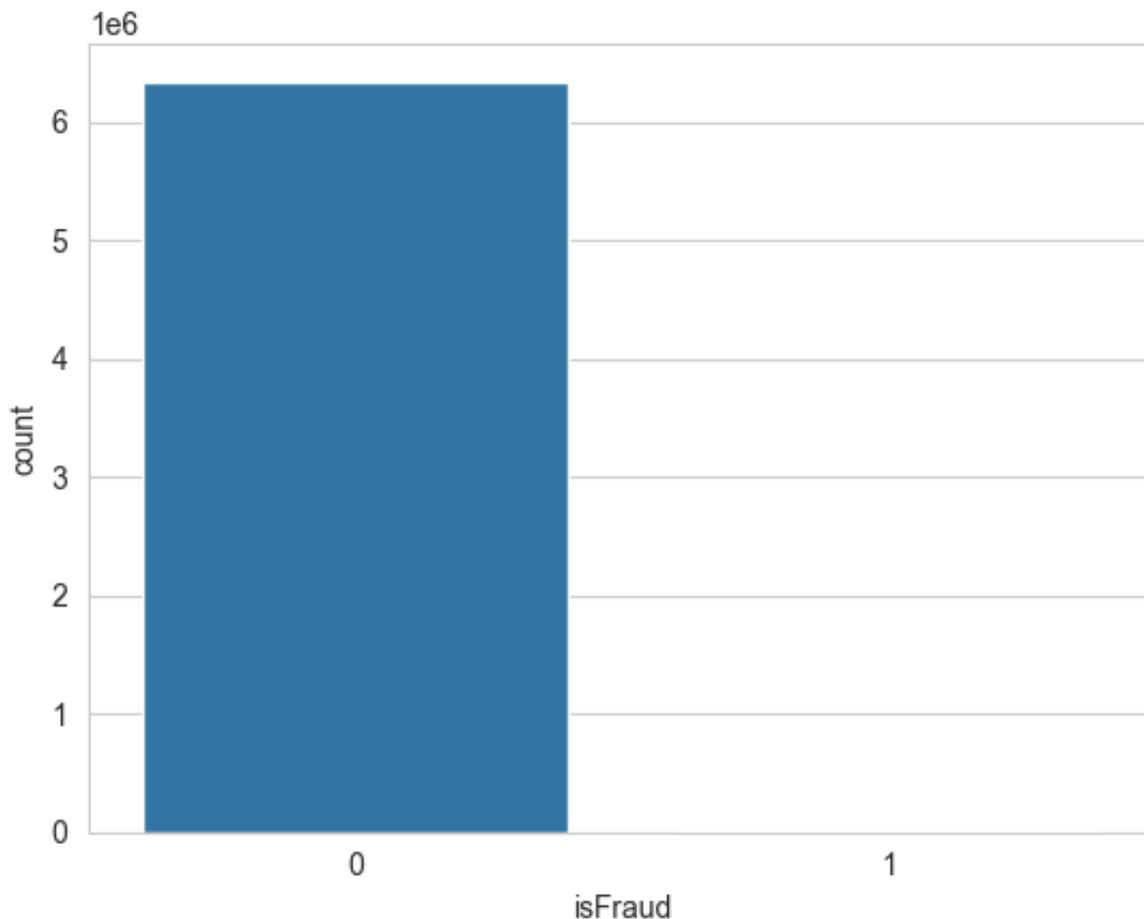
<Figure size 10000x10000 with 0 Axes>

```
In [19]: df['isFraud'].value_counts()
```

```
Out[19]: isFraud
0      6343476
1         7717
Name: count, dtype: int64
```

```
In [21]: sb.countplot(df, x='isFraud')
```

```
Out[21]: <Axes: xlabel='isFraud', ylabel='count'>
```



As we can see from the figure above, there is an enormous difference between the transactions.

Only 0.13% (8,213) transactions in the dataset are fraudulent indicating high class imbalance in the dataset. This is important because if we build a machine learning model on this highly skewed data, the non-fraudulent transactions will influence the training of the model almost entirely, thus affecting the results.

### Types of Transactions

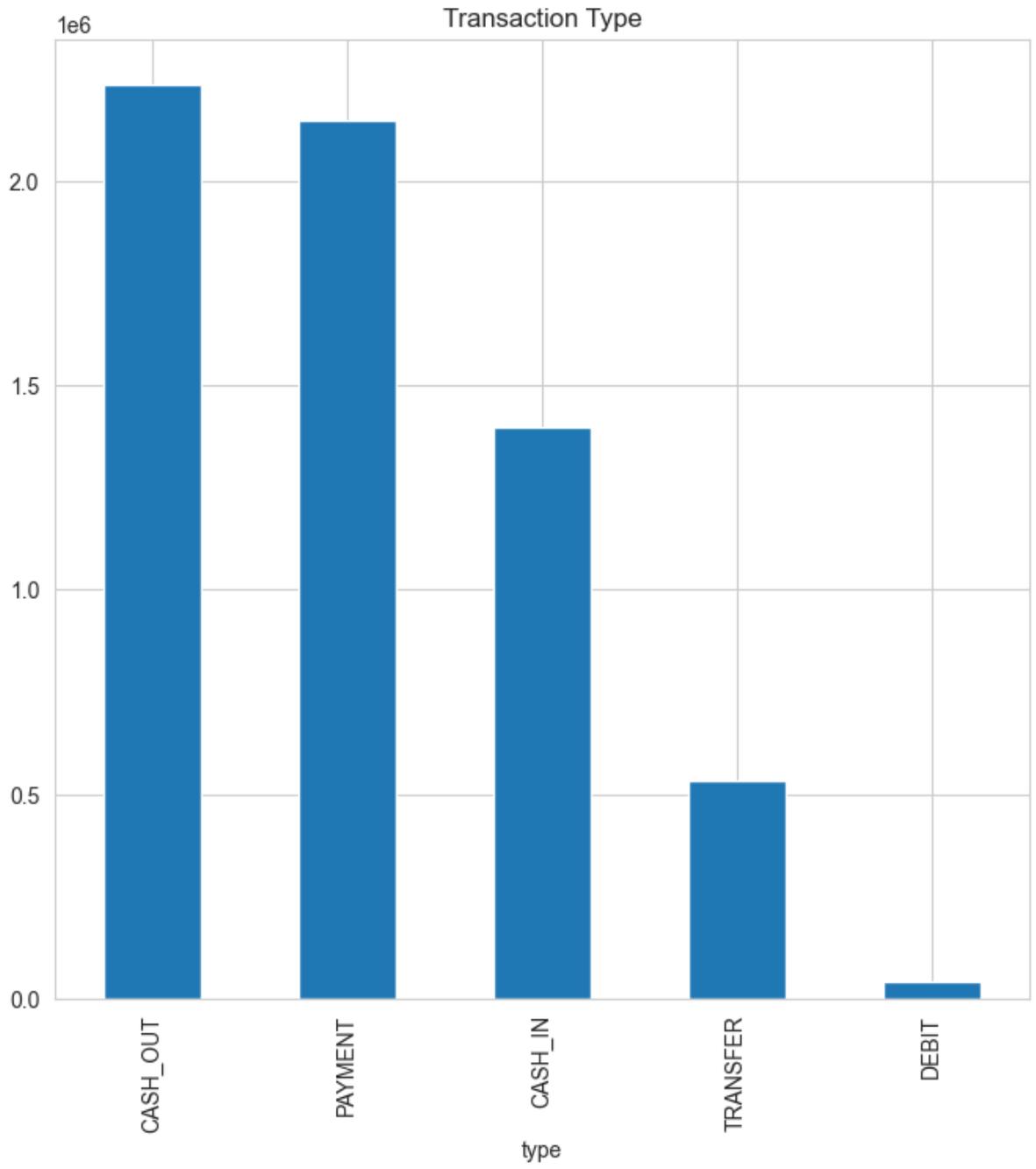
In this section, let's explore the dataset by examining the type variable. We present what the different types of transactions are and which of these types can be fraudulent.

The following plot shows the frequencies of the different transaction types:

```
In [22]: print(df.type.value_counts(ascending=True))
```

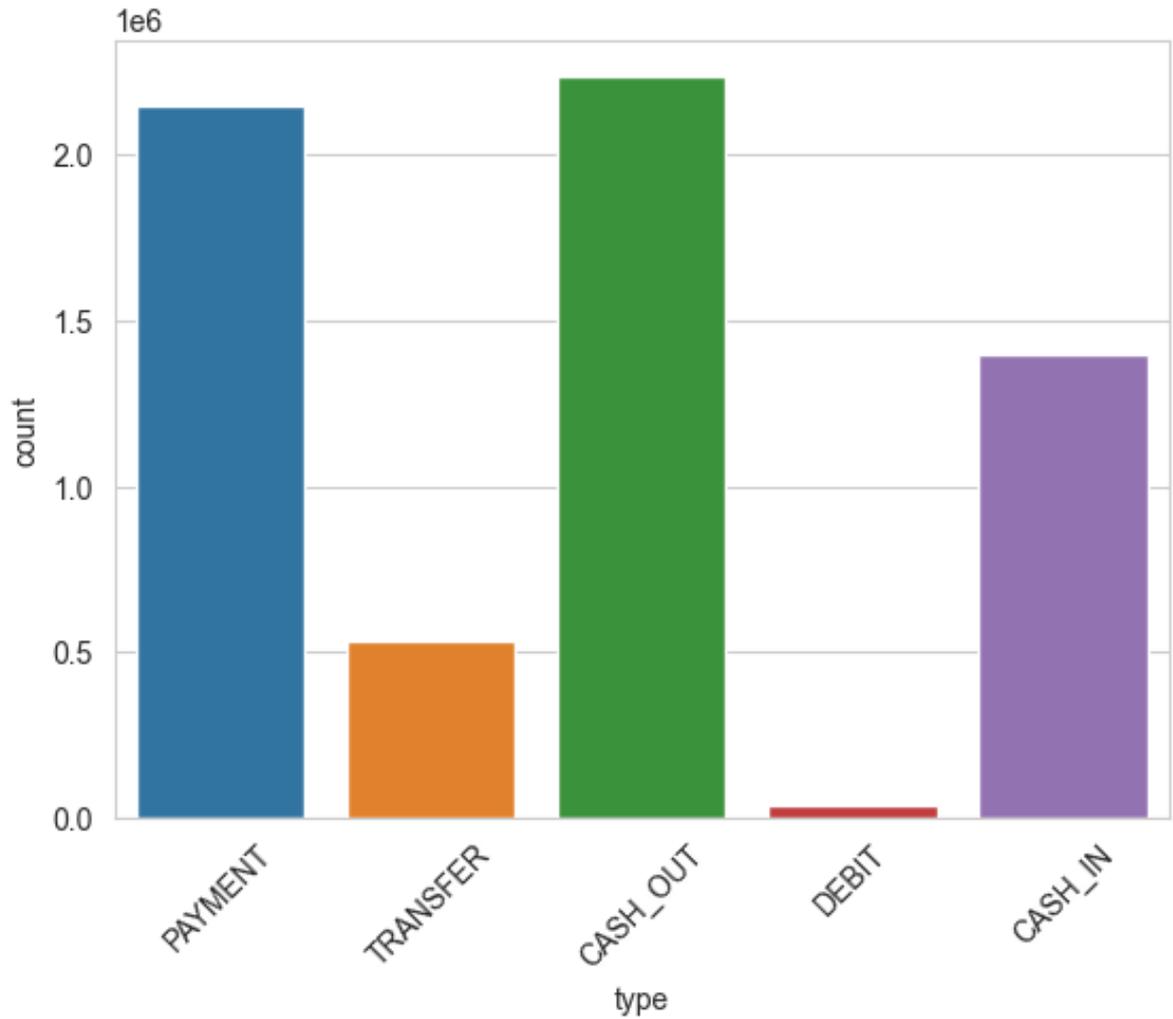
```
type
DEBIT          41310
TRANSFER      531817
CASH_IN       1396865
PAYMENT       2147832
CASH_OUT      2233369
Name: count, dtype: int64
```

```
In [23]: # Visualize the above data
f, ax = plt.subplots(1, 1, figsize=(8,8))
df.type.value_counts().plot(kind='bar', title="Transaction Type", a
x=ax, figsize=(8,8))
plt.show()
```

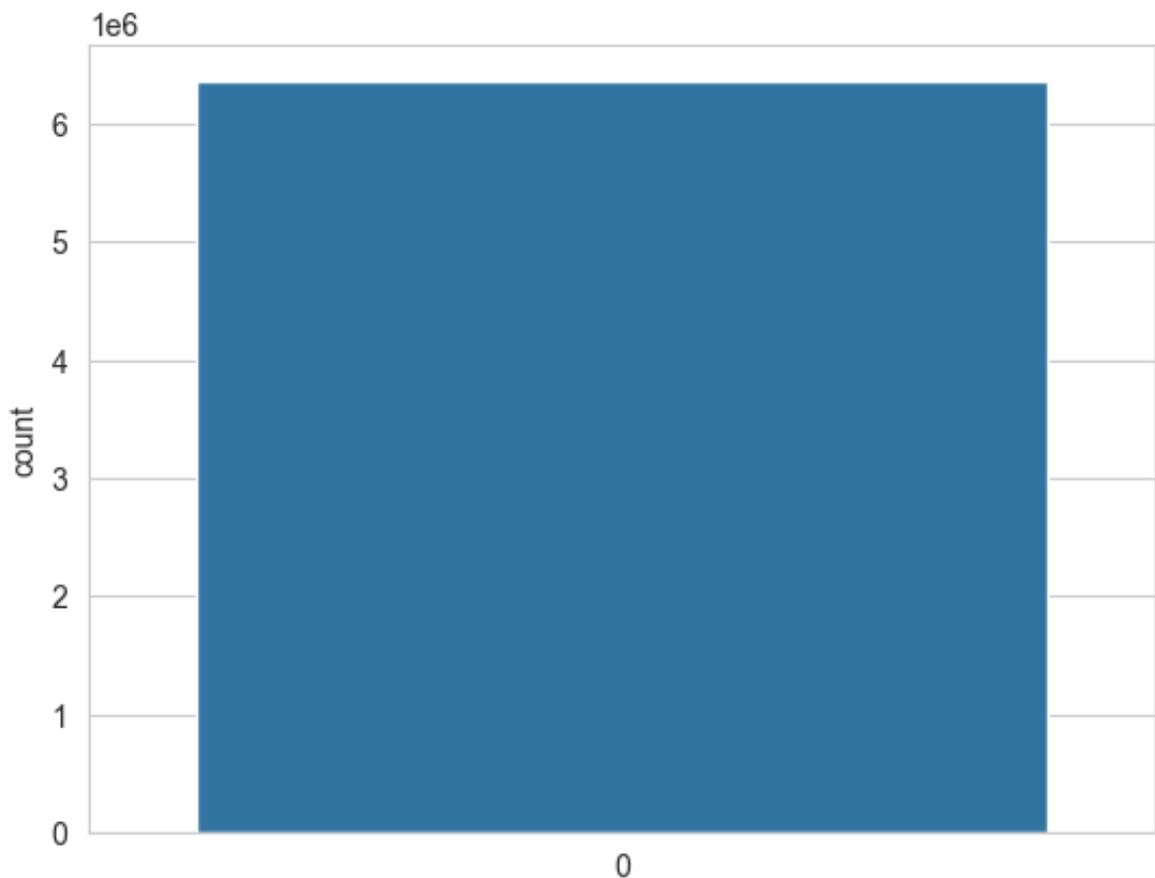


```
In [24]: g= sb.countplot(x='type', data=df)
plt.xticks(rotation=45)
```

```
Out[24]: (array([0, 1, 2, 3, 4]),
 [Text(0, 0, 'PAYMENT'),
  Text(1, 0, 'TRANSFER'),
  Text(2, 0, 'CASH_OUT'),
  Text(3, 0, 'DEBIT'),
  Text(4, 0, 'CASH_IN')])
```



```
In [25]: sb.countplot(data=df['isFraud'])  
plt.show()
```

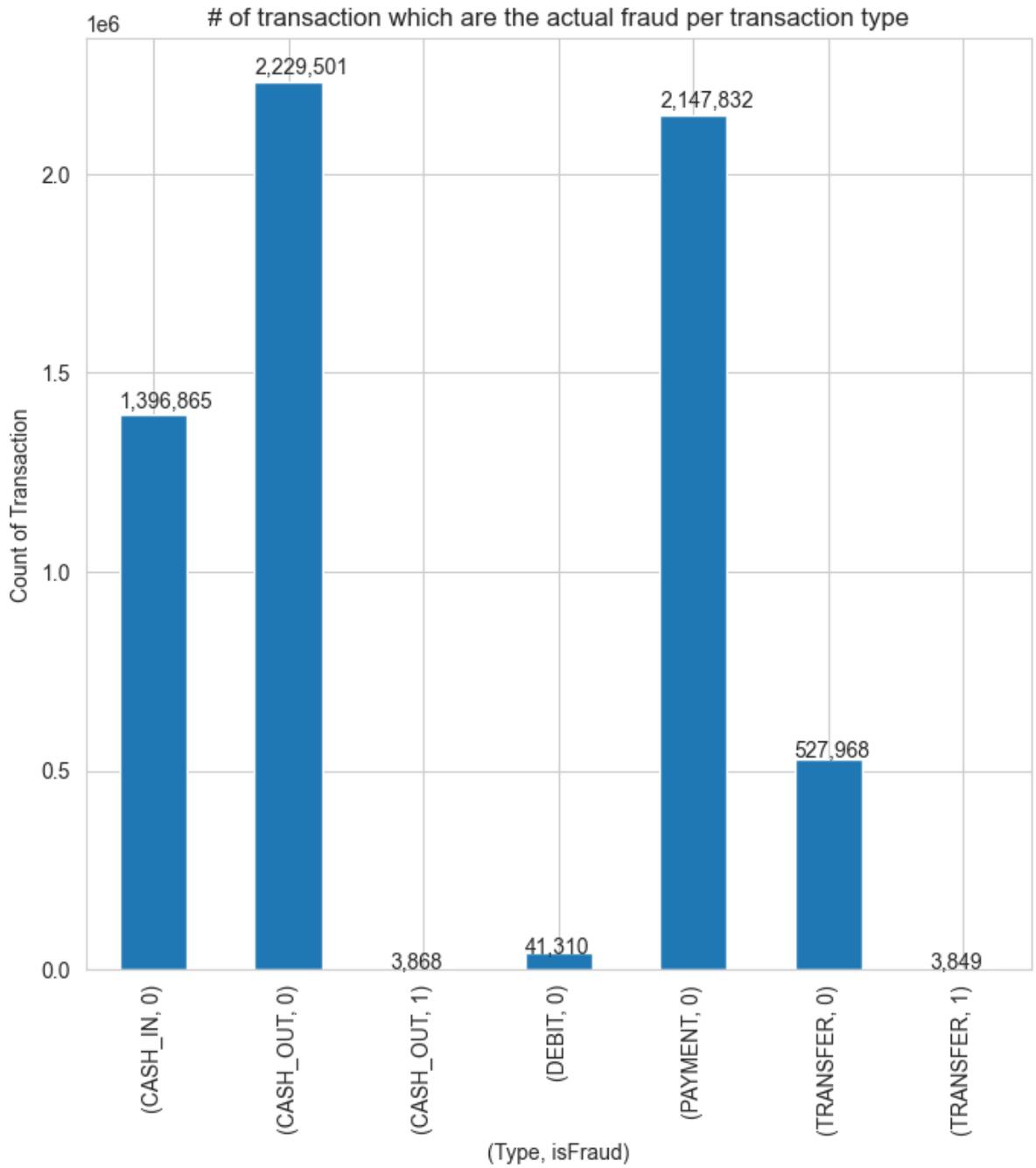


The most frequent transaction types are CASH-OUT and PAYMENT.

There are 1 flags which stand out to me and it's interesting to look into: isFraud column. From the data Dictionary, isFraud is the indicator which indicates the actual fraud transactions.

Let's quickly check what kinds of transaction are being flagged and are fraud.

```
In [26]: ax = df.groupby(['type', 'isFraud']).size().plot(kind='bar', figsize=(8,8))  
ax.set_title("# of transaction which are the actual fraud per transaction type")  
ax.set_xlabel("(Type, isFraud)")  
ax.set_ylabel("Count of Transaction")  
for p in ax.patches:  
    ax.annotate(str(format(int(p.get_height()), ',d')), (p.get_x(), p.get_height()*1.01))
```



```
In [27]: fraud_df = df[(df["isFraud"] == 1)]
fraud_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7717 entries, 2 to 6351191
Data columns (total 10 columns):
#   Column                Non-Null Count  Dtype
---  -
0   step                  7717 non-null   int64
1   type                  7717 non-null   object
2   amount                7717 non-null   float64
3   nameOrig              7717 non-null   object
4   oldBalanceOrig        7717 non-null   float64
5   newBalanceOrig        7717 non-null   float64
6   nameDest              7717 non-null   object
7   oldBalanceDest        7717 non-null   float64
8   newBalanceDest        7717 non-null   float64
9   isFraud               7717 non-null   int64
dtypes: float64(5), int64(2), object(3)
memory usage: 663.2+ KB
```

```
In [28]: non_fraud = len(fraud_df[fraud_df.isFraud == 0])
fraud = len(fraud_df[fraud_df.isFraud == 1])
print(non_fraud)
print(fraud)
```

```
0
7717
```

```
In [29]: fraud_df.describe(include=['object'])
```

Out[29]:

	type	nameOrig	nameDest
count	7717	7717	7717
unique	2	7717	7677
top	CASH_OUT	C1305486145	C385133759
freq	3868	1	2

```
In [30]: fraud_df['type'].unique()
```

Out[30]: array(['TRANSFER', 'CASH\_OUT'], dtype=object)

```
In [32]: fraud_df['type'].value_counts()
```

```
Out[32]: type
CASH_OUT      3868
TRANSFER      3849
Name: count, dtype: int64
```