In [2]:

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
```

In [3]:

```
fraud=pd.read_csv('Fraud.csv')
```

In [4]:

```
fraud.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 [5]:

```
fraud.head()
```

Out[5]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbal
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]:

```
fraud.isna().any()
```

Out[6]:

```
step              False
type              False
amount            False
nameOrig          False
oldbalanceOrg     False
newbalanceOrig    False
nameDest          False
oldbalanceDest    False
newbalanceDest    False
isFraud           False
isFlaggedFraud    False
dtype: bool
```

In [7]:

```
fraud.isna().sum()
```

Out[7]:

```
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 [8]:

```
fraud.describe()
```

Out[8]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceD
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224996e+
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674129e+
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146614e+
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111909e+
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561793e+

In [9]:

```
fraud.shape
```

Out[9]:

```
(6362620, 11)
```

In [10]:

```
fraud.columns
```

Out[10]:

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

In [11]:

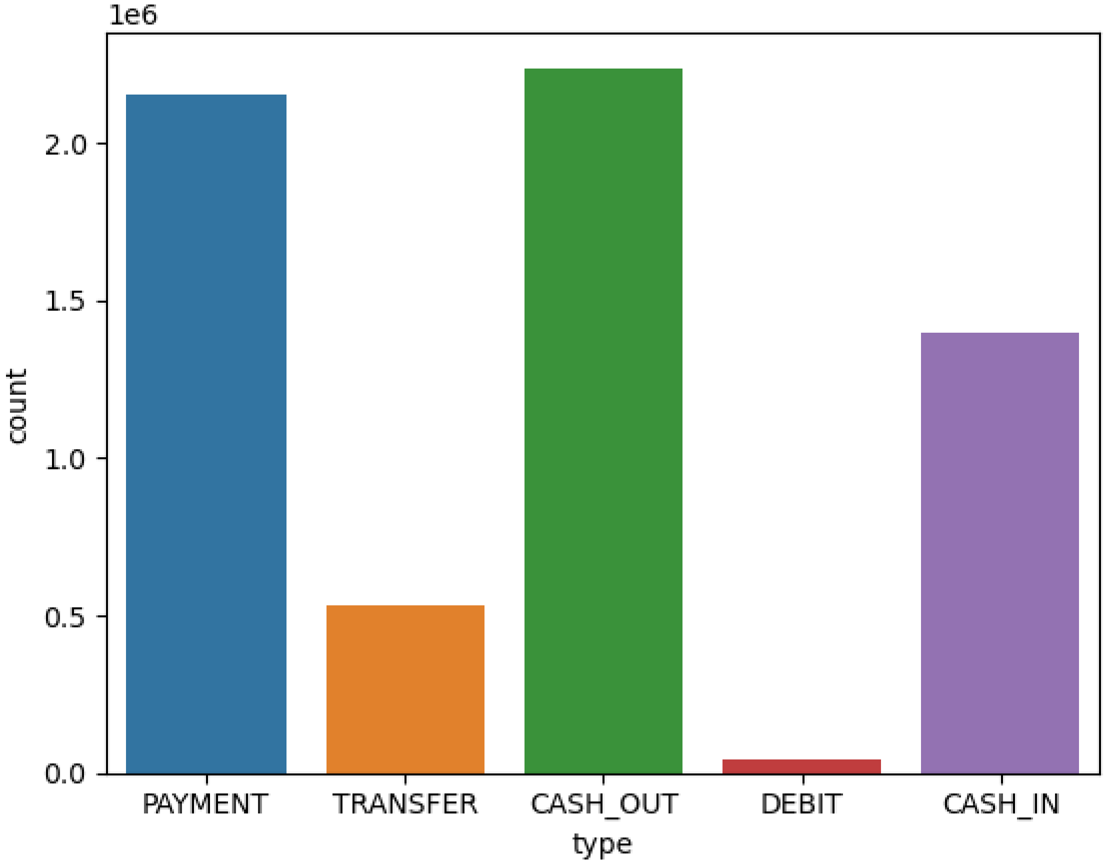
```
import seaborn as sns
```

In [12]:

```
sns.countplot(x='type',data=fraud)
```

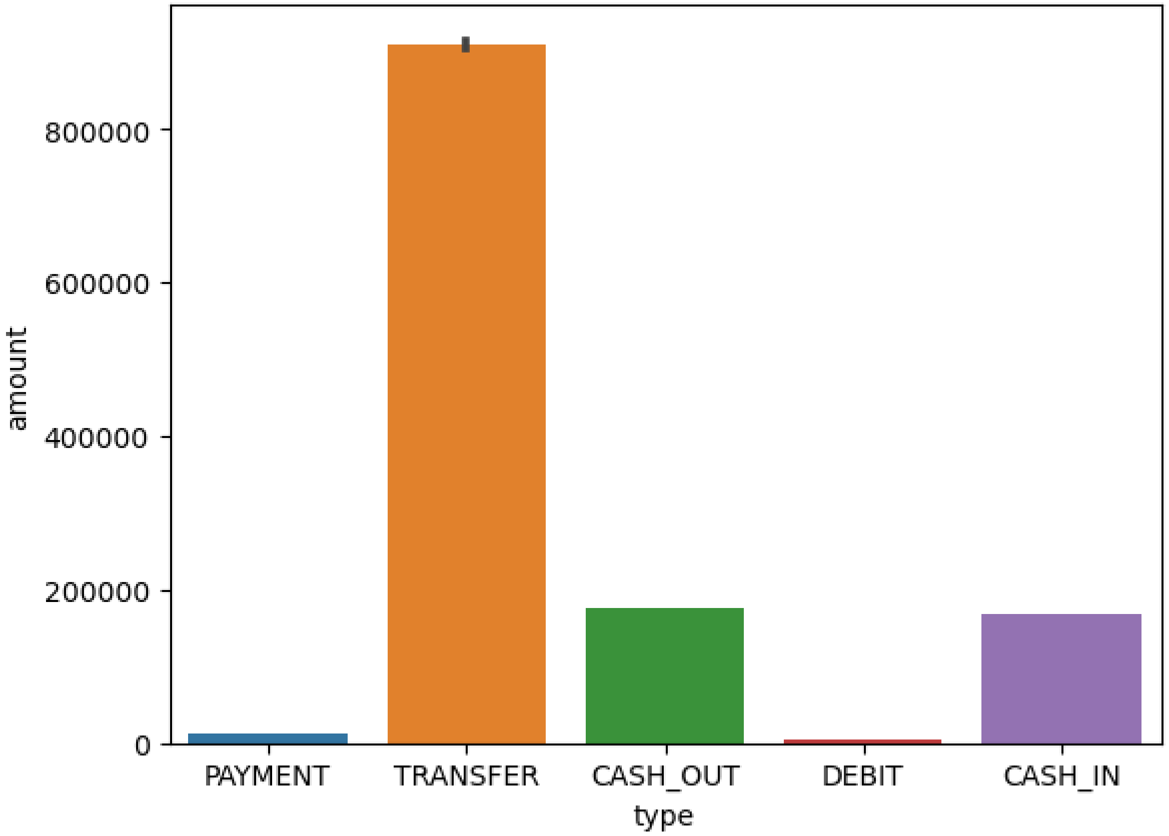
Out[12]:

```
<AxesSubplot:xlabel='type', ylabel='count'>
```



```
In [13]: sns.barplot(x='type',y='amount',data=fraud)
```

```
Out[13]: <AxesSubplot:xlabel='type', ylabel='amount'>
```



```
In [14]: # Both the graph clearly shows that mostly the type cash_out and transfer are maxim
```

```
In [15]: fraud['isFraud'].value_counts()
```

Out[15]:

```
0    6354407
1       8213
Name: isFraud, dtype: int64
```

In [16]:

```
#the dataset is not in same count.so there is a need of sampling
```

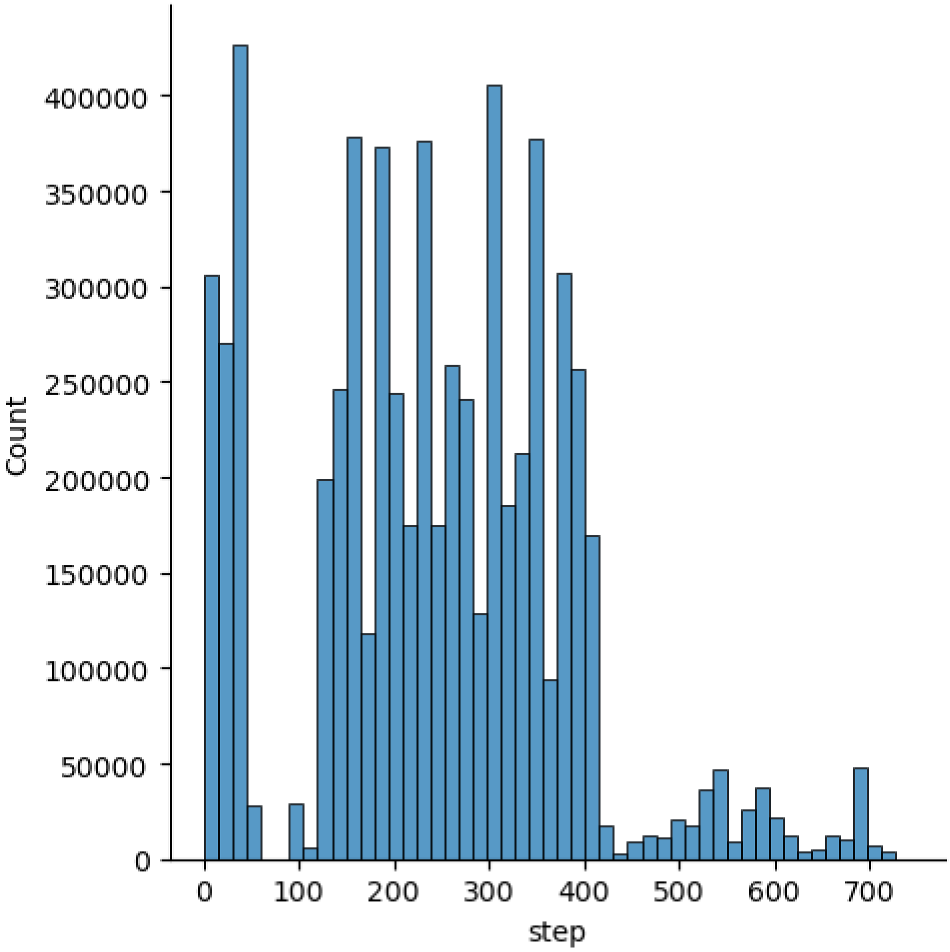
In [17]:

```
#Distribution of step column using displot
```

In [18]:

```
plt.figure(figsize=(15,6))
sns.displot(fraud['step'],bins=50)
plt.show()
```

```
<Figure size 1500x600 with 0 Axes>
```



In [19]:

```
#The graph shows the maximum distribution between 200 to 400
```

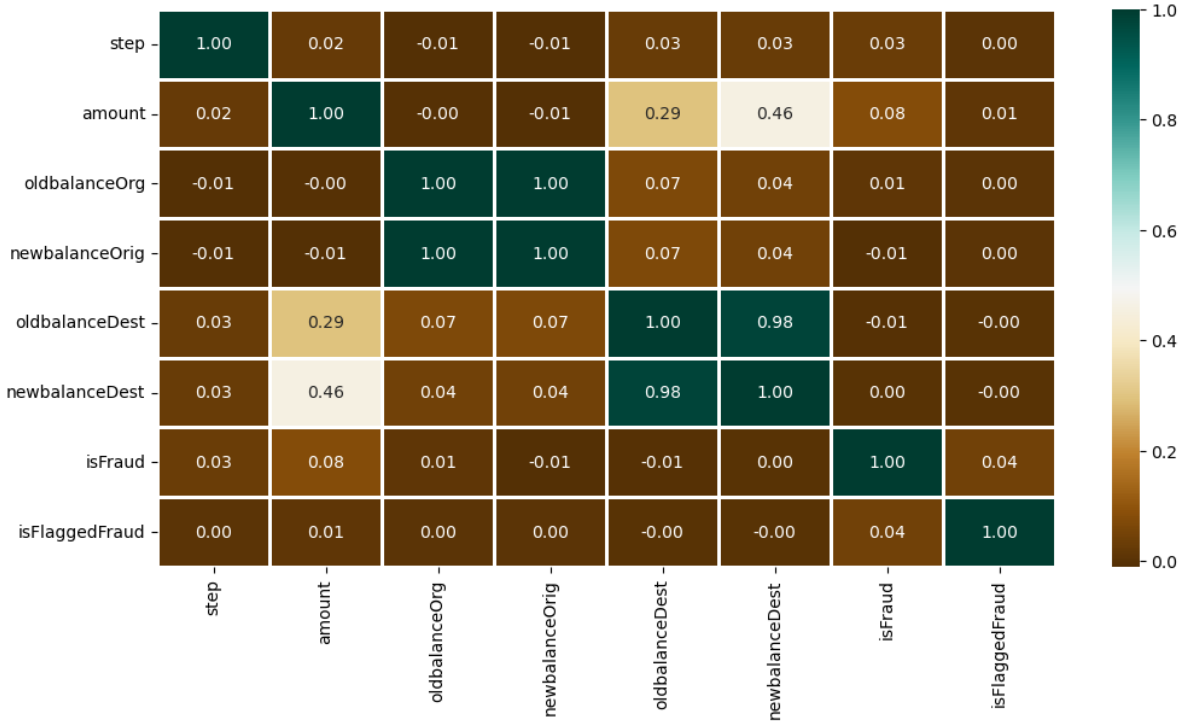
In [20]:

```
#Heatmap-finding cooorelation between features

plt.figure(figsize=(12,6))
sns.heatmap(fraud.corr(),
            cmap='BrBG',
            fmt='.2f',
            linewidths=2,
            annot=True)
```

Out[20]:

```
<AxesSubplot:>
```



In [21]:

```
"""data preprocessing
this involves:
    1.Encoding of type columns
    2.dropping irrevelant columns like nameOrg,nameDest
    3.data splitting"""
```

Out[21]:

```
'data preprocessing\nthis involves:\n    1.Encoding of type columns\n    2.droppin
g irrevelant columns like nameOrg,nameDest\n    3.data splitting'
```

In [22]:

```
type_new = pd.get_dummies(fraud['type'], drop_first=True)
data_new = pd.concat([fraud, type_new], axis=1)
data_new.head()
```

Out[22]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbal
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 [23]:

```
X = data_new.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
y = data_new['isFraud']
```

In [24]:

```
X.shape,y.shape
```

Out[24]:

```
((6362620, 11), (6362620,))
```

In [25]:

```
#training and testing
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
```

In [26]:

```
!pip install pandas scikit-learn xgboost
```

```
Requirement already satisfied: pandas in c:\users\ankii\anaconda3\lib\site-package
s (1.4.4)
Requirement already satisfied: scikit-learn in c:\users\ankii\anaconda3\lib\site-p
ackages (1.0.2)
Requirement already satisfied: xgboost in c:\users\ankii\anaconda3\lib\site-packag
es (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\ankii\anaconda3
\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\ankii\anaconda3\lib\site-p
ackages (from pandas) (2022.1)
Requirement already satisfied: numpy>=1.18.5 in c:\users\ankii\anaconda3\lib\site-
packages (from pandas) (1.21.5)
Requirement already satisfied: joblib>=0.11 in c:\users\ankii\anaconda3\lib\site-p
ackages (from scikit-learn) (1.1.0)
Requirement already satisfied: scipy>=1.1.0 in c:\users\ankii\anaconda3\lib\site-p
ackages (from scikit-learn) (1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\ankii\anaconda3\li
b\site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\ankii\anaconda3\lib\site-packa
ges (from python-dateutil>=2.8.1->pandas) (1.16.0)
```

In [27]:

```python
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score as ras
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:

```python
models = [LogisticRegression(), XGBClassifier(),
          SVC(kernel='rbf', probability=True),
          RandomForestClassifier(n_estimators=7,
                                 criterion='entropy',
                                 random_state=7)]

for i in range(len(models)):
    models[i].fit(X_train, y_train)
    print(f'{models[i]} : ')

    train_preds = models[i].predict_proba(X_train)[:, 1]
    print('Training Accuracy : ', ras(y_train, train_preds))

    y_preds = models[i].predict_proba(X_test)[:, 1]
    print('Validation Accuracy : ', ras(y_test, y_preds))
    print()
```

```
LogisticRegression() :
Training Accuracy :  0.8873943416757113
Validation Accuracy :  0.8849915552513516

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...) :
Training Accuracy :  0.9999774189140321
Validation Accuracy :  0.999212631773824
```

In [ ]:

```
```