```
Importing Libraries
```

In [2]:
```
import numpy as np
import pandas as pd
```

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

Out[3]:

	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 [4]:
```
df.shape
```

Out[4]:
```
(6362620, 11)
```

Finding Missing Values

In [5]:
```
df.isnull().values.any()
```

Out[5]:
```
False
```

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.describe()
```

Out[7]:

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

In [8]:
```
legit = len(df[df.isFraud == 0])
fraud = len(df[df.isFraud == 1])
legit_percent = (legit / (fraud + legit)) * 100
fraud_percent = (fraud / (fraud + legit)) * 100

print("Number of Legit transactions: ", legit)
print("Number of Fraud transactions: ", fraud)
print("Percentage of Legit transactions: {:.4f} %".format(legit_percent))
print("Percentage of Fraud transactions: {:.4f} %".format(fraud_percent))
```

```
Number of Legit transactions:  6354407
Number of Fraud transactions:  8213
Percentage of Legit transactions: 99.8709 %
Percentage of Fraud transactions: 0.1291 %
```

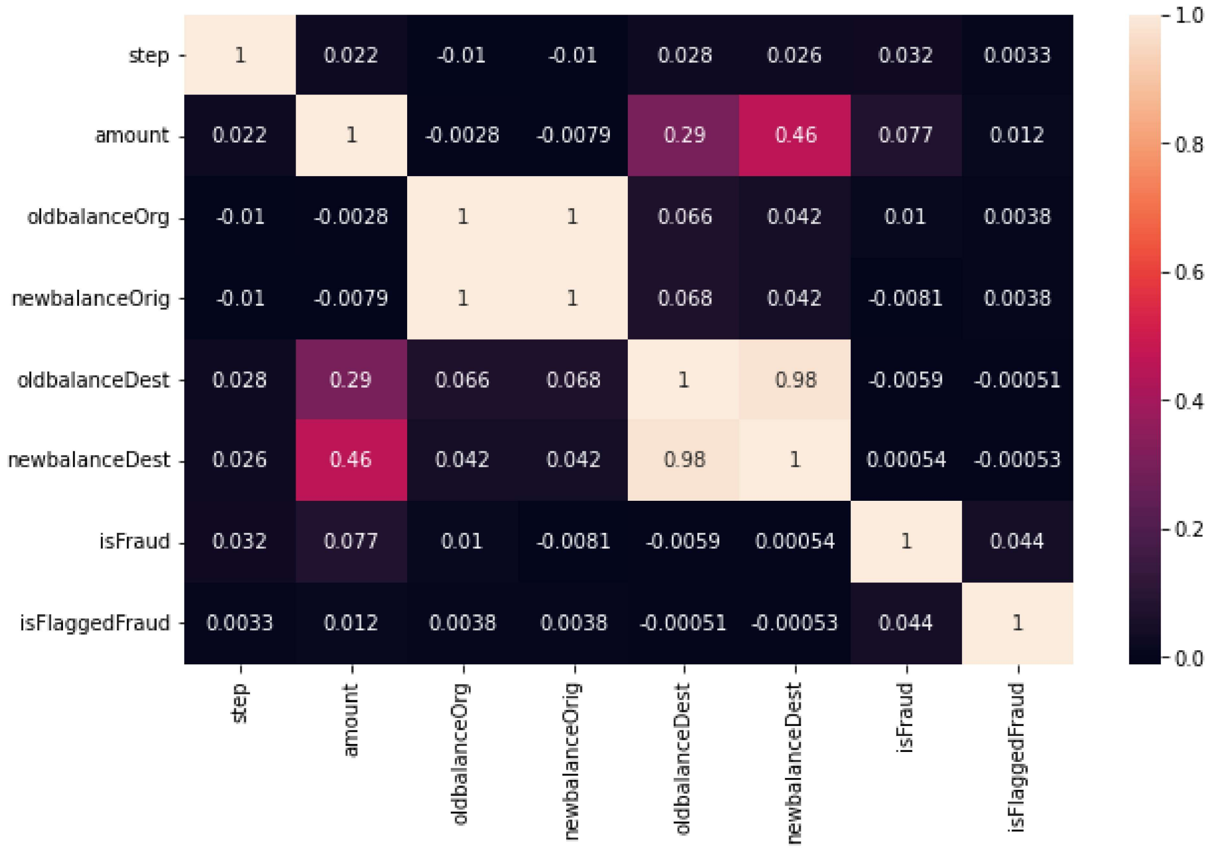
These results prove that this is a highly unbalanced data as Percentage of Legit transactions= 99.87 % and Percentage of Fraud transactions= 0.1291 % . Therefore Decision tree and random forest are good methods for this kind of unbalanced data

In [9]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

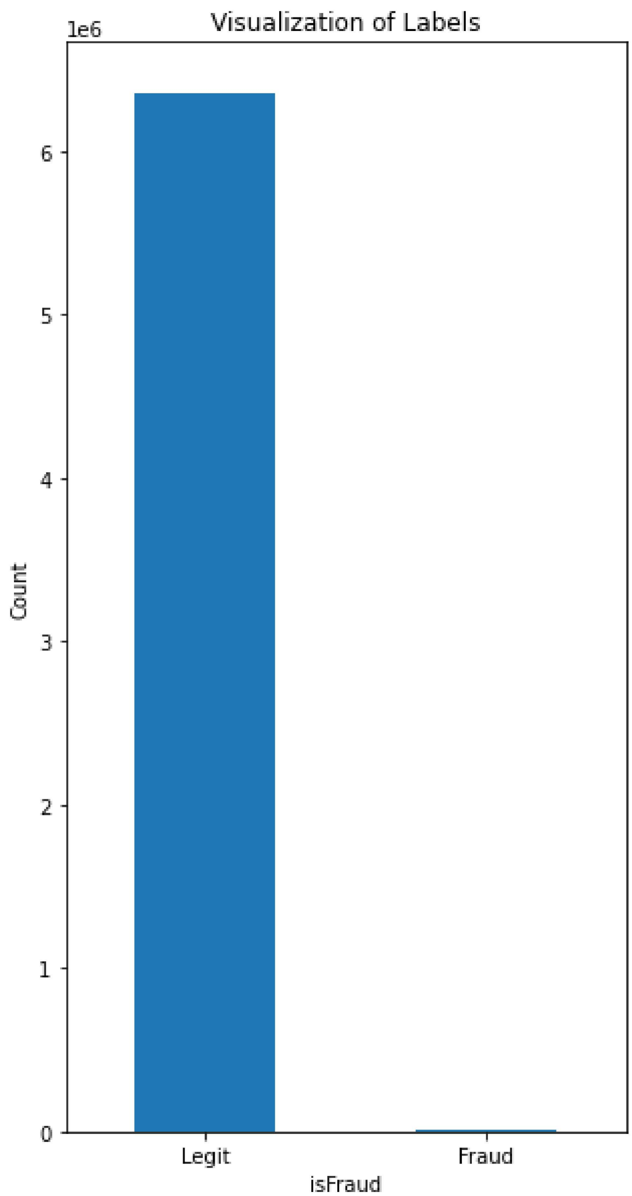
In [10]:
```
corr=df.corr()

plt.figure(figsize=(10,6))
sns.heatmap(corr,annot=True)
```

Out[10]: <AxesSubplot:>

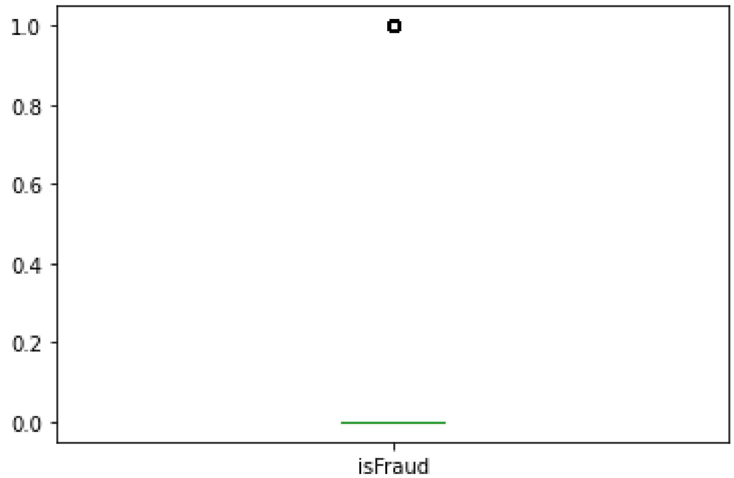


In [11]:
```
plt.figure(figsize=(5,10))
labels = ["Legit", "Fraud"]
count_classes = df.value_counts(df['isFraud'], sort= True)
count_classes.plot(kind = "bar", rot = 0)
plt.title("Visualization of Labels")
plt.ylabel("Count")
plt.xticks(range(2), labels)
plt.show()
```



In [12]:
```
df['isFraud'].plot.box()
```

Out[12]: <AxesSubplot:>



In [13]:
```
og_df=df.copy()
og_df.head()
```

Out[13]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbala
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 [14]:
```
objList = og_df.select_dtypes(include = "object").columns
print (objList)
```

```
Index(['type', 'nameOrig', 'nameDest'], dtype='object')
```

In [15]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for feat in objList:
    og_df[feat] = le.fit_transform(og_df[feat].astype(str))

print (og_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            int32  
 2   amount          float64
 3   nameOrig        int32  
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        int32  
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64  
 10  isFlaggedFraud  int64  
dtypes: float64(5), int32(3), int64(3)
memory usage: 461.2 MB
None
```

In [16]:
```
og_df.head()
```

Out[16]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	ne\
0	1	3	9839.64	757869	170136.0	160296.36	1662094	0.0	
1	1	3	1864.28	2188998	21249.0	19384.72	1733924	0.0	
2	1	4	181.00	1002156	181.0	0.00	439685	0.0	
3	1	1	181.00	5828262	181.0	0.00	391696	21182.0	
4	1	3	11668.14	3445981	41554.0	29885.86	828919	0.0	

Finding Multicolinearity

In [17]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(df):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = df.columns
    vif["VIF"] = [variance_inflation_factor(df.values, i) for i in range(df.shape

    return(vif)

calc_vif(og_df)
```

Out[17]:

	variables	VIF
0	step	2.791610
1	type	4.467405
2	amount	4.149312
3	nameOrig	2.764234
4	oldbalanceOrg	576.803777
5	newbalanceOrig	582.709128
6	nameDest	3.300975
7	oldbalanceDest	73.349937
8	newbalanceDest	85.005614
9	isFraud	1.195305
10	isFlaggedFraud	1.002587

In [19]:
```
og_df['Actual_amount_orig'] = og_df.apply(lambda x: x['oldbalanceOrg'] - x['newba
```

In [21]:
```
og_df['Actual_amount_dest'] = og_df.apply(lambda x: x['oldbalanceDest'] - x['newb
```

In [22]:
```
og_df['TransactionPath'] = og_df.apply(lambda x: x['nameOrig'] + x['nameDest'],ax
```

In [23]:
```
og_df = og_df.drop(['oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalance

calc_vif(og_df)
```

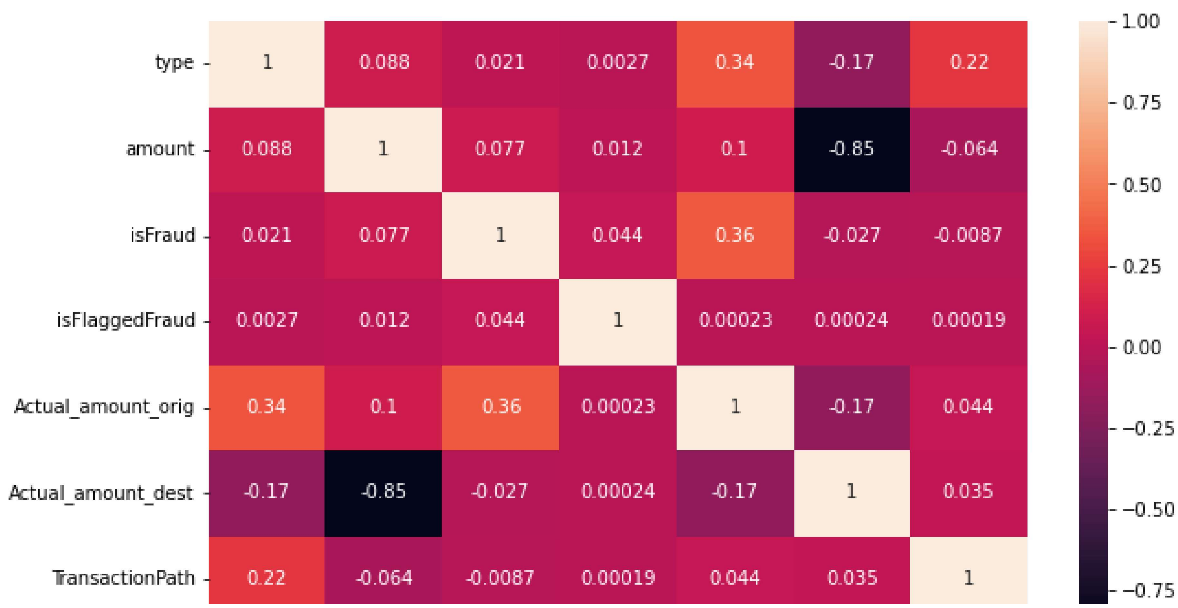
Out[23]:

	variables	VIF
0	type	2.687803
1	amount	3.818902
2	isFraud	1.184479
3	isFlaggedFraud	1.002546
4	Actual_amount_orig	1.307910
5	Actual_amount_dest	3.754335
6	TransactionPath	2.677167

In [24]:
```
corr=og_df.corr()

plt.figure(figsize=(10,6))
sns.heatmap(corr,annot=True)
```

Out[24]: <AxesSubplot:>



Using Variance Inflation Factor and Corr Heatmap I can check whether any of the two variables are highly correlated to each other and can drop one which is less correlated

Model Selection

In [25]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
import itertools
from collections import Counter
import sklearn.metrics as metrics
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMat
```

In [27]:
```
scaler = StandardScaler()
og_df["NormalizedAmount"] = scaler.fit_transform(og_df["amount"].values.reshape(-
og_df.drop(["amount"], inplace= True, axis= 1)

Y = og_df["isFraud"]
X = og_df.drop(["isFraud"], axis= 1)
```

In [28]:
```
(X_train, X_test, Y_train, Y_test) = train_test_split(X, Y, test_size= 0.3, rando

print("Shape of X_train: ", X_train.shape)
print("Shape of X_test: ", X_test.shape)
```

```
Shape of X_train:  (4453834, 6)
Shape of X_test:  (1908786, 6)
```

In [29]:
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)

Y_pred_dt = decision_tree.predict(X_test)
decision_tree_score = decision_tree.score(X_test, Y_test) * 100
```

In [32]:
```
random_forest = RandomForestClassifier(n_estimators= 100)
random_forest.fit(X_train, Y_train)
```

Out[32]:
```
RandomForestClassifier()
```

In [33]:
```
Y_pred_rf = random_forest.predict(X_test)
random_forest_score = random_forest.score(X_test, Y_test) * 100
```

In [34]:
```
print("Decision Tree Score: ", decision_tree_score)
print("Random Forest Score: ", random_forest_score)
```

```
Decision Tree Score:  99.92372115051137
Random Forest Score:  99.95887438403257
```

In [35]:
```
print("TP,FP,TN,FN - Decision Tree")
tn, fp, fn, tp = confusion_matrix(Y_test, Y_pred_dt).ravel()
print(f'True Positives: {tp}')
print(f'False Positives: {fp}')
print(f'True Negatives: {tn}')
print(f'False Negatives: {fn}')

print("--------------------------------------------------------------------------

# key terms of Confusion Matrix - RF

print("TP,FP,TN,FN - Random Forest")
tn, fp, fn, tp = confusion_matrix(Y_test, Y_pred_rf).ravel()
print(f'True Positives: {tp}')
print(f'False Positives: {fp}')
print(f'True Negatives: {tn}')
print(f'False Negatives: {fn}')
```

```
TP,FP,TN,FN - Decision Tree
True Positives: 1718
False Positives: 739
True Negatives: 1905612
False Negatives: 717
--------------------------------------------------------------------------------
---------
TP,FP,TN,FN - Random Forest
True Positives: 1710
False Positives: 60
True Negatives: 1906291
False Negatives: 725
```

Hence Random Forest Score is overall good then Decision Tree

What are the key factors that predict fraudulent customer?

Transaction history of customers Name of organization is spam or not ?

```
What kind of prevention should be adopted while company update its
infrastructure?

Use verified apps and software
Should use safe internet connections and browse secured sites
```

```
Assuming these actions have been implemented, how would you determine if they
work?

Keep eye on your banking history
Keep changing your security password
```

In [ ]: