Introduction:

The fraud detection logs were provided by a multinational company, who is the provider of the mobile financial service which is currently running in more than 14 countries all around the world.

This is a sample of 1 row with headers explanation:

1,PAYMENT,1060.31,C429214117,1089.0,28.69,M1591654462,0.0,0.0,0,0

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

oldbalanceOrg - 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.

Importing pandas,

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn import tree, metrics
from io import StringIO
from IPython.display import Image
import pydotplus
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
```

Loading financial dataset:

In [3]:
```
filepath='C:\\Users\\61435\\Desktop\\SpringBoard_Assignments\\Capstone2-Fraud
 Detection\\Capstone2-Fraud-Detection\\Financial Datasets For Fraud Detection.
csv'
df= pd.read_csv(filepath)
```

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

Out[4]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldb
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	

Check for the data type of all columns:

In [4]:
```
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
```

A.Data Cleaning and Wrangling

Check for the missing data:

In [5]:
```
missing = pd.concat([df.isnull().sum(), 100 * df.isnull().mean()], axis=1)
missing.columns=['count', '%']
missing.sort_values(by='count',ascending=False)
```

Out[5]:

	count	%
step	0	0.0
type	0	0.0
amount	0	0.0
nameOrig	0	0.0
oldbalanceOrg	0	0.0
newbalanceOrig	0	0.0
nameDest	0	0.0
oldbalanceDest	0	0.0
newbalanceDest	0	0.0
isFraud	0	0.0
isFlaggedFraud	0	0.0

Conclusion:No missing data found. Analysing each features.Firstly,string data

In [6]:
```
df.select_dtypes('object')
```

Out[6]:

	type	nameOrig	nameDest
0	PAYMENT	C1231006815	M1979787155
1	PAYMENT	C1666544295	M2044282225
2	TRANSFER	C1305486145	C553264065
3	CASH_OUT	C840083671	C38997010
4	PAYMENT	C2048537720	M1230701703
...	...	...	...
6362615	CASH_OUT	C786484425	C776919290
6362616	TRANSFER	C1529008245	C1881841831
6362617	CASH_OUT	C1162922333	C1365125890
6362618	TRANSFER	C1685995037	C2080388513
6362619	CASH_OUT	C1280323807	C873221189

6362620 rows × 3 columns

Now let's check the numeric features.

In [7]:
```
df.select_dtypes(include = ['int64','float64'])
```

Out[7]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isF
0	1	9839.64	170136.00	160296.36	0.00	0.00	
1	1	1864.28	21249.00	19384.72	0.00	0.00	
2	1	181.00	181.00	0.00	0.00	0.00	
3	1	181.00	181.00	0.00	21182.00	0.00	
4	1	11668.14	41554.00	29885.86	0.00	0.00	
...	...	...	...	...	...	...	
6362615	743	339682.13	339682.13	0.00	0.00	339682.13	
6362616	743	6311409.28	6311409.28	0.00	0.00	0.00	
6362617	743	6311409.28	6311409.28	0.00	68488.84	6379898.11	
6362618	743	850002.52	850002.52	0.00	0.00	0.00	
6362619	743	850002.52	850002.52	0.00	6510099.11	7360101.63	

6362620 rows × 8 columns

B.Exploratory Data Analysis (EDA)

1. DATA PROFILES — PLOTS AND TABLES

Conduct EDA on the 'Capstone2-Fraud Detection' to examine relationships between variables and other patterns in the data.

In [8]:
```
df.describe().T
```

Out[8]:

	count	mean	std	min	25%	50%	75'
step	6362620.0	2.433972e+02	1.423320e+02	1.0	156.00	239.000	3.350000e+0
amount	6362620.0	1.798619e+05	6.038582e+05	0.0	13389.57	74871.940	2.087215e+0
oldbalanceOrg	6362620.0	8.338831e+05	2.888243e+06	0.0	0.00	14208.000	1.073152e+0
newbalanceOrig	6362620.0	8.551137e+05	2.924049e+06	0.0	0.00	0.000	1.442584e+0
oldbalanceDest	6362620.0	1.100702e+06	3.399180e+06	0.0	0.00	132705.665	9.430367e+0
newbalanceDest	6362620.0	1.224996e+06	3.674129e+06	0.0	0.00	214661.440	1.111909e+0
isFraud	6362620.0	1.290820e-03	3.590480e-02	0.0	0.00	0.000	0.000000e+0
isFlaggedFraud	6362620.0	2.514687e-06	1.585775e-03	0.0	0.00	0.000	0.000000e+0

In [9]:
```
df[df['isFraud']==1]
```

Out[9]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameD
2	1	TRANSFER	181.00	C1305486145	181.00	0.0	C553264(
3	1	CASH_OUT	181.00	C840083671	181.00	0.0	C389970
251	1	TRANSFER	2806.00	C1420196421	2806.00	0.0	C9727658
252	1	CASH_OUT	2806.00	C2101527076	2806.00	0.0	C10072517
680	1	TRANSFER	20128.00	C137533655	20128.00	0.0	C1848415(
...	...	...	...	...	...	...	
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C7769192
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C18818418
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C13651258
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C2080388(
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C8732211

8213 rows × 11 columns

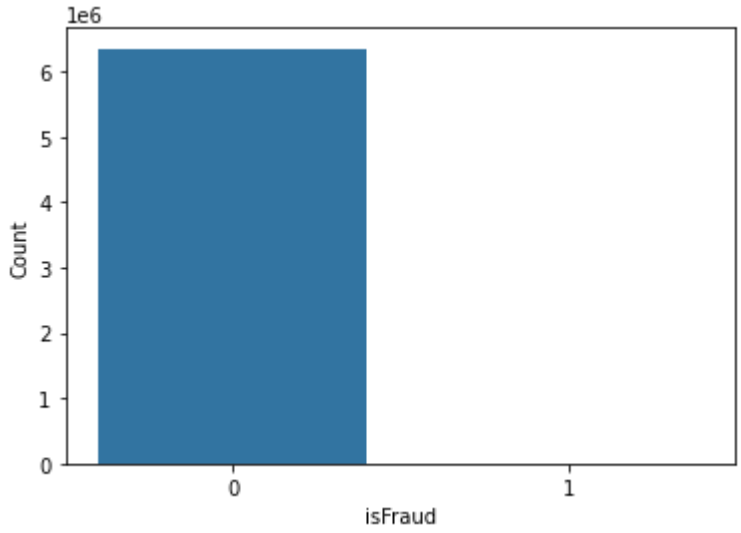
In [10]:
```
df[df['isFlaggedFraud']==1]
```

Out[10]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameI
2736446	212	TRANSFER	4953893.08	C728984460	4953893.08	4953893.08	C639921
3247297	250	TRANSFER	1343002.08	C1100582606	1343002.08	1343002.08	C1147517
3760288	279	TRANSFER	536624.41	C1035541766	536624.41	536624.41	C1100697
5563713	387	TRANSFER	4892193.09	C908544136	4892193.09	4892193.09	C89114(
5996407	425	TRANSFER	10000000.00	C689608084	19585040.37	19585040.37	C1392803
5996409	425	TRANSFER	9585040.37	C452586515	19585040.37	19585040.37	C1109166
6168499	554	TRANSFER	3576297.10	C193696150	3576297.10	3576297.10	C484597
6205439	586	TRANSFER	353874.22	C1684585475	353874.22	353874.22	C1770418
6266413	617	TRANSFER	2542664.27	C786455622	2542664.27	2542664.27	C661958
6281482	646	TRANSFER	10000000.00	C19004745	10399045.08	10399045.08	C1806199
6281484	646	TRANSFER	399045.08	C724693370	10399045.08	10399045.08	C1909486
6296014	671	TRANSFER	3441041.46	C917414431	3441041.46	3441041.46	C1082139
6351225	702	TRANSFER	3171085.59	C1892216157	3171085.59	3171085.59	C130806(
6362460	730	TRANSFER	10000000.00	C2140038573	17316255.05	17316255.05	C1395467
6362462	730	TRANSFER	7316255.05	C1869569059	17316255.05	17316255.05	C1861208
6362584	741	TRANSFER	5674547.89	C992223106	5674547.89	5674547.89	C1366804

In [11]:
```
print(df.isFraud.value_counts())
sns.countplot(data=df, x='isFraud')
plt.ylabel('Count')
plt.show()
```

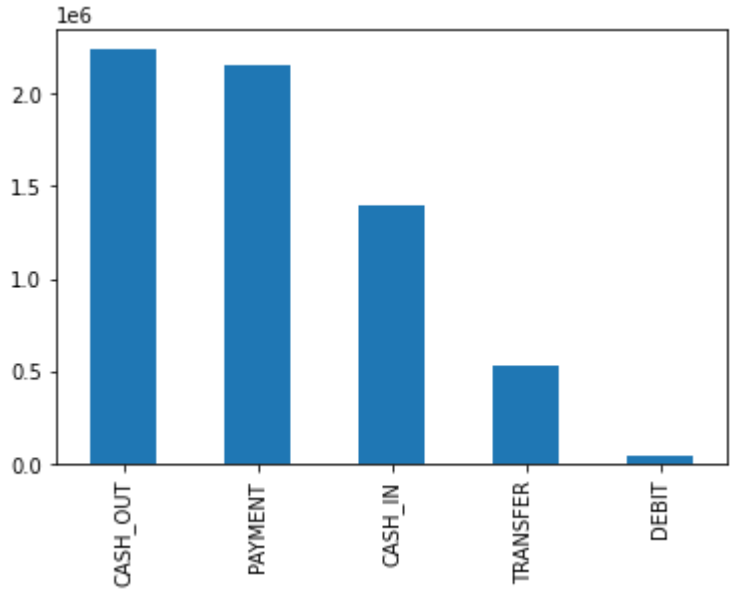
```
0    6354407
1       8213
Name: isFraud, dtype: int64
```



Count number of each type of transaction

In [12]:
```
print(df.type.value_counts())
df.type.value_counts().plot(kind='bar')
plt.show()
```

```
CASH_OUT    2237500
PAYMENT     2151495
CASH_IN     1399284
TRANSFER     532909
DEBIT         41432
Name: type, dtype: int64
```



Check for relationship between isFraud and isFlaggedFraud:

In [13]:
```
pd.crosstab(df.isFraud,df.isFlaggedFraud)
```

Out[13]:

isFlaggedFraud	0	1
isFraud		
0	6354407	0
1	8197	16

In [14]:
```
df.groupby('type')['isFraud','isFlaggedFraud'].sum()
```

```
C:\Users\61435\anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWar
ning: Indexing with multiple keys (implicitly converted to a tuple of keys) w
ill be deprecated, use a list instead.
  """Entry point for launching an IPython kernel.
```

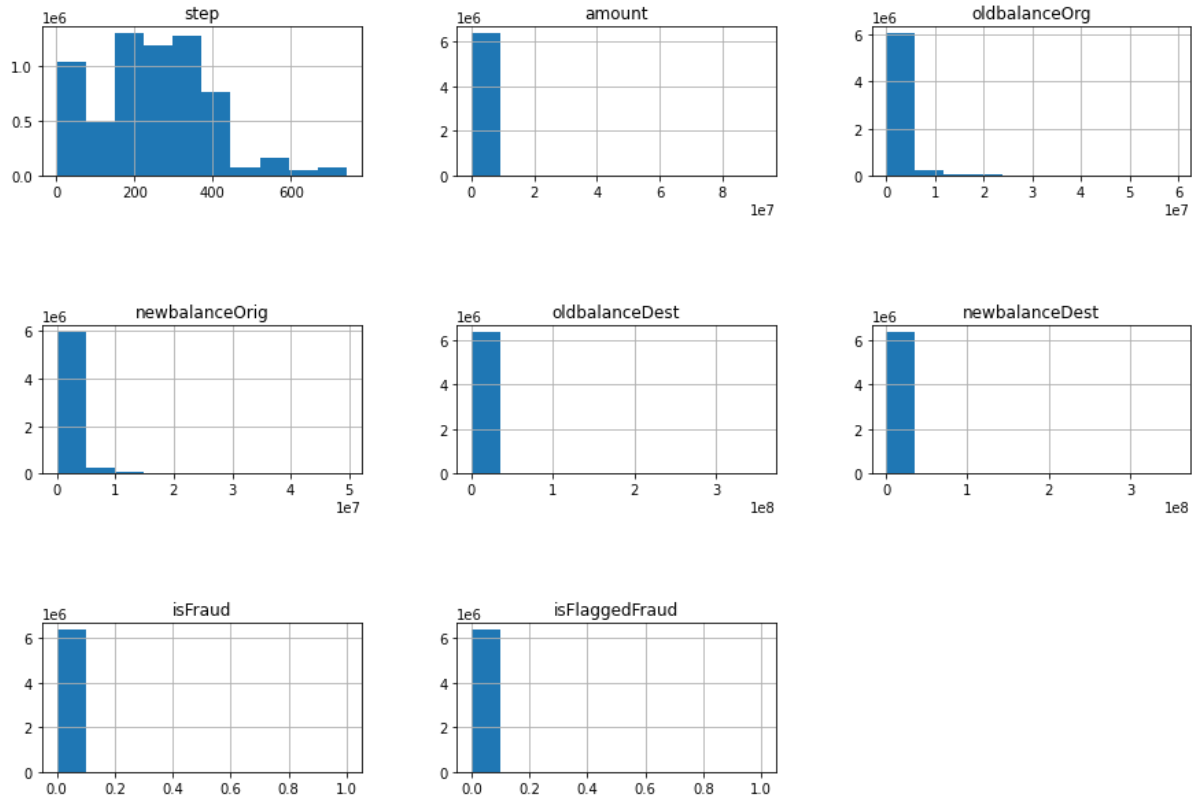
Out[14]:

type	isFraud	isFlaggedFraud
CASH_IN	0	0
CASH_OUT	4116	0
DEBIT	0	0
PAYMENT	0	0
TRANSFER	4097	16

Conclusion:Fraud occurs only in 2 type of transactions: TRANSFER and CASH_OUT The type of transactions in which isFlaggedFraud is set : TRANSFER

Histograms for each variable in df

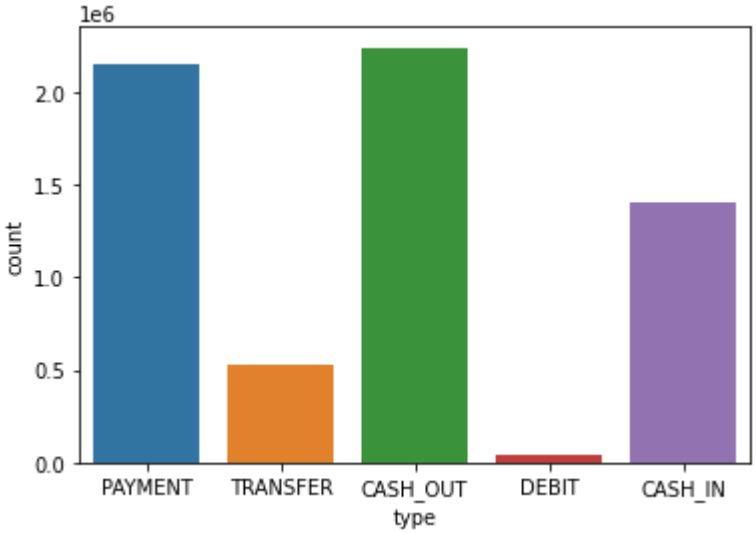
In [15]:
```
df.hist(figsize=(15,10))
plt.subplots_adjust(hspace=1)
```



Categorical variable: type Check for the categorical variable in dataset:

In [16]:
```
#select categorical variable 'type'
df_cat = df.select_dtypes(include = 'object').copy()
#get counts of each variable value
df_cat.type.value_counts()
#count plot for one variable
sns.countplot(data = df_cat, x = 'type')
```

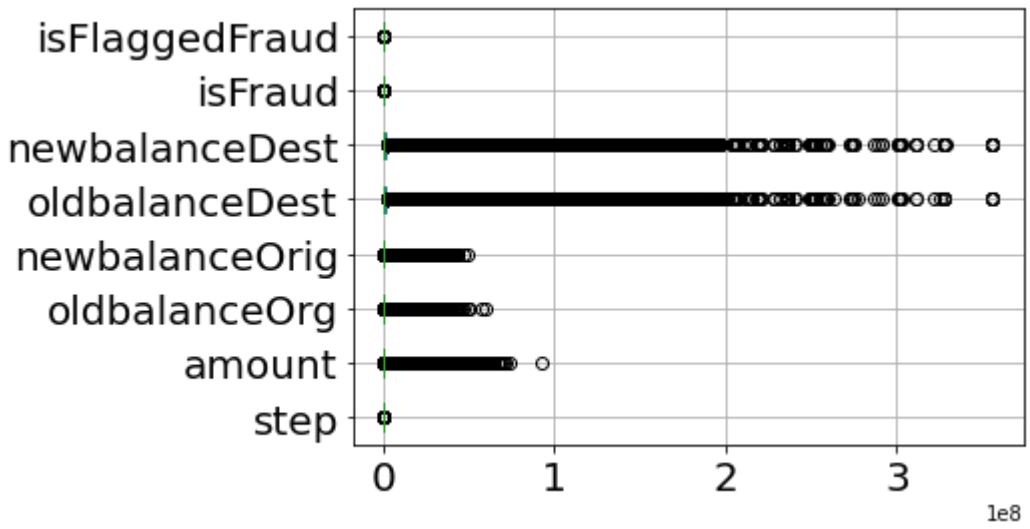
Out[16]: <AxesSubplot:xlabel='type', ylabel='count'>



Reviewing for Outliers and Anamolies

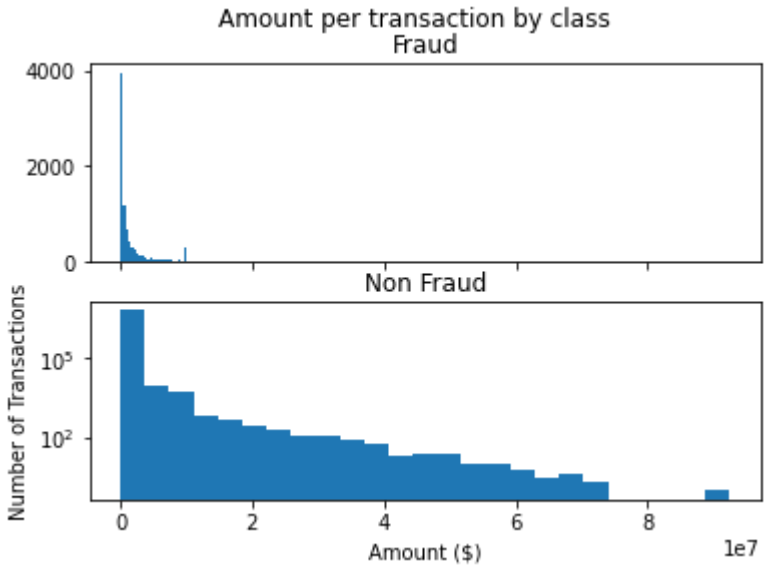
Create a boxplot for every column in df

In [17]:
```
boxplot = df.boxplot(grid=True, vert=False,fontsize=20)
```



Conclusion:In these cases outliers are expected as acounts are not interlinked and Bank allows customers to place all types of transactions.

In [18]:
```
fraud = df[df['isFraud']==1]
nonfraud = df[df['isFraud']==0]
f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per transaction by class')
bins = 25
ax1.hist(fraud.amount, bins = bins)
ax1.set_title('Fraud')
ax2.hist(nonfraud.amount, bins = bins)
ax2.set_title('Non Fraud')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.yscale('log')
plt.show();
```

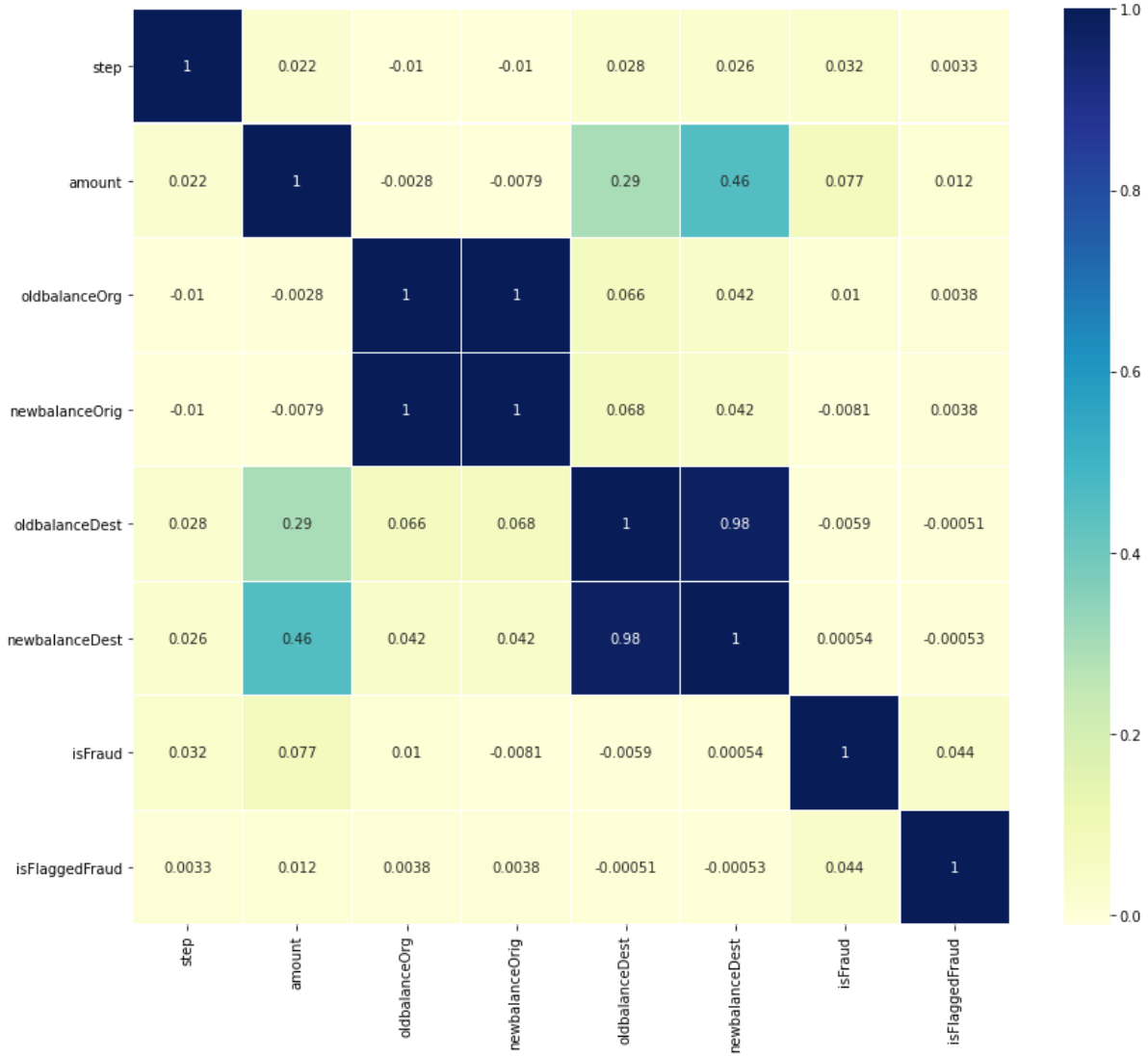


Conclusion:Less number of transaction amount in fraud comapare to non fraud data

2. DATA RELATIONSHIPS

Create the correlation matrix heat map:Correlation coefficients for each variable in the dataset

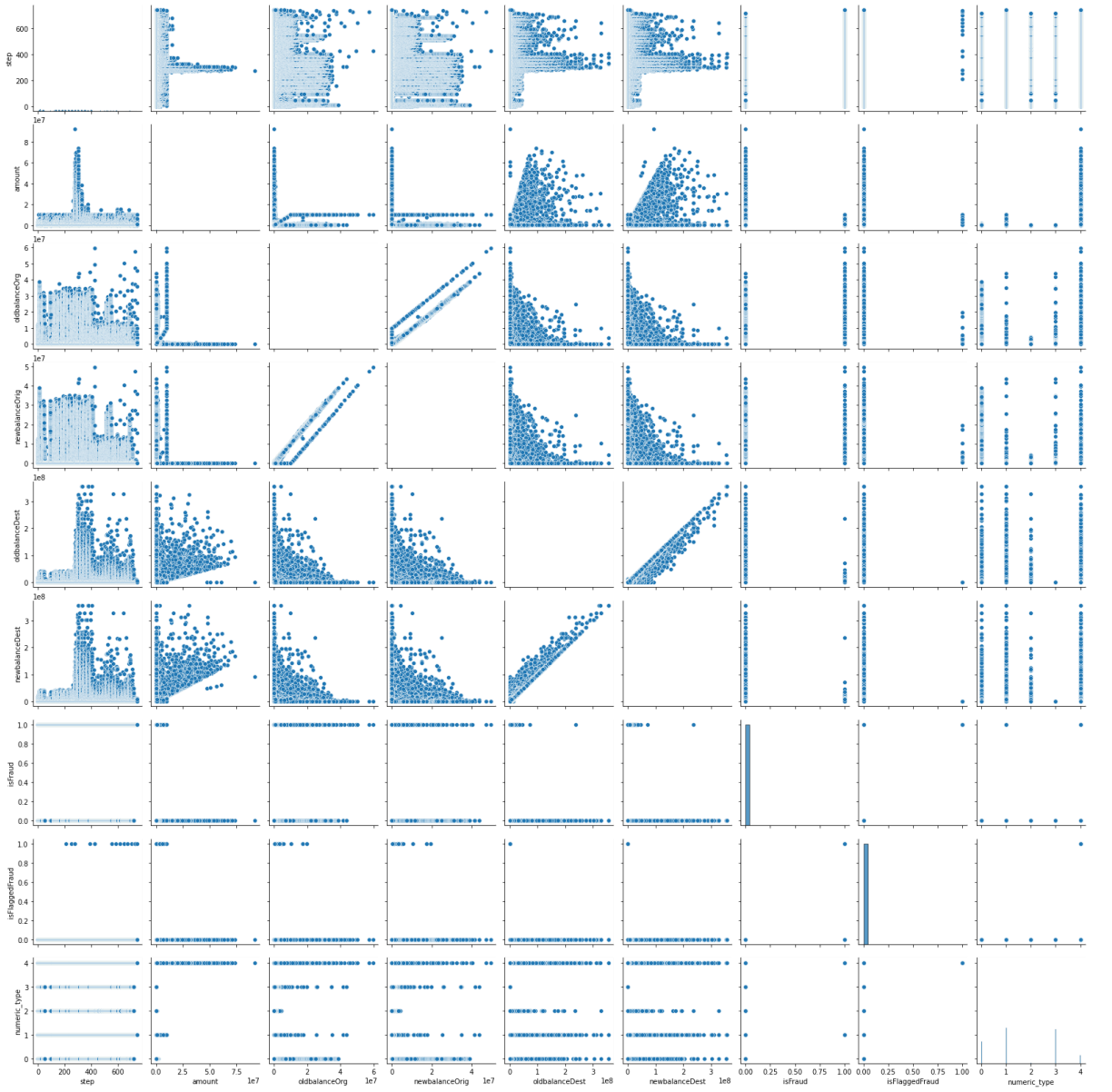
In [19]:
```
plt.figure(figsize=(14,12))
sns.heatmap(df.corr(),linewidths=.1,cmap="YlGnBu", annot=True)
plt.yticks(rotation=0);
```



Conclusion: OldbalanceOrg, NewbalanceOrig are closely related. Similarly, OldbalanceDest, NewbalanceDest are closely related. isFraud is related to amount coloumn. So this will be the coloumn on which we need to work on. Moderately Strong Negative Correlation:NewbalanceDest and amount

Creat pair plots

In [42]:
```
#pair plots
sns.pairplot(df)
plt.show()
```



3. IDENTIFYING AND CREATING FEATURES

Check PCA for the correrated features:

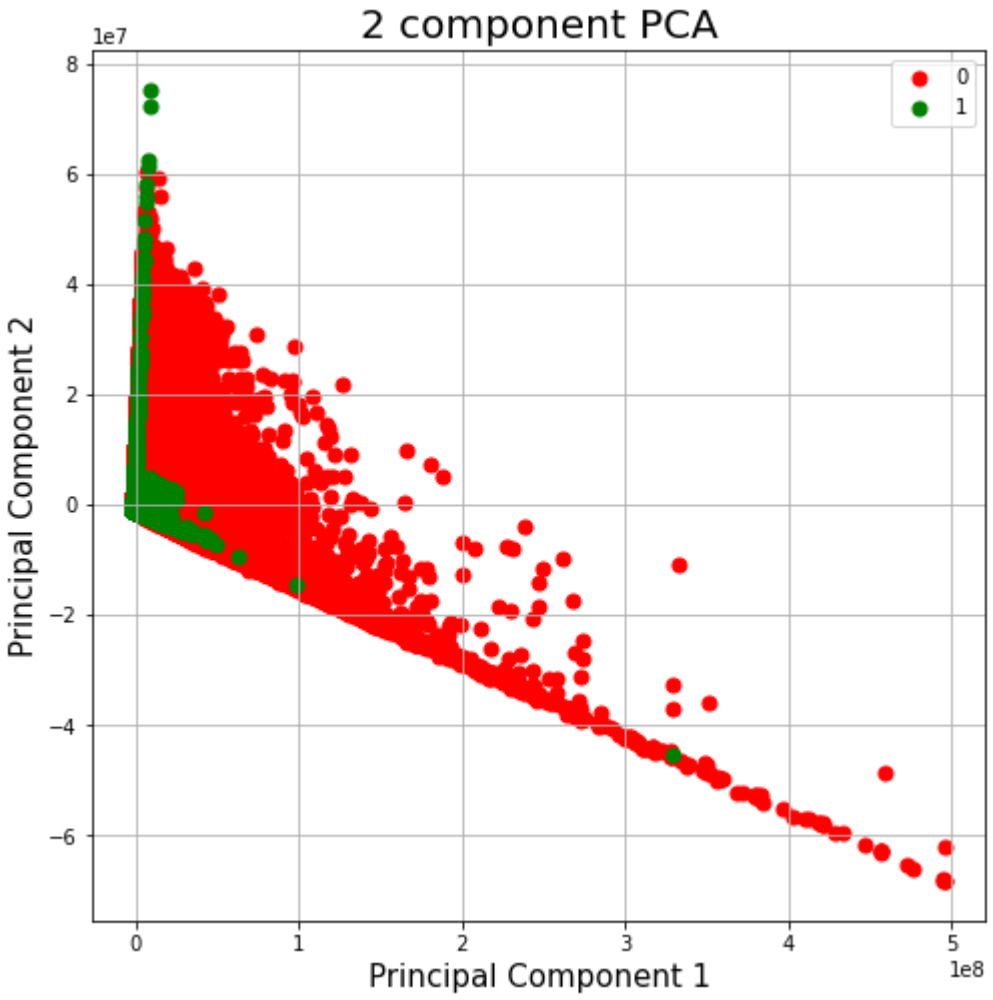
In [12]:
```
features_array = ['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest',
'newbalanceDest']
# Separating out the features
features= df.loc[:, features_array].values
print(features)
# Standardizing the features
standard_features = StandardScaler().fit_transform(features)
print("**************************")
print(standard_features)
```

```
[[9.83964000e+03 1.70136000e+05 1.60296360e+05 0.00000000e+00
  0.00000000e+00]
 [1.86428000e+03 2.12490000e+04 1.93847200e+04 0.00000000e+00
  0.00000000e+00]
 [1.81000000e+02 1.81000000e+02 0.00000000e+00 0.00000000e+00
  0.00000000e+00]
 ...
 [6.31140928e+06 6.31140928e+06 0.00000000e+00 6.84888400e+04
  6.37989811e+06]
 [8.50002520e+05 8.50002520e+05 0.00000000e+00 0.00000000e+00
  0.00000000e+00]
 [8.50002520e+05 8.50002520e+05 0.00000000e+00 6.51009911e+06
  7.36010163e+06]]
**************************
[[-2.81559923e-01 -2.29810037e-01 -2.37621696e-01 -3.23813895e-01
  -3.33411405e-01]
 [-2.94767262e-01 -2.81359380e-01 -2.85812295e-01 -3.23813895e-01
  -3.33411405e-01]
 [-2.97554804e-01 -2.88653782e-01 -2.92441707e-01 -3.23813895e-01
  -3.33411405e-01]
 ...
 [ 1.01539526e+01  1.89649113e+00 -2.92441707e-01 -3.03665258e-01
   1.40302700e+00]
 [ 1.10976490e+00  5.58104668e-03 -2.92441707e-01 -3.23813895e-01
  -3.33411405e-01]
 [ 1.10976490e+00  5.58104668e-03 -2.92441707e-01  1.59138312e+00
   1.66981230e+00]]
```

In [22]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(features)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
```

In [23]:
```
finalDf = pd.concat([principalDf, df[['isFraud']]], axis = 1)
```

In [24]:
```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = [0,1]
colors = ['r', 'g']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['isFraud'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



In [25]:
```
pca.explained_variance_ratio_
```

Out[25]:
```
array([0.59007217, 0.39557502])
```

Conclusion:the first principal component contains 59% of the variance and the second principal component contains 39.55% of the variance. Together, the two components contain 98.55% of the information. Therefore, inneed to include other feature 'type' in consideration.

4. Feature Engineering

Categorical data: As there are no missing data, let's analyse the categorical data in the dataframe.

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

Out[4]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldb
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 [5]:
```
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
```

From above information we can conclude that type, nameOrig and nameDest are of object type.

In [6]:
```
df['type'].unique()
```

Out[6]:
```
array(['PAYMENT', 'TRANSFER', 'CASH_OUT', 'DEBIT', 'CASH_IN'],
      dtype=object)
```

In [7]:
```
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()
enc.fit(df['type'])
df['numeric_type'] = enc.transform(df['type'])
```

In [8]:
```
df.head()
```

Out[8]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldb
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 [9]:
```
df_new=df.drop(['type'],axis=1)
```

In [10]:
```
df_new.head()
```

Out[10]:

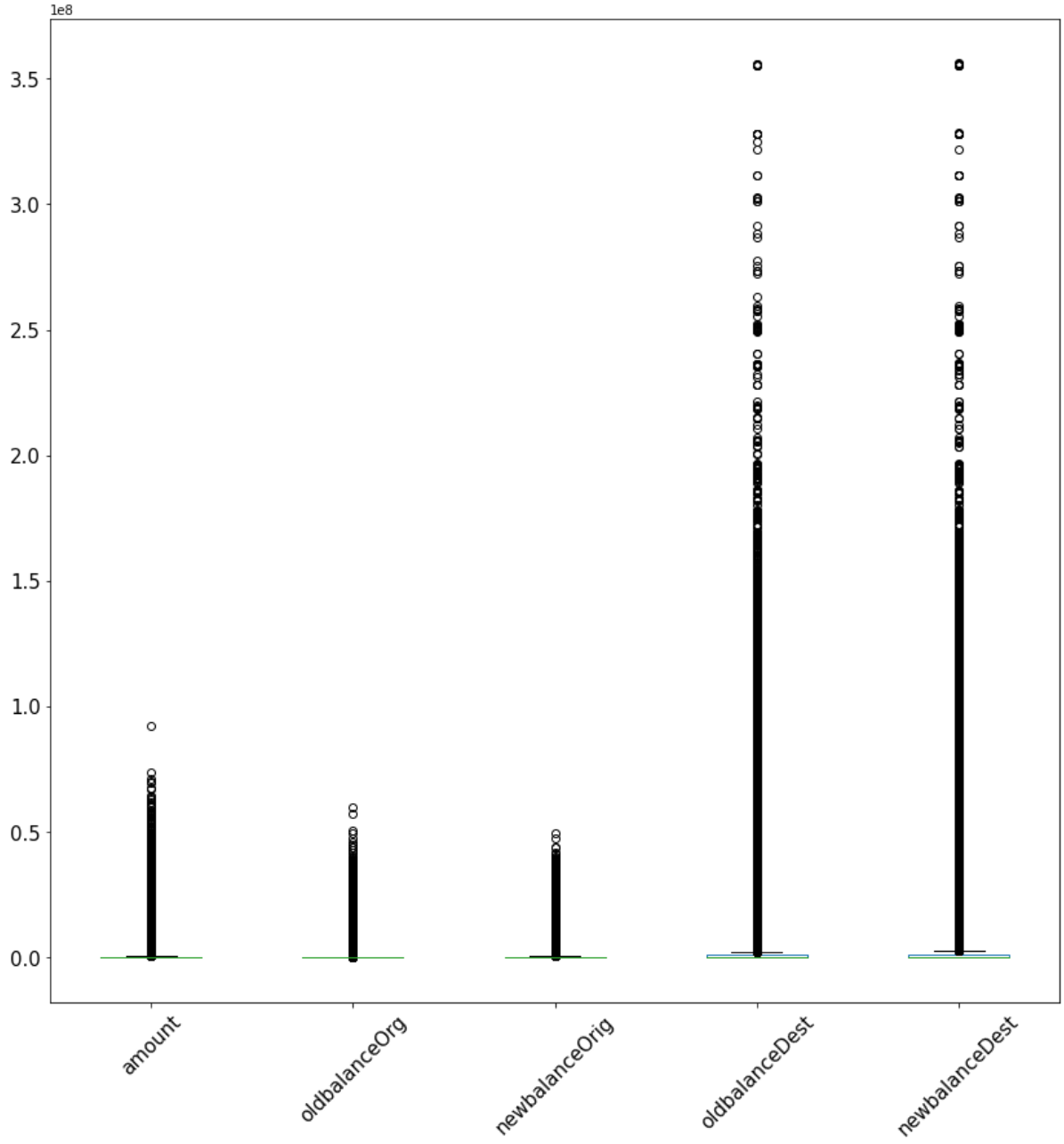
	step	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest
0	1	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0
1	1	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0
2	1	181.00	C1305486145	181.0	0.00	C553264065	0.0
3	1	181.00	C840083671	181.0	0.00	C38997010	21182.0
4	1	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0

In [10]:
```
print(f"% of transactions where difference between opening and closing balance
of customer is not equal to transaction amount is {(1-len(df[np.abs(df.oldbala
nceOrg-df.newbalanceOrig) == (df.amount)])/len(df))*100}")
print(f"% of transactions where difference between opening and closing balance
of recipient is not equal to transaction amount is {(1-len(df[np.abs(df.oldbal
anceDest-df.newbalanceDest) == (df.amount)])/len(df))*100}")
print(f"% of transactions where opening and closing balance of customer is equ
al to 0 but transaction amount is not equal to 0 is {(1-len(df[(df.oldbalanceO
rg==0)&(df.newbalanceOrig==0)&(df.amount!=0)])/len(df))*100}")
print(f"% of transactions where opening and closing balance of recipient is eq
ual to 0 but transaction amount is not equal to 0 is {(1-len(df[(df.oldbalance
Dest==0)&(df.newbalanceDest==0)&(df.amount!=0)])/len(df))*100}")
print("% of transfer transactions where the opening and closing balance of the
customer remained the same is " + str((len(df[(df.type=="CASH_OUT")&(df.oldbal
anceOrg==df.newbalanceOrig)&(df.amount != 0)])/len(df))*100))
```

```
% of transactions where difference between opening and closing balance of cus
tomer is not equal to transaction amount is 85.39760350295947
% of transactions where difference between opening and closing balance of rec
ipient is not equal to transaction amount is 85.05254124873088
% of transactions where opening and closing balance of customer is equal to 0
but transaction amount is not equal to 0 is 67.16810056234695
% of transactions where opening and closing balance of recipient is equal to
0 but transaction amount is not equal to 0 is 63.57984603826725
% of transfer transactions where the opening and closing balance of the custo
mer remained the same is 16.122053493686565
```

Check for outliers

In [13]:
```
df_new[features_array].boxplot(figsize=(14,14),grid=False, rot=45, fontsize=15
)
plt.show()
```



Outliers are excepted for this data set as outliers are expainable.

Conclusion: Only feature on which which was needed to be modified was 'type' as it was categorical in nature.

5. Modelling

Seperating features from target(Defining X,y for modelling)

In [11]:
```
features=['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest','numeric_type','isFlaggedFraud']
X=df_new[features]
y=df_new['isFraud']
```

In [12]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=246)
```

5.0 Base Model(Logistic regression)

In [13]:
```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=5).fit(X_train,y_train)
y_predLogistic=clf.predict(X_test)
pd.Series(y_predLogistic)
print(y_predLogistic)

clf
```

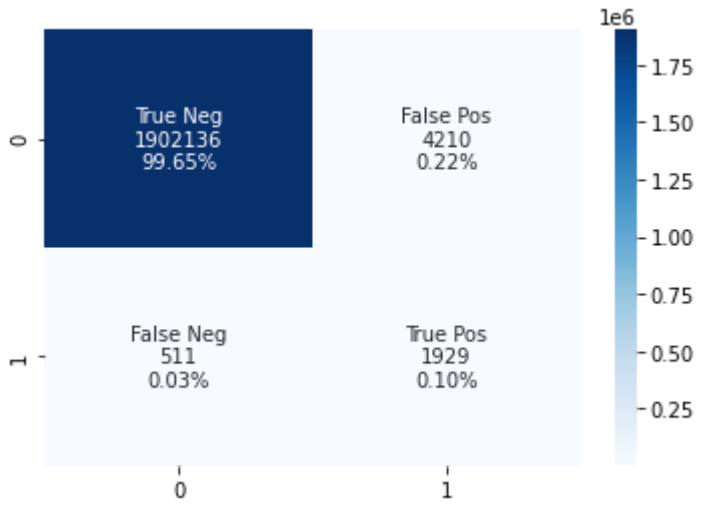
```
[0 0 0 ... 0 0 0]
```

Out[13]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=5, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [21]:
```
from sklearn.metrics import f1_score,classification_report,roc_auc_score
from sklearn.metrics import confusion_matrix
print(f"F1 score of Logistic Regression classifier is {f1_score(clf.predict(X_
test),y_test)}")
print(f"AUC of Logistic Regression classifier is {roc_auc_score(clf.predict(X_
test),y_test)}")
print('Precision score for "Yes"' , metrics.precision_score(y_test,y_predLogis
tic, pos_label =1))
print('Precision score for "No"' , metrics.precision_score(y_test,y_predLogist
ic, pos_label =0))
print('Recall score for "Yes"' , metrics.recall_score(y_test,y_predLogistic, p
os_label =1))
print('Recall score for "No"' , metrics.recall_score(y_test,y_predLogistic, po
s_label =0))
cf_matrix=confusion_matrix(y_test,y_predLogistic)

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

```
F1 score of Logistic Regression classifier is 0.44970276255973896
AUC of Logistic Regression classifier is 0.6569759919452235
Precision score for "Yes" 0.3142205570939893
Precision score for "No" 0.9997314267964578
Recall score for "Yes" 0.7905737704918033
Recall score for "No" 0.9977915866269816
```

Out[21]: <AxesSubplot:>



5.1 DecisionTree(Entropy model- No Depth)

In [22]:
```
entr_model = tree.DecisionTreeClassifier(criterion="entropy", random_state = 4
2)
entr_model.fit(X_train,y_train)

y_pred=entr_model.predict(X_test)
pd.Series(y_pred)
print(y_pred)

entr_model
```

```
[0 0 0 ... 0 0 0]
```

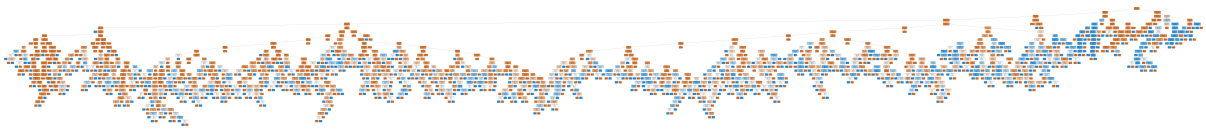
Out[22]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

Now we want to visualize the tree

In [23]:
```
dot_data = StringIO()

tree.export_graphviz(entr_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names=X_train.columns,class_n
ames = ["NO", "YES"])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.664766 to fi
t
```

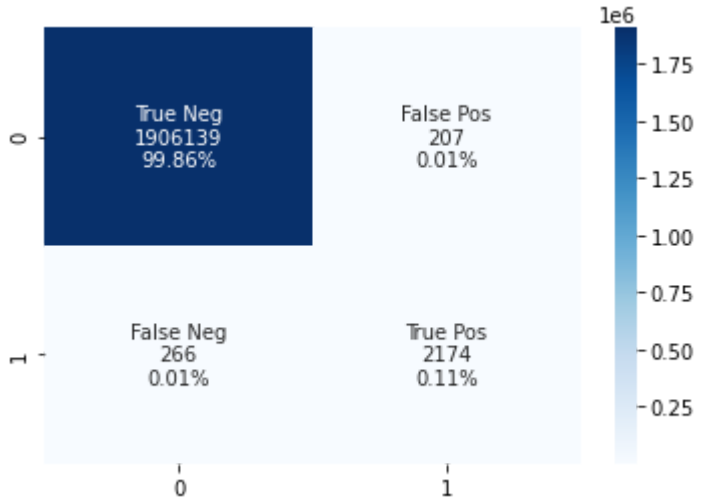
Out[23]:



In [25]:
```
print(f"F1 score of entr_model classifier is {f1_score(entr_model.predict(X_test),y_test)}")
print(f"AUC of entr_model classifier is {roc_auc_score(entr_model.predict(X_test),y_test)}")
print('Precision score for "Yes"' , metrics.precision_score(y_test,y_pred, pos_label =1))
print('Precision score for "No"' , metrics.precision_score(y_test,y_pred, pos_label =0))
print('Recall score for "Yes"' , metrics.recall_score(y_test,y_pred, pos_label =1))
print('Recall score for "No"' , metrics.recall_score(y_test,y_pred, pos_label =0))

cf_matrix=confusion_matrix(y_test,y_pred)

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

```
F1 score of entr_model classifier is 0.9018875751918689
AUC of entr_model classifier is 0.9564611045634894
Precision score for "Yes" 0.9130617387652247
Precision score for "No" 0.9998604703617542
Recall score for "Yes" 0.8909836065573771
Recall score for "No" 0.999891415304462
```

Out[25]: <AxesSubplot:>



This is over fitting model, let's take max depth approach to get to appropriate model.

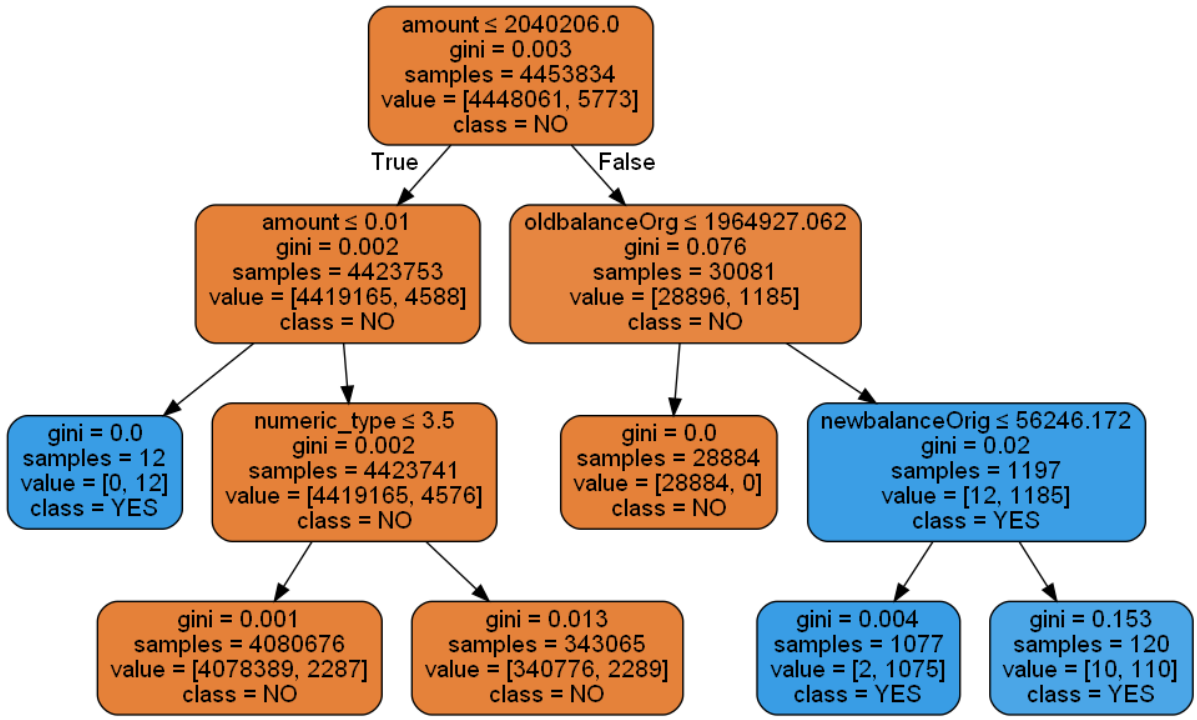
5.2 DecisionTree(Gini impurity model - max depth 3)

In [26]:
```
gini_model = tree.DecisionTreeClassifier(criterion='gini', random_state = 1234
, max_depth = 3)


gini_model.fit(X_train, y_train)
y_pred2 = gini_model.predict(X_test)
y_pred2 = pd.Series(y_pred2)
gini_model
```

Out[26]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=1234, splitter='best')
```

In [27]:
```
dot_data = StringIO()
tree.export_graphviz(gini_model, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names=X_train.columns,class_n
ames = ["NO", "YES"])


graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

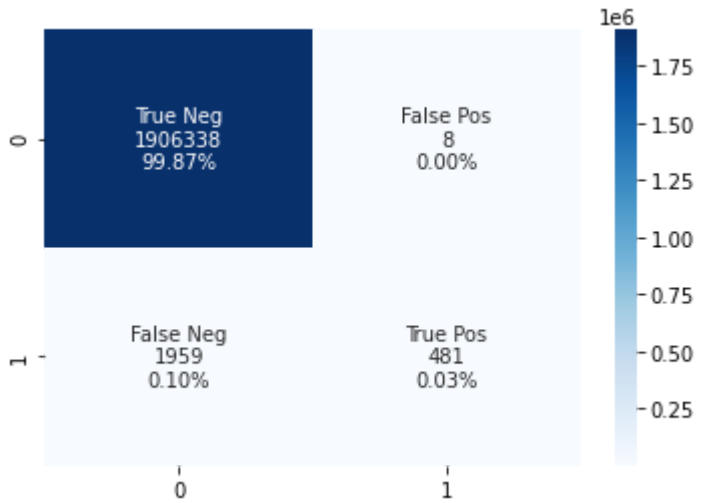
Out[27]:



In [28]:
```
print(f"F1 score of gini_model classifier is {f1_score(gini_model.predict(X_test),y_test)}")
print(f"AUC of gini_model classifier is {roc_auc_score(gini_model.predict(X_test),y_test)}")
print('Precision score for "Yes"' , metrics.precision_score(y_test,y_pred2, pos_label =1))
print('Precision score for "No"' , metrics.precision_score(y_test,y_pred2, pos_label =0))
print('Recall score for "Yes"' , metrics.recall_score(y_test,y_pred2, pos_label =1))
print('Recall score for "No"' , metrics.recall_score(y_test,y_pred2, pos_label =0))
cf_matrix=confusion_matrix(y_test,y_pred2)

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

```
F1 score of gini_model classifier is 0.32843974052577674
AUC of gini_model classifier is 0.9913067560180396
Precision score for "Yes" 0.983640081799591
Precision score for "No" 0.9989734302364883
Recall score for "Yes" 0.1971311475409836
Recall score for "No" 0.9999958034900275
```

Out[28]: <AxesSubplot:>



5.3 DecisionTree(Gini impurity model - no depth)

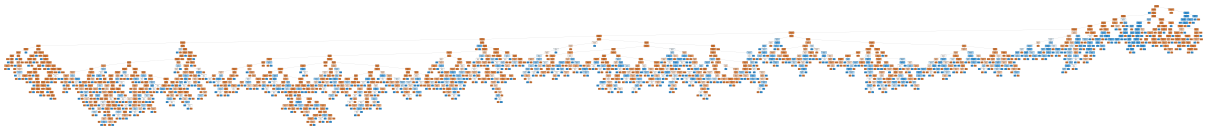
In [29]:
```
gini_model2 = tree.DecisionTreeClassifier(criterion="gini", random_state= 34)
gini_model2.fit(X_train, y_train)

y_pred3 = gini_model2.predict(X_test)
y_pred3 = pd.Series(y_pred3)

gini_model2
```

Out[29]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=Non
e,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=34, splitter='best')
```

In [30]:
```
dot_data = StringIO()
tree.export_graphviz(gini_model2 , out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names=X_train.columns,class_n
ames = ["NO", "YES"])

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

```
dot: graph is too large for cairo-renderer bitmaps. Scaling by 0.566893 to fi
t
```

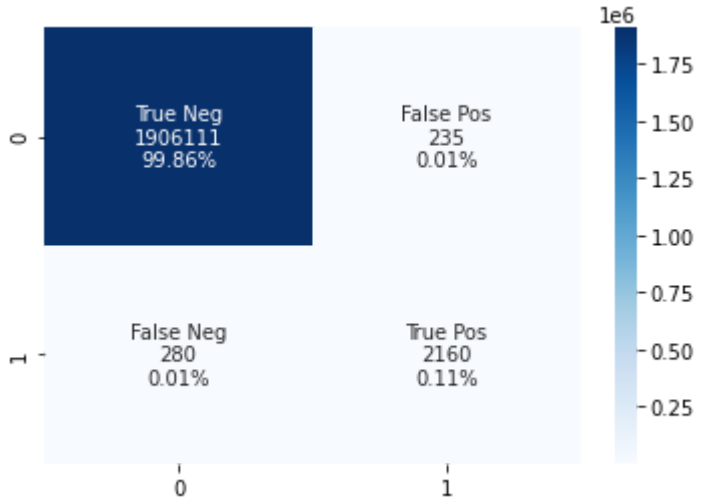
Out[30]:



In [31]:
```
print(f"F1 score of gini_model2 classifier is {f1_score(gini_model2.predict(X_test),y_test)}")
print(f"AUC of gini_model2 classifier is {roc_auc_score(gini_model2.predict(X_test),y_test)}")
print('Precision score for "Yes"' , metrics.precision_score(y_test,y_pred3, pos_label =1))
print('Precision score for "No"' , metrics.precision_score(y_test,y_pred3, pos_label =0))
print('Recall score for "Yes"' , metrics.recall_score(y_test,y_pred3, pos_label =1))
print('Recall score for "No"' , metrics.recall_score(y_test,y_pred3, pos_label =0))
cf_matrix=confusion_matrix(y_test,y_pred3)

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

```
F1 score of gini_model2 classifier is 0.8934850051706309
AUC of gini_model2 classifier is 0.9508660200114988
Precision score for "Yes" 0.9018789144050104
Precision score for "No" 0.9998531256179871
Recall score for "Yes" 0.8852459016393442
Recall score for "No" 0.9998767275195584
```

Out[31]: <AxesSubplot:>



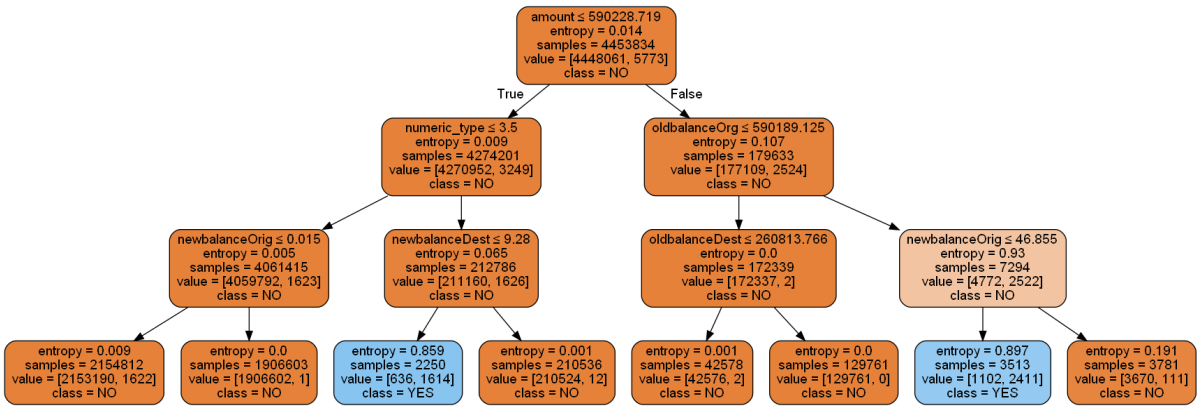
5.4 DecisionTree(Entropy model- max depth 3)

In [32]:
```
entr_model2 = tree.DecisionTreeClassifier(criterion="entropy", max_depth = 3,
random_state = 254)
entr_model2.fit(X_train, y_train)
y_pred4 = entr_model2.predict(X_test)
y_pred4 = pd.Series(y_pred4)
entr_model2
```

Out[32]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=3, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=254, splitter='best')
```

In [33]:
```
import graphviz
dot_data = StringIO()
tree.export_graphviz(entr_model2, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names=X_train.columns,class_n
ames = ["NO", "YES"])


graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

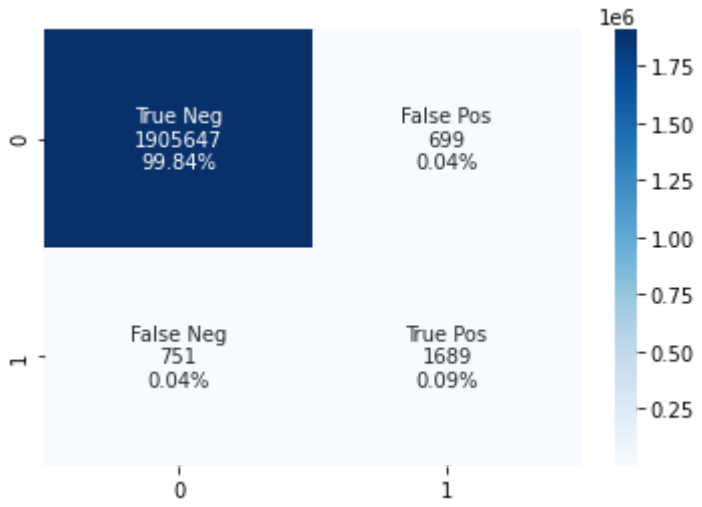
Out[33]:



In [34]:
```
print(f"F1 score of entr_model2 classifier is {f1_score(entr_model2.predict(X_test),y_test)}")
print(f"AUC of entr_model2 classifier is {roc_auc_score(entr_model2.predict(X_test),y_test)}")
print('Precision score for "Yes"' , metrics.precision_score(y_test,y_pred4, pos_label =1))
print('Precision score for "No"' , metrics.precision_score(y_test,y_pred4, pos_label =0))
print('Recall score for "Yes"' , metrics.recall_score(y_test,y_pred4, pos_label =1))
print('Recall score for "No"' , metrics.recall_score(y_test,y_pred4, pos_label =0))

cf_matrix=confusion_matrix(y_test,y_pred4)

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

```
F1 score of entr_model2 classifier is 0.6996685998343
AUC of entr_model2 classifier is 0.8534462477663354
Precision score for "Yes" 0.707286432160804
Precision score for "No" 0.9996060633718667
Recall score for "Yes" 0.6922131147540984
Recall score for "No" 0.9996333299411544
```

Out[34]: <AxesSubplot:>



5.5 RandomForest

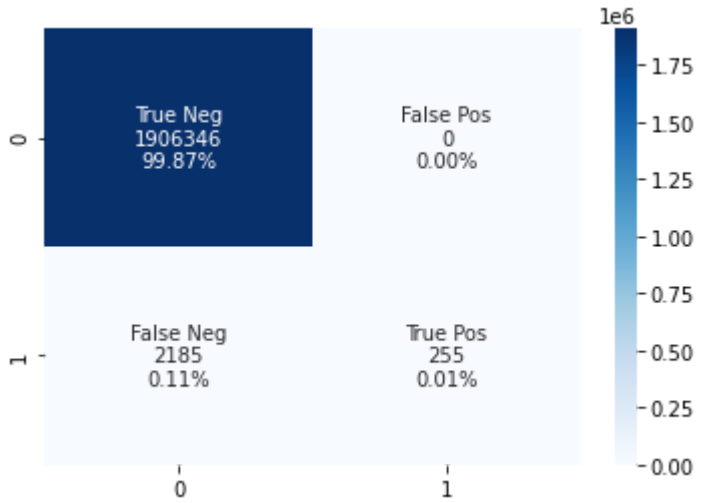
In [35]:
```
RFModel = RandomForestClassifier(max_depth= 3, random_state= 42)
RFModel.fit(X_train, y_train)
y_pred5=RFModel.predict(X_test)
y_pred5 = pd.Series(y_pred5)
RFModel
```

Out[35]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=3, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=42, verbose
=0,
                       warm_start=False)
```

In [36]:
```
print(f"F1 score of RFModel classifier is {f1_score(RFModel.predict(X_test),y_test)}")
print(f"AUC of RFModel classifier is {roc_auc_score(RFModel.predict(X_test),y_test)}")
print('Precision score for "Yes"' , metrics.precision_score(y_test,y_pred5, pos_label =1))
print('Precision score for "No"' , metrics.precision_score(y_test,y_pred5, pos_label =0))
print('Recall score for "Yes"' , metrics.recall_score(y_test,y_pred5, pos_label =1))
print('Recall score for "No"' , metrics.recall_score(y_test,y_pred5, pos_label =0))

cf_matrix=confusion_matrix(y_test,y_pred5)

group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in
                cf_matrix.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in
                     cf_matrix.flatten()/np.sum(cf_matrix)]
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
          zip(group_names,group_counts,group_percentages)]
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues')
```

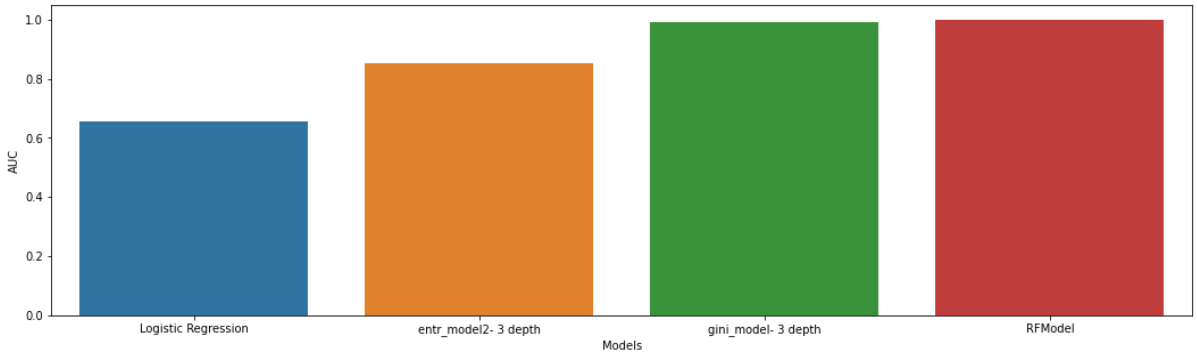
```
F1 score of RFModel classifier is 0.18923933209647495
AUC of RFModel classifier is 0.9994275702097583
Precision score for "Yes" 1.0
Precision score for "No" 0.9988551404195164
Recall score for "Yes" 0.10450819672131148
Recall score for "No" 1.0
```

Out[36]: <AxesSubplot:>



Model Performance

In [37]:
```
f, ax = plt.subplots(figsize=(18,5))
sns.barplot(x=["Logistic Regression","entr_model2- 3 depth","gini_model- 3 dep
th","RFModel"],
            y=[roc_auc_score(clf.predict(X_test),y_test),
               roc_auc_score(entr_model2.predict(X_test),y_test),roc_auc_score
(gini_model.predict(X_test),y_test),roc_auc_score(RFModel.predict(X_test),y_te
st)])
plt.ylabel("AUC")
plt.xlabel("Models")
plt.show()
```



Conclusion: As we can see, Random Forest model is performing the best out of the four models in terms of F1 score, AUC and Confusion matrix. Accuracy cannot be used in this cases becasue of

In [38]:
```
df_probs = pd.DataFrame({"Actual":y_test,"Predicted":RFModel.predict_proba(X_t
est)[:,1],"Amount":X_test.amount})
```

In [41]:
```
print("% Frauds captured by 'Amount > 200,000' strategy in number and amount:"
)
print(len(df_probs[(df_probs.Amount > 200000) & (df_probs.Actual==1)])/len(df_
probs[df_probs.Actual==1]))
print(sum(df_probs[(df_probs.Amount > 200000) & (df_probs.Actual==1)]["Amount"
])/sum(df_probs[df_probs.Actual==1]["Amount"]))
```

```
% Frauds captured by 'Amount > 200,000' strategy in number and amount:
0.6754098360655738
0.981130808606318
```

Proposals to improve the model

1. Hyperparameter tuning is not done on any model. There is still lot of scope of improvement by tuning the models.
2. Selecting the best model based out of precision and recall rather than AUC and F-1 score.