In [1]:
```
#Begin by importing necessary libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


import datetime
import pytz


import datetime as dt
from datetime import timezone

from dateutil.relativedelta import *

import os

from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

%matplotlib inline
```

In [2]:
```
import plotly.graph_objs as go
import plotly.express as px
```

In [3]:
```
#import the dataset into jupyter notebook
fraud=pd.read_csv('C:/Users/WANJIKU/OneDrive/Documents/fraud.csv')
fraud.head()
```

Out[3]:

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud | isFlag |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | PAYMENT | 9839.64 | C1231006815 | 170136.0 | 160296.36 | M1979787155 | 0.0 | 0.0 | 0 | |
| **1** | 1 | PAYMENT | 1864.28 | C1666544295 | 21249.0 | 19384.72 | M2044282225 | 0.0 | 0.0 | 0 | |
| **2** | 1 | TRANSFER | 181.00 | C1305486145 | 181.0 | 0.00 | C553264065 | 0.0 | 0.0 | 1 | |
| **3** | 1 | CASH_OUT | 181.00 | C840083671 | 181.0 | 0.00 | C38997010 | 21182.0 | 0.0 | 1 | |
| **4** | 1 | PAYMENT | 11668.14 | C2048537720 | 41554.0 | 29885.86 | M1230701703 | 0.0 | 0.0 | 0 | |

# EXPLORATORY DATA ANALYSIS

In [4]:
```
fraud.info
```

Out[4]:
```
<bound method DataFrame.info of         step      type      amount     nameOrig  oldbalanceOrg  \
0          1   PAYMENT     9839.64  C1231006815      170136.00
1          1   PAYMENT     1864.28  C1666544295       21249.00
2          1  TRANSFER      181.00  C1305486145         181.00
3          1  CASH_OUT      181.00   C840083671         181.00
4          1   PAYMENT    11668.14  C2048537720       41554.00
...      ...       ...         ...          ...            ...
101608    10   PAYMENT     7477.02   C513257306         785.00
101609    10  CASH_OUT   282252.35   C210473293      220339.29
101610    10   PAYMENT    17289.01   C807582280           0.00
101611    10  TRANSFER  1347146.45  C1315779140        1605.00
101612    10  CASH_OUT   469539.21   C515691325       31133.00

        newbalanceOrig     nameDest  oldbalanceDest  newbalanceDest  isFraud  \
0            160296.36  M1979787155            0.00            0.00        0
1             19384.72  M2044282225            0.00            0.00        0
2                 0.00   C553264065            0.00            0.00        1
3                 0.00    C38997010        21182.00            0.00        1
4             29885.86  M1230701703            0.00            0.00        0
...                ...          ...             ...             ...      ...
101608            0.00   M524833426            0.00            0.00        0
101609            0.00  C1172042998        95156.01        91286.97        0
101610            0.00   M334249577            0.00            0.00        0
101611            0.00  C1631408038       541639.43      2186953.43        0
101612            0.00  C1383702768            0.00       469539.21        0

        isFlaggedFraud
0                    0
1                    0
2                    0
3                    0
4                    0
...                ...
101608               0
101609               0
101610               0
101611               0
101612               0

[101613 rows x 11 columns]>
```

In [5]:
```
fraud.columns
```

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

In [6]:
```
#checking for any missing data
fraud.isnull().sum()
```

Out[6]:
```
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 [7]:
```
fraud.describe()
```

Out[7]:

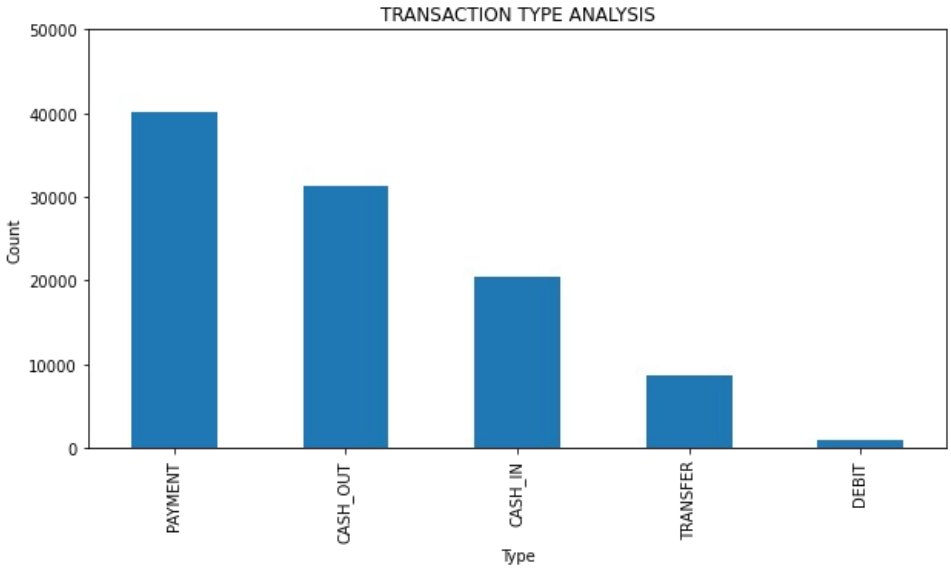
| | step | amount | oldbalanceOrg | newbalanceOrig | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|
| **count** | 101613.000000 | 1.016130e+05 | 1.016130e+05 | 1.016130e+05 | 1.016130e+05 | 1.016130e+05 | 101613.000000 | 101613.0 |
| **mean** | 8.523457 | 1.740901e+05 | 9.071753e+05 | 9.234992e+05 | 8.810428e+05 | 1.183998e+06 | 0.001142 | 0.0 |
| **std** | 1.820681 | 3.450199e+05 | 2.829575e+06 | 2.867319e+06 | 2.399949e+06 | 2.797761e+06 | 0.033768 | 0.0 |
| **min** | 1.000000 | 3.200000e-01 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000 | 0.0 |
| **25%** | 8.000000 | 1.001659e+04 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000e+00 | 0.000000 | 0.0 |
| **50%** | 9.000000 | 5.338541e+04 | 2.019047e+04 | 0.000000e+00 | 2.105800e+04 | 5.178343e+04 | 0.000000 | 0.0 |
| **75%** | 10.000000 | 2.124984e+05 | 1.947150e+05 | 2.192178e+05 | 5.919217e+05 | 1.063122e+06 | 0.000000 | 0.0 |
| **max** | 10.000000 | 1.000000e+07 | 3.893942e+07 | 3.894623e+07 | 3.400874e+07 | 3.894623e+07 | 1.000000 | 0.0 |

In [8]:
```
fraud['type'].value_counts()
```

Out[8]:
```
PAYMENT     40062
CASH_OUT    31310
CASH_IN     20540
TRANSFER     8689
DEBIT        1012
Name: type, dtype: int64
```

In [9]:
```
# to visualize how the diffrent types of transactions are spread through the dataset

plt.figure(figsize=(10,5))
fraud['type'].value_counts().plot(kind='bar')

plt.title('TRANSACTION TYPE ANALYSIS')
plt.xlabel('Type')
plt.ylabel('Count')

plt.ylim(0,50000)
```

Out[9]:
```
(0.0, 50000.0)
```



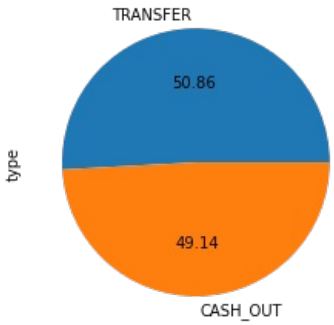

In [10]:
```
df=fraud.loc[fraud['isFraud']==1]
df.head()

#here we lock only the rows that have our target variable as 1 and compare how transaction type is distributed
```

Out[10]:

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud | isFlag |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **2** | 1 | TRANSFER | 181.0 | C1305486145 | 181.0 | 0.0 | C553264065 | 0.0 | 0.0 | 1 | |
| **3** | 1 | CASH_OUT | 181.0 | C840083671 | 181.0 | 0.0 | C38997010 | 21182.0 | 0.0 | 1 | |
| **251** | 1 | TRANSFER | 2806.0 | C1420196421 | 2806.0 | 0.0 | C972765878 | 0.0 | 0.0 | 1 | |
| **252** | 1 | CASH_OUT | 2806.0 | C2101527076 | 2806.0 | 0.0 | C1007251739 | 26202.0 | 0.0 | 1 | |
| **680** | 1 | TRANSFER | 20128.0 | C137533655 | 20128.0 | 0.0 | C1848415041 | 0.0 | 0.0 | 1 | |

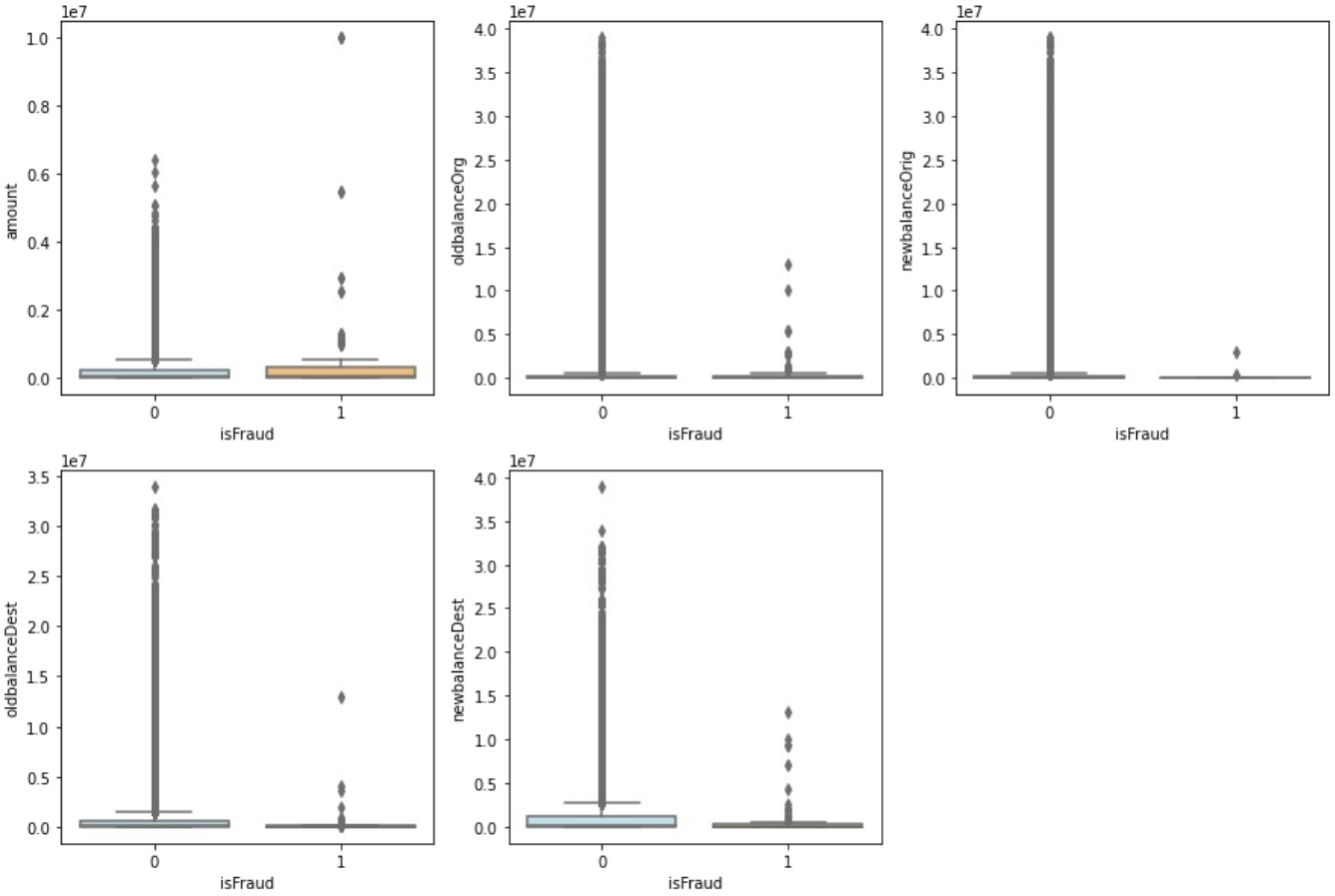
In [11]:
```
df['type'].value_counts().plot(kind='pie', autopct='%.2f', labels=df['type'])
```

Out[11]:
```
<AxesSubplot:ylabel='type'>
```



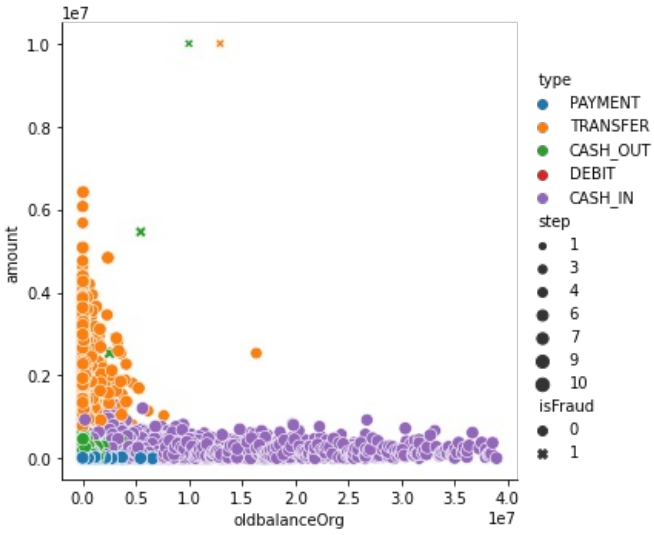

In [12]:
```
num_list = ['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest', 'newbalanceDest']

fig = plt.figure(figsize=(15,10))

for i in range(len(num_list)):
    column=num_list[i]
    sub=fig.add_subplot(2,3, i+1)
    sns.boxplot(x='isFraud', y=column, data=fraud, palette='RdYlBu_r')
```

```
#the boxplots show how the value of numerical features vary across the target group.
# for example the balances both old and new and on the originator and destination have distint difference when
# and when target is 1 suggesting that they are important predictors
#amount appears to be less outstanding as the boxplot distribution is similar between target groups
```



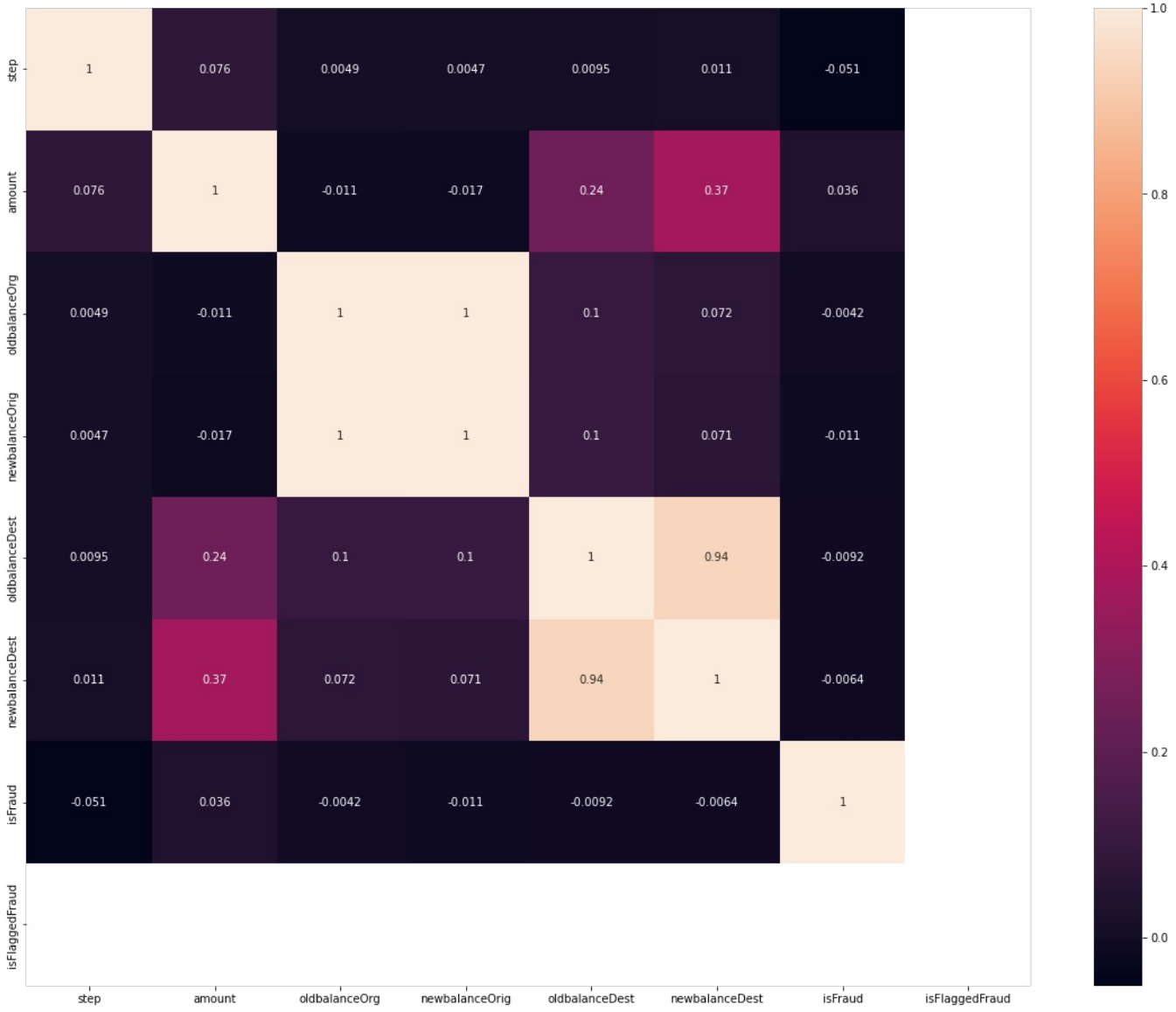

In [13]:
```
sns.relplot(x="oldbalanceOrg", y="amount", data=fraud, kind="scatter",
            hue="type", size="step",
            style='isFraud',
            )
```

Out[13]:
```
<seaborn.axisgrid.FacetGrid at 0x2a3ed533ac0>
```




In [14]:
```
#checking correlation among the features

plt.figure(figsize=(20,16))
sns.heatmap(fraud.corr(), fmt='.2g', annot=True)
```

Out[14]:
```
<AxesSubplot:>
```




In [15]:
```
#check the number of unique values on object datatypes

fraud.select_dtypes(include='object').nunique()
```

Out[15]:
```
type            5
nameOrig    101613
nameDest     52280
dtype: int64
```

In [16]:
```
fraud[['type', 'nameOrig', 'nameDest']] = fraud[['type', 'nameOrig', 'nameDest']].astype(str)
```

In [17]:
```
from sklearn import preprocessing
```

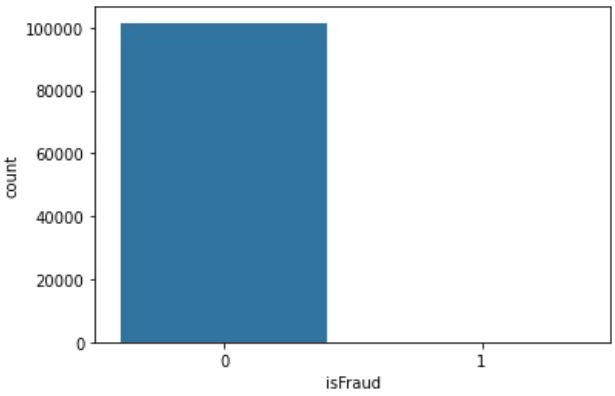
In [18]:
```
for col in fraud.select_dtypes(include=['object']).columns:
    label_encoder=preprocessing.LabelEncoder()
    label_encoder.fit(fraud[col].unique())
    fraud[col]=label_encoder.transform(fraud[col])
    print(f"{col}:{fraud[col].unique()}")
```

```
type:[3 4 1 2 0]
nameOrig:[12031 34878 15860 ... 91466 16397 75956]
nameDest:[32431 33746  9437 ... 24586 42340 38337]
```

In [19]:
```
# to check if our label isFraud is balance

sns.countplot(fraud['isFraud'])

fraud['isFraud'].value_counts()
```

```
C:\Users\WANJIKU\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[19]:
```
0    101497
1       116
Name: isFraud, dtype: int64
```

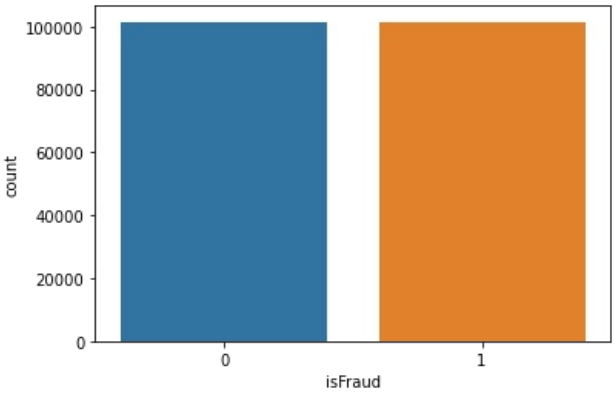



In [20]:
```
#oversampling the minority class to balance the label

from sklearn.utils import resample

fraud_majority=fraud[(fraud['isFraud']==0)]
fraud_minority=fraud[(fraud['isFraud']==1)]

fraud_minority_upsampled=resample(fraud_minority,
                                  replace=True,
                                  n_samples=101497,
                                  random_state=0)

fraud_upsampled=pd.concat([fraud_minority_upsampled, fraud_majority])
```

In [21]:
```
sns.countplot(fraud_upsampled['isFraud'])
fraud_upsampled['isFraud'].value_counts()
```

```
C:\Users\WANJIKU\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[21]:
```
1    101497
0    101497
Name: isFraud, dtype: int64
```




In [22]:
```
fraud_upsampled.shape
```

Out[22]:
```
(202994, 11)
```

In [23]:
```
fraud_upsampled.head()
```

Out[23]:

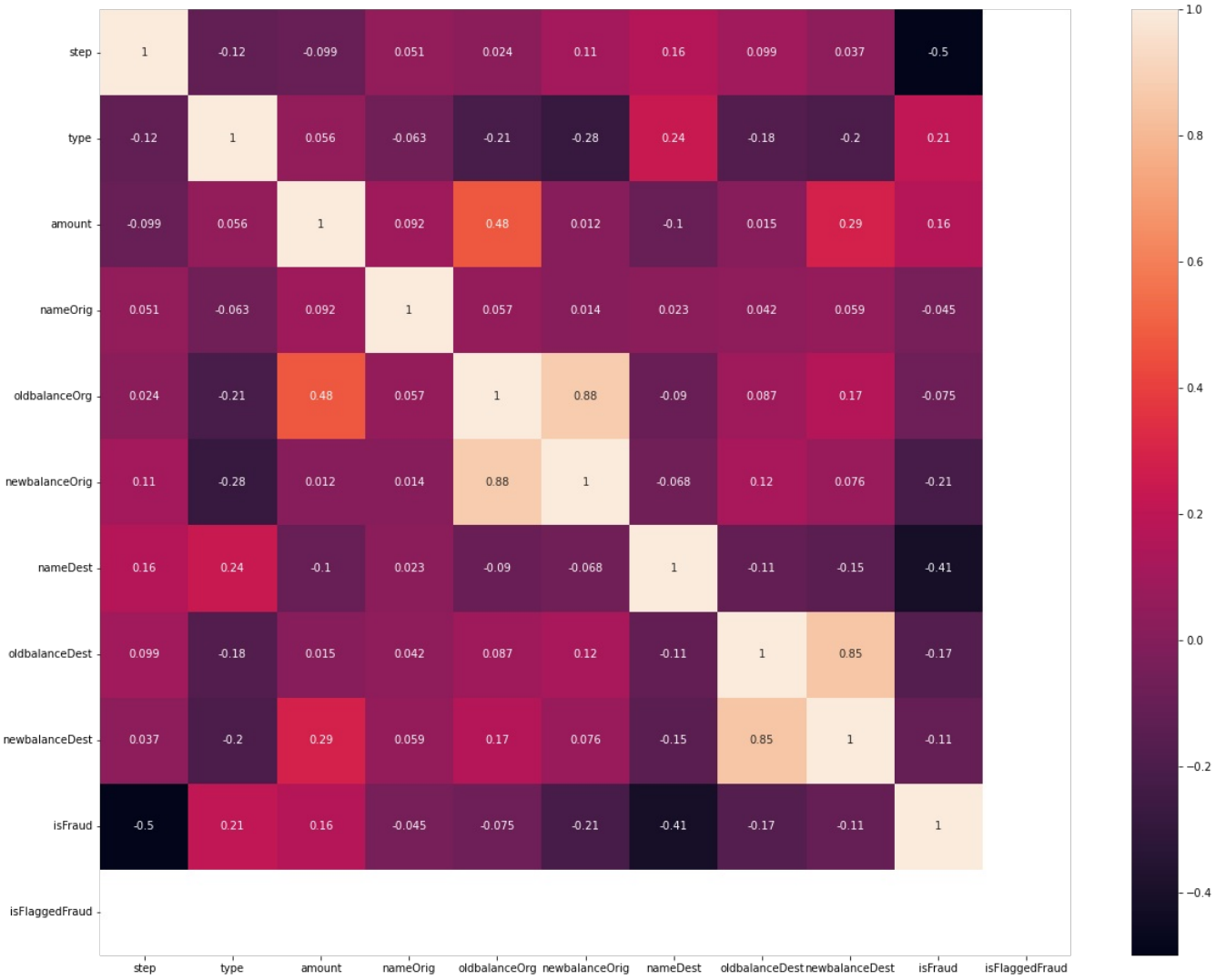
| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud | isFlaggedFraud |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **5558** | 6 | 4 | 33332.86 | 53661 | 33332.86 | 0.0 | 4169 | 0.00 | 0.00 | 1 | 0 |
| **5747** | 6 | 1 | 25975.86 | 240 | 25975.86 | 0.0 | 8804 | 98152.00 | 28041.27 | 1 | 0 |
| **7154** | 6 | 4 | 13704.00 | 38516 | 13704.00 | 0.0 | 1790 | 0.00 | 1658746.09 | 1 | 0 |
| **9285** | 7 | 1 | 262434.54 | 69051 | 262434.54 | 0.0 | 8324 | 19525.79 | 438233.86 | 1 | 0 |
| **9285** | 7 | 1 | 262434.54 | 69051 | 262434.54 | 0.0 | 8324 | 19525.79 | 438233.86 | 1 | 0 |

In [24]:
```
#check correlation in the upsampled df

plt.figure(figsize=(20,16))
```

```
sns.heatmap(fraud_upsampled.corr(), fmt='.2g', annot=True)
```

Out[24]:

```
<AxesSubplot:>
```




In [25]:

```
fraud_upsampled.drop(columns='isFlaggedFraud', inplace=True)
```

In [26]:

```
#train test split

X = fraud_upsampled.drop('isFraud', axis=1)
y = fraud_upsampled['isFraud']
```

# DECISION TREE

In [27]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

In [28]:

```
#train and test

X_train, X_test, y_train, y_test = train_test_split (X,y, test_size=0.2, random_state=0)
```

In [29]:

```
#perform param grid to get the best hyper parameters for the model

dtree = DecisionTreeClassifier()

param_grid = { 'max_depth': [ 2,4,6,8],
               'min_samples_split': [2,4,6,8],
               'min_samples_leaf': [1,2,3,4],
               'max_features': ['auto', 'sqrt', 'log2'],
               'random_state': [0,7,42]}

grid_search = GridSearchCV (dtree, param_grid, cv=5)

grid_search.fit(X_train, y_train)

print (grid_search.best_params_)
```

```
{'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state': 0}
```

In [30]:

```
#fit the model
```

```
dtree = DecisionTreeClassifier (max_depth=8, min_samples_leaf=1, min_samples_split=2, max_features='auto', rand

dtree.fit(X_train, y_train)
```

Out[30]:
```
DecisionTreeClassifier(max_depth=8, max_features='auto', random_state=0)
```

In [31]:
```
y_pred = dtree.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

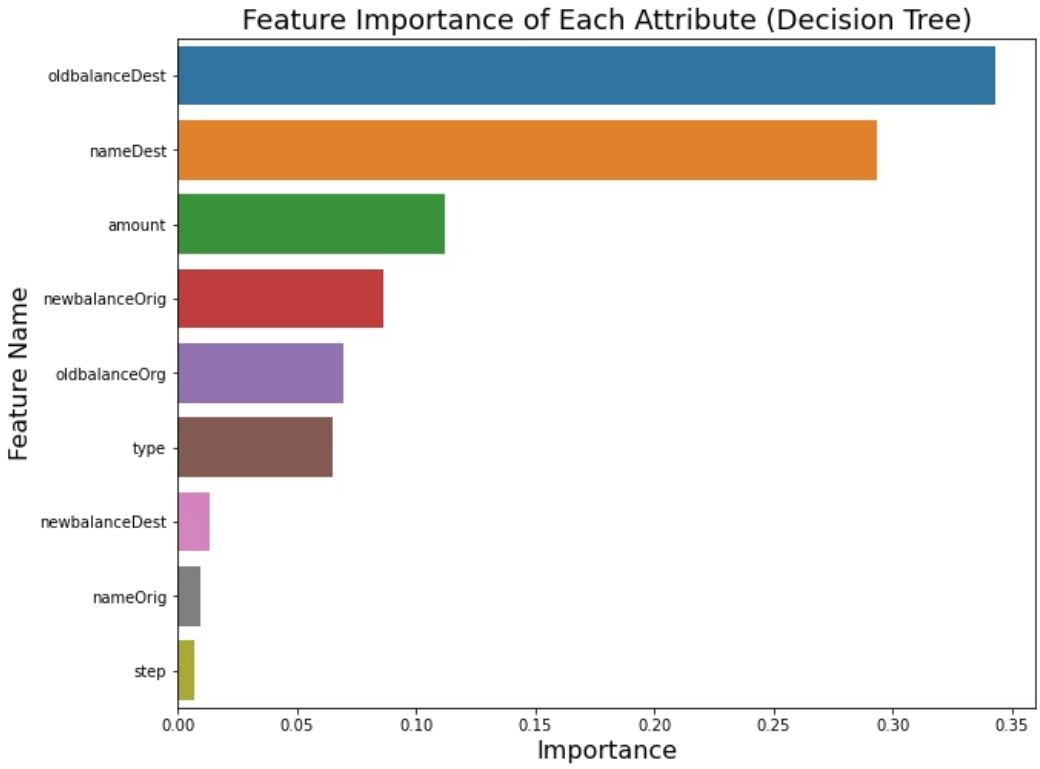
```
Accuracy Score: 98.27 %
```

In [32]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
```

In [33]:
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

```
F-1 Score: 0.9827089337175793
Precision Score: 0.9827089337175793
Recall Score: 0.9827089337175793
Jaccard Score: 0.9660056657223796
Log Loss: 0.5972260978502754
```

In [34]:
```
imp_df = pd.DataFrame({ "Feature Name": X_train.columns,
                        "Importance": dtree.feature_importances_})

fi= imp_df.sort_values(by = "Importance", ascending=False)

plt.figure(figsize=(10,8))
sns.barplot(data=fi, x= 'Importance', y= 'Feature Name')
plt.title('Feature Importance of Each Attribute (Decision Tree)', fontsize=18)
plt.xlabel( 'Importance', fontsize=16)
plt.ylabel( 'Feature Name', fontsize=16)
plt.show()
```

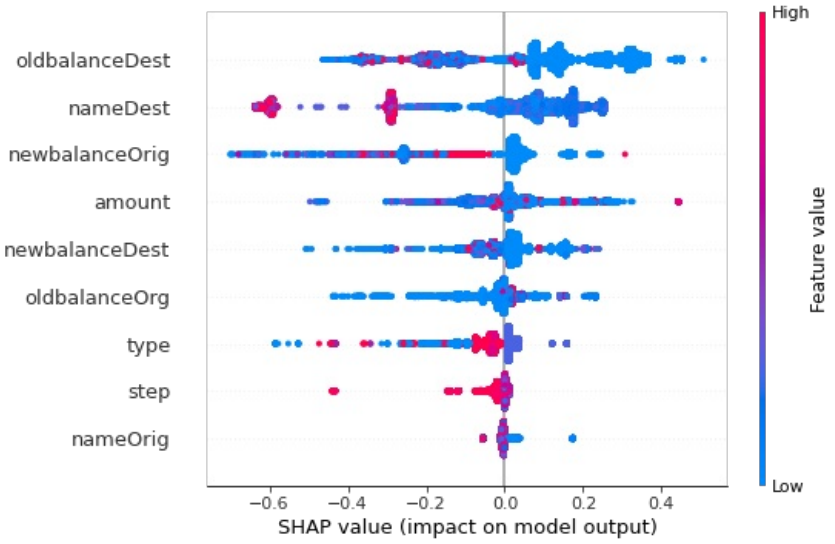



In [35]:
```
!pip install shap
```

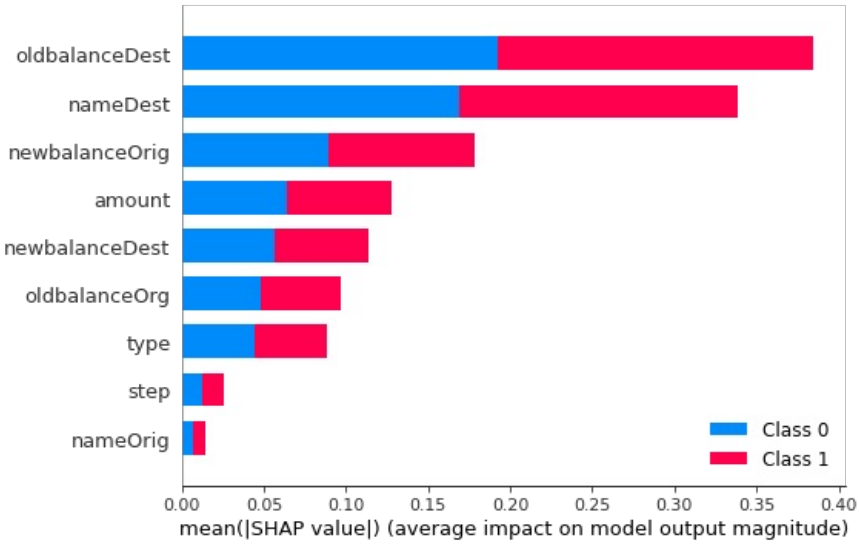
```
Requirement already satisfied: shap in c:\users\wanjiku\anaconda3\lib\site-packages (0.41.0)
Requirement already satisfied: numba in c:\users\wanjiku\anaconda3\lib\site-packages (from shap) (0.55.1)
Requirement already satisfied: scipy in c:\users\wanjiku\anaconda3\lib\site-packages (from shap) (1.7.3)
Requirement already satisfied: packaging>20.9 in c:\users\wanjiku\anaconda3\lib\site-packages (from shap) (21.3
)
Requirement already satisfied: tqdm>4.25.0 in c:\users\wanjiku\anaconda3\lib\site-packages (from shap) (4.64.0)
Requirement already satisfied: cloudpickle in c:\users\wanjiku\anaconda3\lib\site-packages (from shap) (2.0.0)
Requirement already satisfied: scikit-learn in c:\users\wanjiku\anaconda3\lib\site-packages (from shap) (1.0.2)
Requirement already satisfied: pandas in c:\users\wanjiku\anaconda3\lib\site-packages (from shap) (1.4.2)
Requirement already satisfied: slicer==0.0.7 in c:\users\wanjiku\anaconda3\lib\site-packages (from shap) (0.0.7
)
Requirement already satisfied: numpy in c:\users\wanjiku\anaconda3\lib\site-packages (from shap) (1.21.5)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\wanjiku\anaconda3\lib\site-packages (from p
ackaging>20.9->shap) (3.0.4)
Requirement already satisfied: colorama in c:\users\wanjiku\anaconda3\lib\site-packages (from tqdm>4.25.0->shap
) (0.4.4)
Requirement already satisfied: llvmlite<0.39,>=0.38.0rc1 in c:\users\wanjiku\anaconda3\lib\site-packages (from
numba->shap) (0.38.0)
Requirement already satisfied: setuptools in c:\users\wanjiku\anaconda3\lib\site-packages (from numba->shap) (6
1.2.0)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\wanjiku\anaconda3\lib\site-packages (from pan
das->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\wanjiku\anaconda3\lib\site-packages (from pandas->shap)
(2021.3)
Requirement already satisfied: six>=1.5 in c:\users\wanjiku\anaconda3\lib\site-packages (from python-dateutil>=
2.8.1->pandas->shap) (1.16.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\wanjiku\anaconda3\lib\site-packages (from sciki
t-learn->shap) (2.2.0)
Requirement already satisfied: joblib>=0.11 in c:\users\wanjiku\anaconda3\lib\site-packages (from scikit-learn-
>shap) (1.1.0)
```

In [36]:
```
import shap
```

In [37]:
```
explainer=shap.TreeExplainer(dtree)

shap_values=explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names=X_test.columns)
```

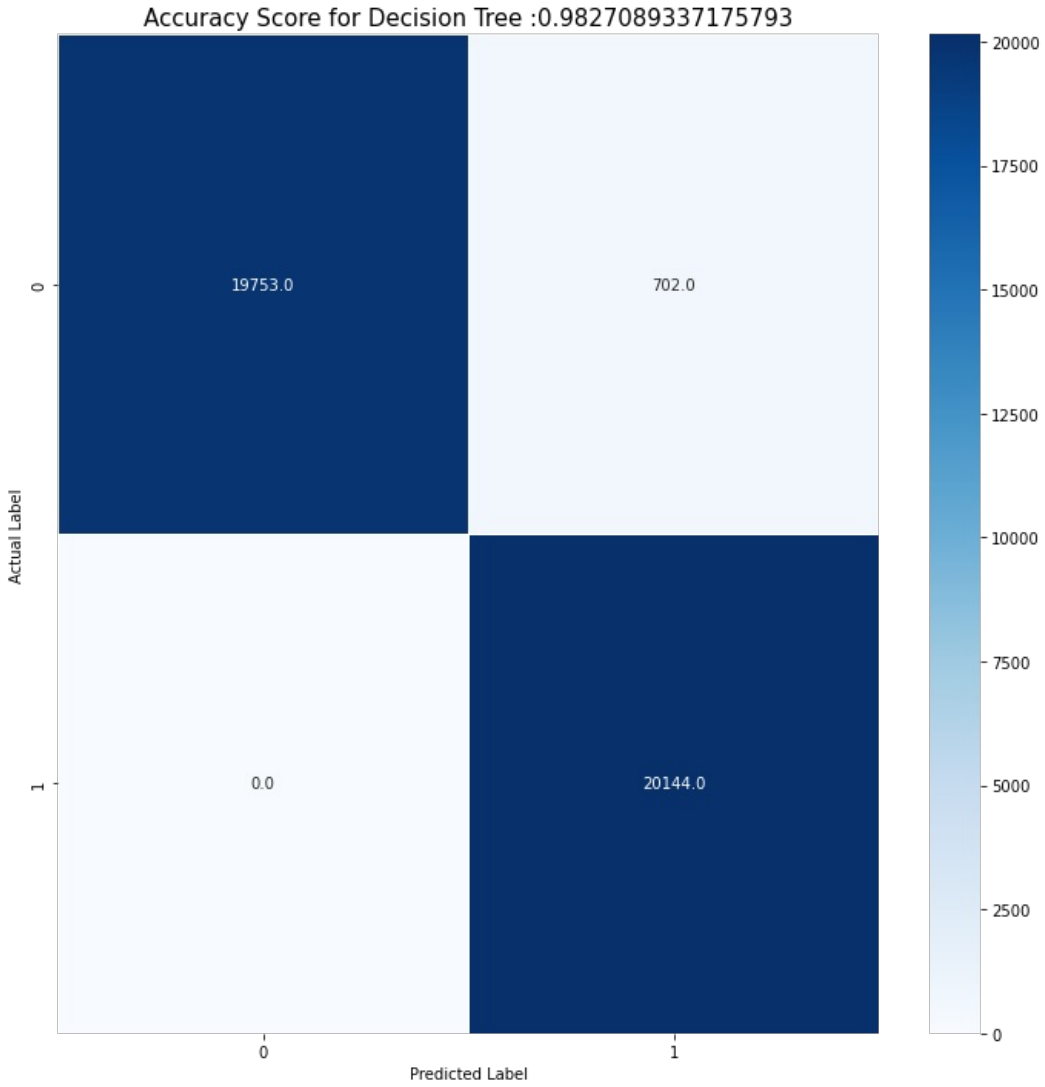



In [38]:
```
shap.summary_plot(shap_values, X_test)
```




In [39]:
```
from sklearn.metrics import confusion_matrix
```

In [40]:
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Blues')




plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for Decision Tree :{0}'.format (dtree.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```




In [41]:
```
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn import metrics
```

In [42]:
```
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]
fraud_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
fraud_actual_predicted.index=y_test.index
fraud_actual_predicted
```

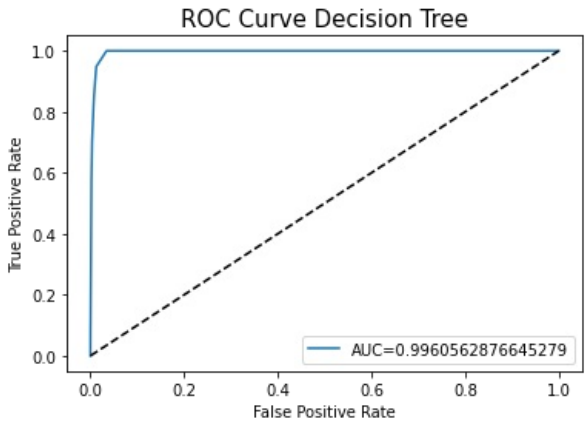
Out[42]:

| | y_actual |
|---|---|
| **35823** | 0 |
| **58307** | 1 |
| **49135** | 0 |
| **25084** | 0 |
| **9535** | 0 |
| **...** | ... |
| **3684** | 1 |
| **91476** | 0 |
| **41075** | 0 |
| **969** | 1 |
| **101577** | 0 |

40599 rows × 1 columns

In [43]:
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)

auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Decision Tree', size=15)
plt.legend(loc=4)
```

Out[43]:
```
<matplotlib.legend.Legend at 0x2a3f9d2c310>
```




In [ ]:
```
```

# RANDOM FOREST

In [44]:
```
fraud_upsampled.head()
```

Out[44]:

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud |
|---|---|---|---|---|---|---|---|---|---|---|
| **5558** | 6 | 4 | 33332.86 | 53661 | 33332.86 | 0.0 | 4169 | 0.00 | 0.00 | 1 |
| **5747** | 6 | 1 | 25975.86 | 240 | 25975.86 | 0.0 | 8804 | 98152.00 | 28041.27 | 1 |
| **7154** | 6 | 4 | 13704.00 | 38516 | 13704.00 | 0.0 | 1790 | 0.00 | 1658746.09 | 1 |
| **9285** | 7 | 1 | 262434.54 | 69051 | 262434.54 | 0.0 | 8324 | 19525.79 | 438233.86 | 1 |
| **9285** | 7 | 1 | 262434.54 | 69051 | 262434.54 | 0.0 | 8324 | 19525.79 | 438233.86 | 1 |

In [45]:
```
#train test split

X = fraud_upsampled.drop('isFraud', axis=1)
y = fraud_upsampled['isFraud']
```

In [46]:
```
X_train, X_test, y_train, y_test = train_test_split (X,y, test_size=0.2, random_state=0)
```

In [47]:
```
from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()

rfc = RandomForestClassifier (max_depth=9, min_samples_leaf=10, min_samples_split=2, max_features='sqrt', rando
```

```
rfc.fit(X_train, y_train)
```

Out[47]:
```
RandomForestClassifier(max_depth=9, max_features='sqrt', min_samples_leaf=10,
                       random_state=0)
```

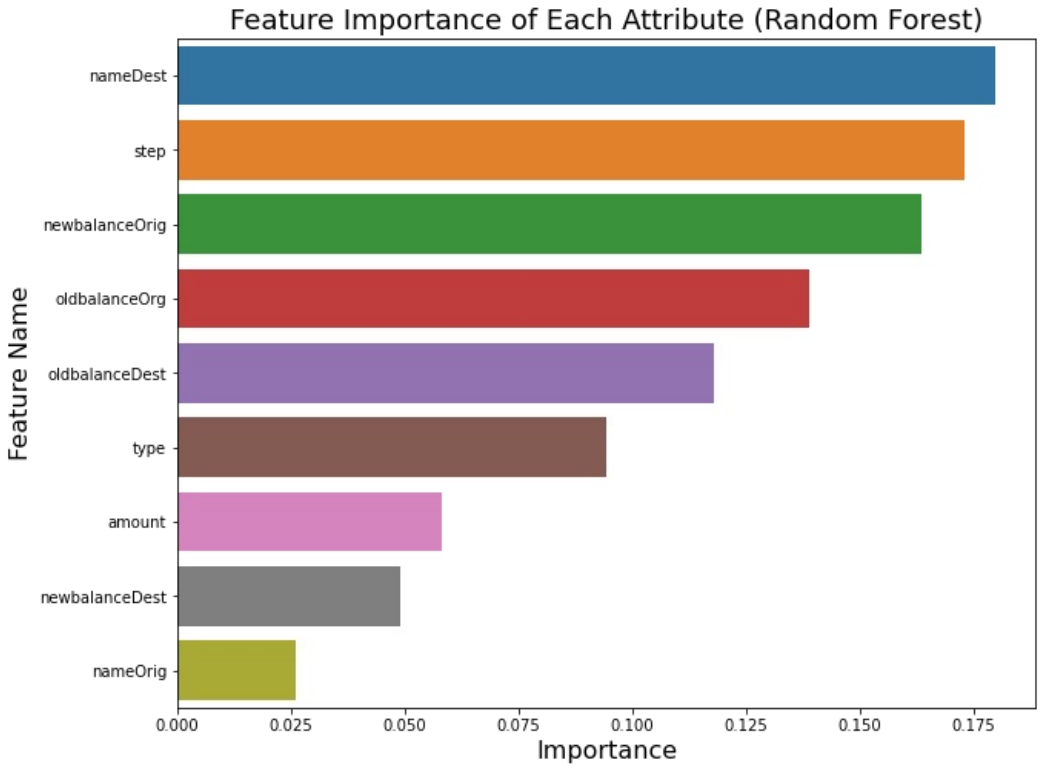
In [48]:
```
y_pred = rfc.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

```
Accuracy Score: 99.81 %
```

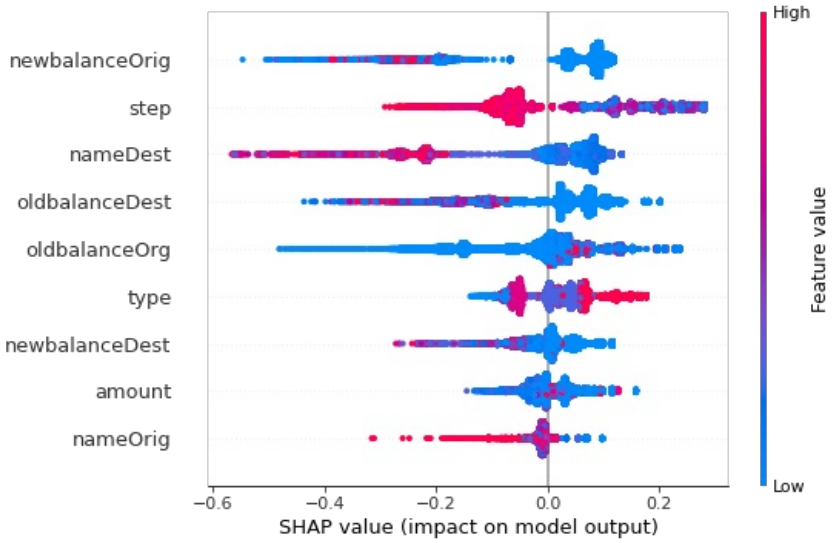
In [49]:
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

```
F-1 Score: 0.9980541392645139
Precision Score: 0.9980541392645139
Recall Score: 0.9980541392645139
Jaccard Score: 0.9961158365701361
Log Loss: 0.06720920474383531
```

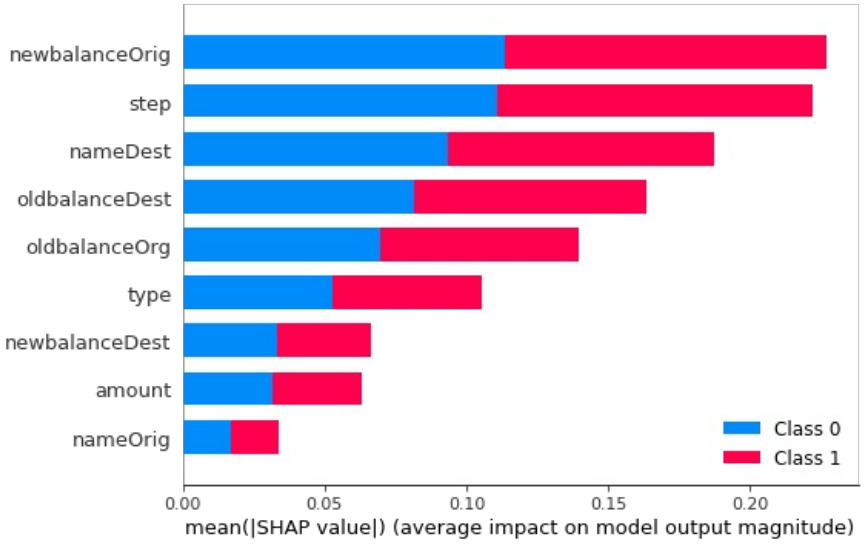
In [50]:
```
imp_df = pd.DataFrame({ "Feature Name": X_train.columns,
                        "Importance": rfc.feature_importances_})

fi= imp_df.sort_values(by = "Importance", ascending=False)

plt.figure(figsize=(10,8))
sns.barplot(data=fi, x= 'Importance', y= 'Feature Name')
plt.title('Feature Importance of Each Attribute (Random Forest)', fontsize=18)
plt.xlabel( 'Importance', fontsize=16)
plt.ylabel( 'Feature Name', fontsize=16)
plt.show()
```



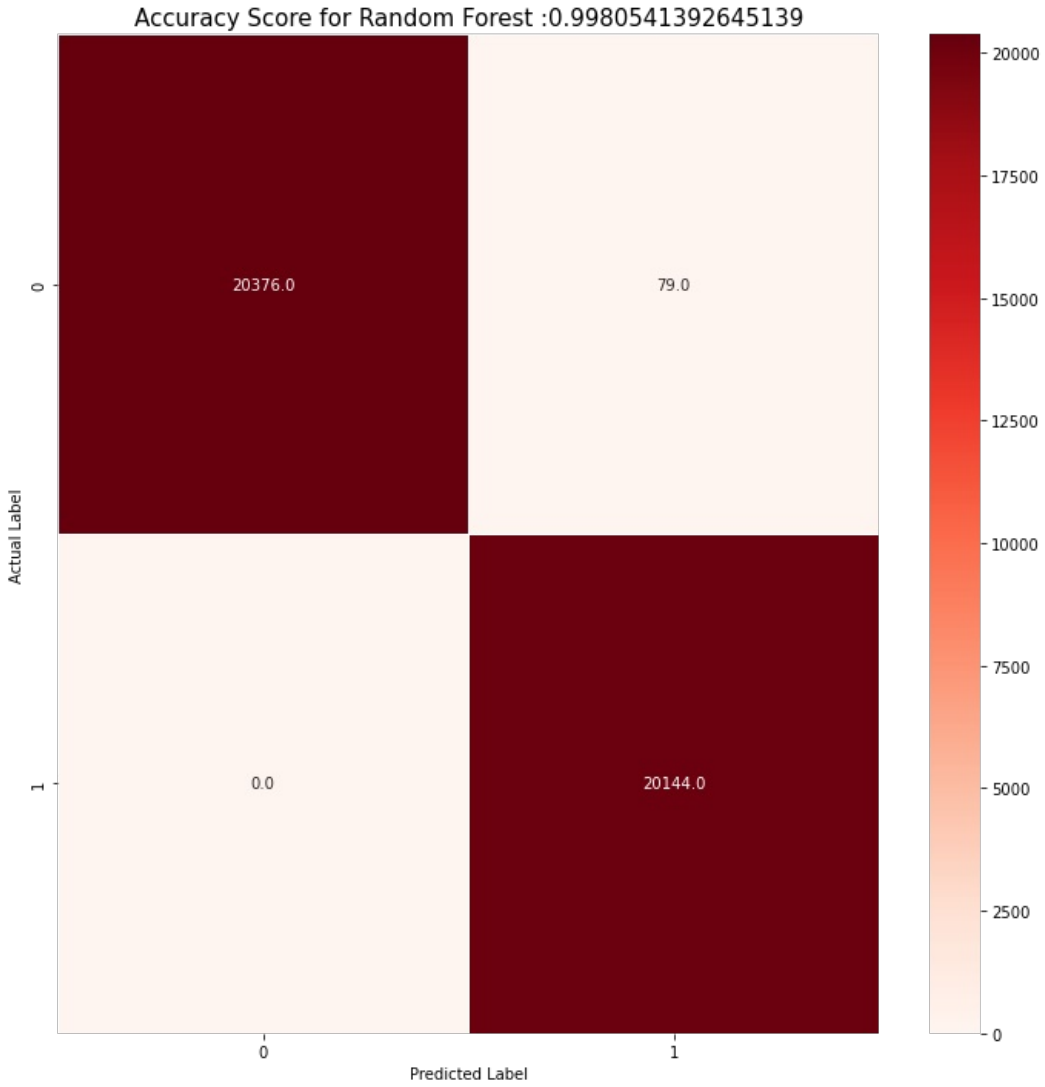

In [51]:
```
explainer=shap.TreeExplainer(rfc)

shap_values=explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names=X_test.columns)
```




In [52]:
```
shap.summary_plot(shap_values, X_test)
```




In [53]:
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Reds')




plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for Random Forest :{0}'.format (rfc.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```




In [54]:
```
y_pred_proba = rfc.predict_proba(X_test)[:][:,1]
fraud_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
fraud_actual_predicted.index=y_test.index
fraud_actual_predicted
```

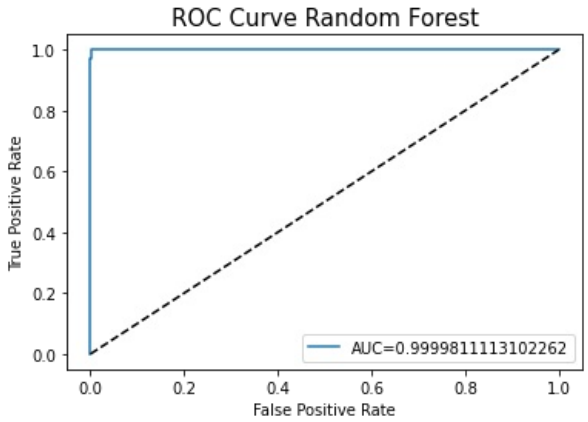
Out[54]:

| | y_actual |
|---|---|
| **35823** | 0 |
| **58307** | 1 |
| **49135** | 0 |
| **25084** | 0 |
| **9535** | 0 |
| **...** | ... |
| **3684** | 1 |
| **91476** | 0 |
| **41075** | 0 |
| **969** | 1 |
| **101577** | 0 |

40599 rows × 1 columns

In [55]:
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)

auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Random Forest', size=15)
plt.legend(loc=4)
```

Out[55]:
```
<matplotlib.legend.Legend at 0x2a3f7902a60>
```




# SUPPORT VECTOR MACHINE

In [56]:
```
fraud_upsampled.head()
```

Out[56]:

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | oldbalanceDest | newbalanceDest | isFraud |
|---|---|---|---|---|---|---|---|---|---|---|
| **5558** | 6 | 4 | 33332.86 | 53661 | 33332.86 | 0.0 | 4169 | 0.00 | 0.00 | 1 |
| **5747** | 6 | 1 | 25975.86 | 240 | 25975.86 | 0.0 | 8804 | 98152.00 | 28041.27 | 1 |
| **7154** | 6 | 4 | 13704.00 | 38516 | 13704.00 | 0.0 | 1790 | 0.00 | 1658746.09 | 1 |
| **9285** | 7 | 1 | 262434.54 | 69051 | 262434.54 | 0.0 | 8324 | 19525.79 | 438233.86 | 1 |
| **9285** | 7 | 1 | 262434.54 | 69051 | 262434.54 | 0.0 | 8324 | 19525.79 | 438233.86 | 1 |

In [57]:
```
from sklearn.svm import SVC
```

In [58]:
```
X = fraud_upsampled.drop('isFraud', axis=1)
y = fraud_upsampled['isFraud']
```

In [59]:
```
X_train, X_test, y_train, y_test = train_test_split (X,y, test_size=0.2, random_state=0)
```

In [60]:
```
svc= SVC()
svc.fit(X_train, y_train)
```

Out[60]:
```
SVC()
```

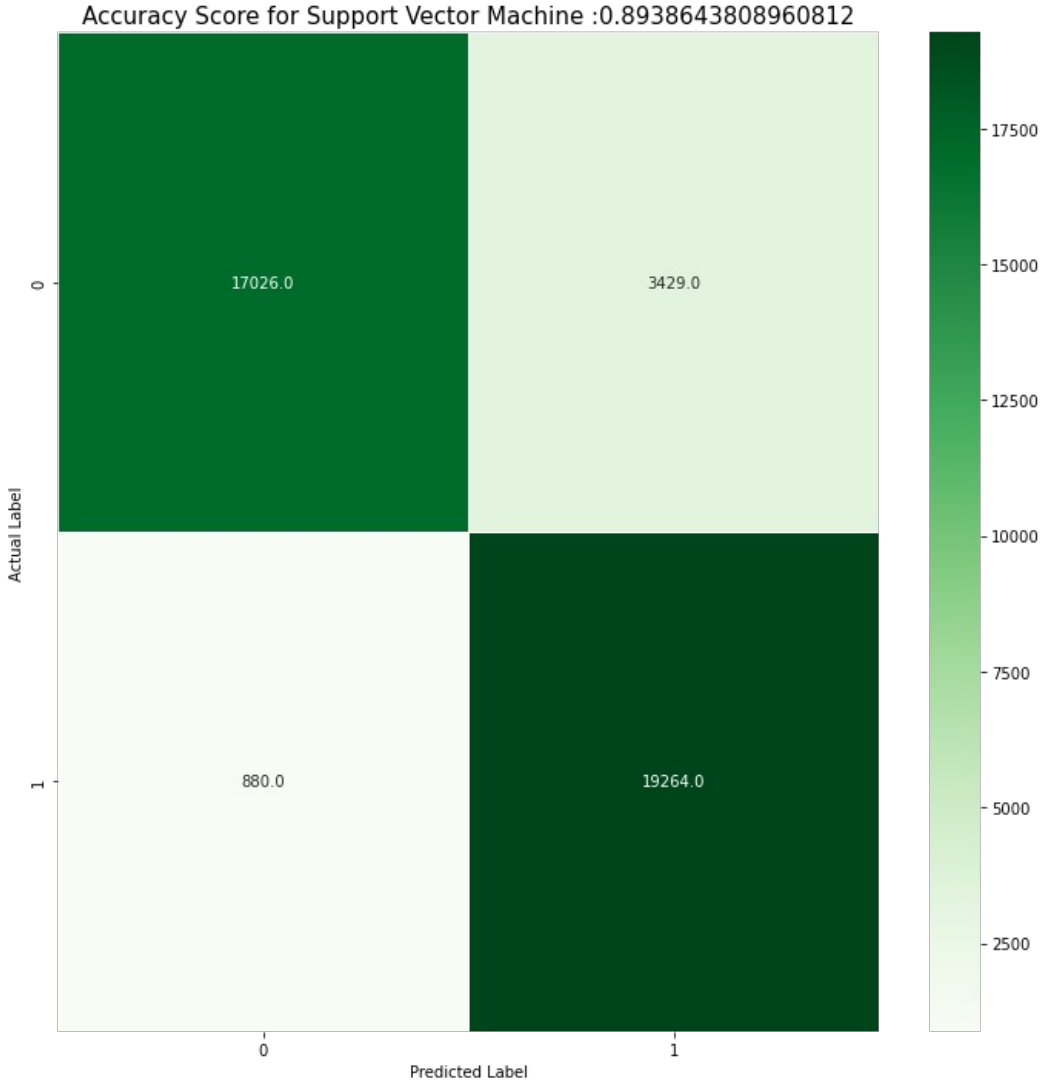
In [61]:
```
y_pred = svc.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

```
Accuracy Score: 89.39 %
```

In [62]:
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

```
F-1 Score: 0.8938643808960812
Precision Score: 0.8938643808960812
Recall Score: 0.8938643808960812
Jaccard Score: 0.8080965529527033
Log Loss: 3.665861949931237
```

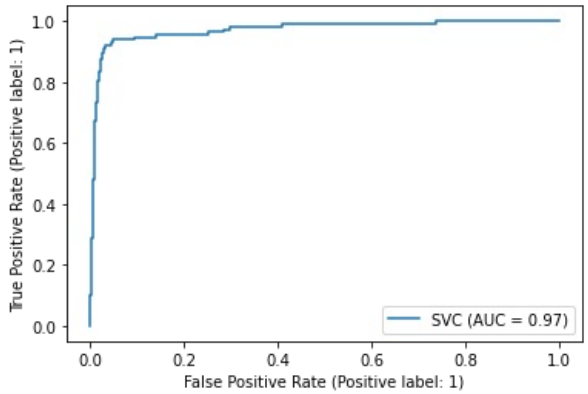
In [63]:
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Greens')




plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for Support Vector Machine :{0}'.format (svc.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```




In [64]:
```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(svc, X_test, y_test)
```

```
Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
```

Out[64]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2a3f5b99610>
```




In [ ]:

In [ ]:

In [ ]:

In [ ]: