PROJECT TITLE: ONLINE PAYMENT FRAUD DETECTION

PROJECT DEFINITION: Fraud detection is defined as a process that detects scams and prevents fraudsters from obtaining money or property through false means. Fraud is a serious business risk that needs to be identified and mitigated in time. The bank in this case study is called BLOSSOM BANK which is a multinational financial services group that offers retail and investment banking, pension management, asset management, and payment services whose headquarters is in London.

PROBLEM STATEMENT: The aim of this project is to predict online payment fraud in Blossom Bank.

In [1]:
```
# import the necessary libraries

# For Data Analysis
import pandas as pd
import numpy as np

# Data visualization
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Load the data set - ONLNE PAYMENT FRAUD DETECTION.CSV
Fraud_D = pd.read_csv(r'C:\Users\MOGTECH\Desktop\ML PROJECT\FINAL PROJECT ON ML- FRAUD DETECTION\Online Payment
```

The features in the dataset

step: represents a unit of time where 1 step equals 1 hour

type: type of online transaction

amount: the amount of the transaction

nameOrig:customer starting the transaction

oldbalanceOrg: balance before the transaction

newbalanceOrg: balance after the transaction

nameDest: recipient of the transaction

oldbalanceDest: initial balance of recepient before the transaction

newbalanceDest: the new balance of the receipient after the transaction

isFraud: fraud transaction

In [3]:
```
# Rename the column header

Fraud_D.columns= ["step", "type", "amount", "customer_starting_transaction", "bal_before_transaction",
            "bal_after_transaction", "recipient_of_transaction", "bal_of_recepient_before_transaction", "bal_of_
```

In [4]:
```
# View data (to give you first five rows)
Fraud_D.head()
```

Out[4]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_transaction
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]:
```
# View data (to give you last five rows)
Fraud_D.tail()
```

Out[5]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_tran
1048570	95	CASH_OUT	132557.35	C1179511630	479803.00	347245.65	C435
1048571	95	PAYMENT	9917.36	C1956161225	90545.00	80627.64	M668
1048572	95	PAYMENT	14140.05	C2037964975	20545.00	6404.95	M1355
1048573	95	PAYMENT	10020.05	C1633237354	90605.00	80584.95	M1964
1048574	95	PAYMENT	11450.03	C1264356443	80584.95	69134.92	M677

In [6]:

```
#Data Verification

Fraud_D.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 10 columns):
 #   Column                             Non-Null Count    Dtype  
---  ------                             --------------    -----  
 0   step                               1048575 non-null  int64  
 1   type                               1048575 non-null  object 
 2   amount                             1048575 non-null  float64
 3   customer_starting_transaction      1048575 non-null  object 
 4   bal_before_transaction             1048575 non-null  float64
 5   bal_after_transaction              1048575 non-null  float64
 6   recipient_of_transaction           1048575 non-null  object 
 7   bal_of_recepient_before_transaction  1048575 non-null  float64
 8   bal_of_receipient_after_transaction  1048575 non-null  float64
 9   fraud_transaction                  1048575 non-null  int64  
dtypes: float64(5), int64(2), object(3)
memory usage: 80.0+ MB
```

In [7]:

```
# statistical analysis of the data

Fraud_D.describe()
```

Out[7]:

	step	amount	bal_before_transaction	bal_after_transaction	bal_of_recepient_before_transaction	bal_of_receipi
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	
mean	2.696617e+01	1.586670e+05	8.740095e+05	8.938089e+05	9.781600e+05	
std	1.562325e+01	2.649409e+05	2.971751e+06	3.008271e+06	2.296780e+06	
min	1.000000e+00	1.000000e-01	0.000000e+00	0.000000e+00	0.000000e+00	
25%	1.500000e+01	1.214907e+04	0.000000e+00	0.000000e+00	0.000000e+00	
50%	2.000000e+01	7.634333e+04	1.600200e+04	0.000000e+00	1.263772e+05	
75%	3.900000e+01	2.137619e+05	1.366420e+05	1.746000e+05	9.159235e+05	
max	9.500000e+01	1.000000e+07	3.890000e+07	3.890000e+07	4.210000e+07	

In [8]:

```
Fraud_D.describe().astype(int)
```

Out[8]:

	step	amount	bal_before_transaction	bal_after_transaction	bal_of_recepient_before_transaction	bal_of_receipient_after_
count	1048575	1048575	1048575	1048575	1048575	
mean	26	158666	874009	893808	978160	
std	15	264940	2971750	3008271	2296780	
min	1	0	0	0	0	
25%	15	12149	0	0	0	
50%	20	76343	16002	0	126377	
75%	39	213761	136642	174599	915923	
max	95	10000000	38900000	38900000	42100000	

In [9]:

```
#Missing values

Fraud_D.isnull()
```

Out[9]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_transaction
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
1048570	False	False	False	False	False	False	False
1048571	False	False	False	False	False	False	False
1048572	False	False	False	False	False	False	False
1048573	False	False	False	False	False	False	False
1048574	False	False	False	False	False	False	False

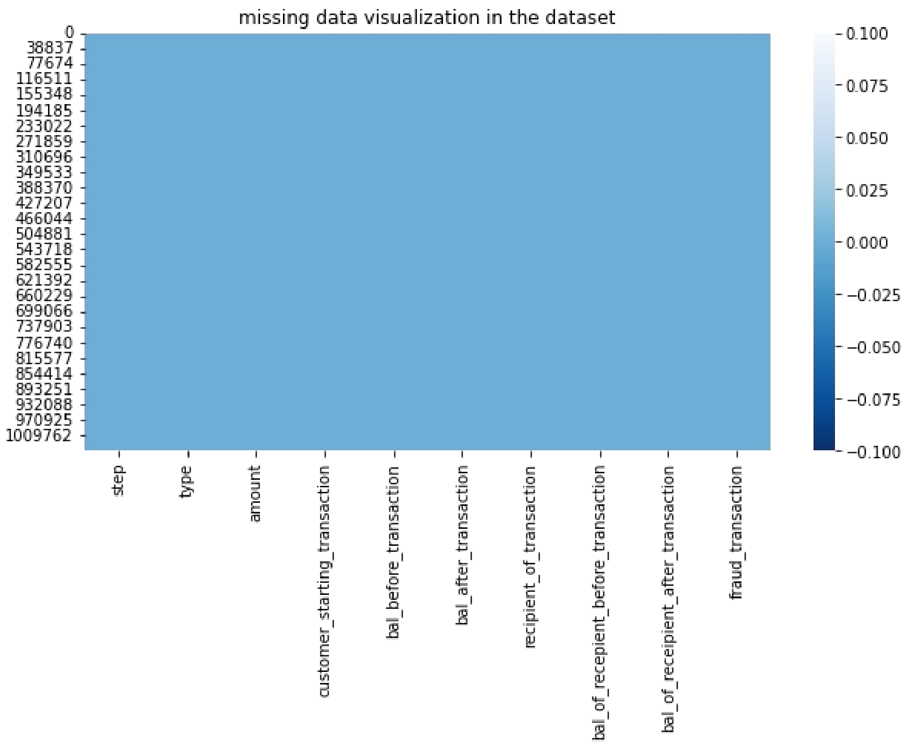
1048575 rows × 10 columns

In [10]:
```
Fraud_D.isnull().sum()
```

Out[10]:
```
step                                0
type                                0
amount                              0
customer_starting_transaction       0
bal_before_transaction              0
bal_after_transaction               0
recipient_of_transaction            0
bal_of_recepient_before_transaction 0
bal_of_receipient_after_transaction 0
fraud_transaction                   0
dtype: int64
```

In [11]:
```
# To visualize the missing values

plt.figure(figsize = (10,5))
plt.title ("missing data visualization in the dataset")
sns.heatmap(Fraud_D.isnull(), cbar =True, cmap= "Blues_r")
```

Out[11]: <AxesSubplot:title={'center':'missing data visualization in the dataset'}>



There is no missing values in the dataset

In [12]:
```
#check shape of the entire dataframe using .shape attribute
Fraud_D.shape
```

Out[12]: (1048575, 10)

We have 1,048,575 rows and 10 columns in the dataset

EXPLORATORY DATA ANALYSIS

Univariate Analysis

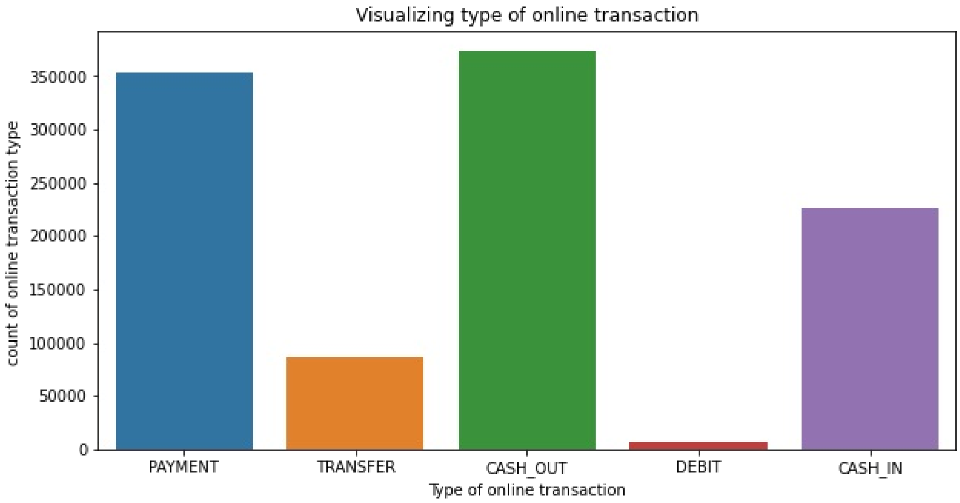
Bivariate Analysis

Multivariate Analysis

Correlation

In [13]:
```
# Univariate Analysis
#visualize type of online transaction
plt.figure(figsize=(10,5))
sns.countplot (x="type", data= Fraud_D)
plt.title ("Visualizing type of online transaction")
plt.xlabel("Type of online transaction")
plt.ylabel("count of online transaction type ")
```

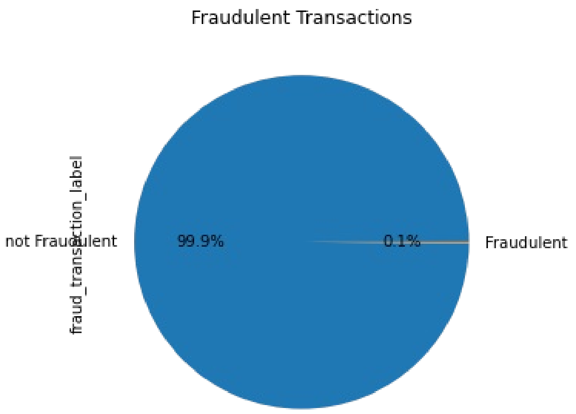
Out[13]: `Text(0, 0.5, 'count of online transaction type ')`



From the chart, it is seen that cash_out and payment is the most common type of online transaction that customers use

In [59]:
```
# create a function that properly labels isFraud

def Fraud (x):
    if x ==1:
        return "Fraudulent"
    else:
        return "not Fraudulent"

# create a new column
Fraud_D["fraud_transaction_label"] = Fraud_D["fraud_transaction"].apply(Fraud)


# create visualization
plt.figure(figsize = (10,5))
plt.title ("Fraudulent Transactions")
Fraud_D.fraud_transaction_label.value_counts().plot.pie(autopct='%1.1f%%')
```

Out[59]: `<AxesSubplot:title={'center':'Fraudulent Transactions'}, ylabel='fraud_transaction_label'>`



From this chart, its shows that most of the online transactions customers does is not fraudulent. Also the dataset is not balance

In [15]:
```
Fraud_D.fraud_transaction_label.value_counts()
```

Out[15]:
```
not Fraudulent    1047433
Fraudulent           1142
Name: fraud_transaction_label, dtype: int64
```

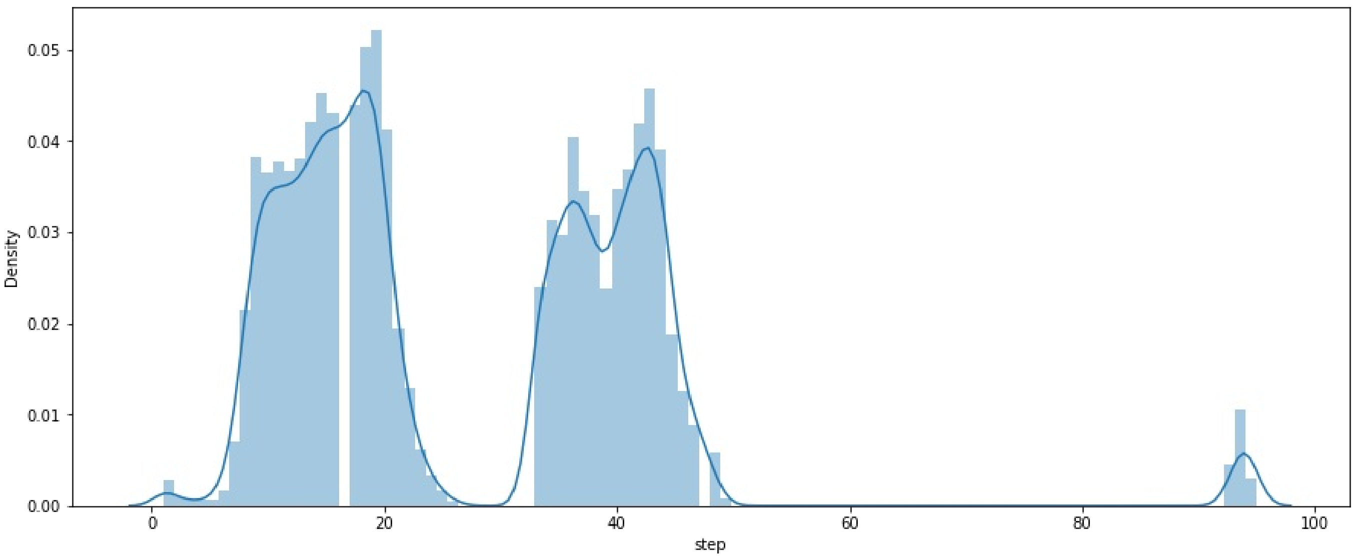
In [16]:
```
1142/1047433*100
```

Out[16]:
```
0.10902845337124188
```

1,142 transactions have been tagged as fraudulent in the dataset, which is approximately 11% of the total number of transactions.

In [17]:
```
#To disable warnings
import warnings
warnings.filterwarnings("ignore")

# Visualization for step column

plt.figure(figsize=(15,6))
sns.distplot(Fraud_D['step'],bins=100)
```

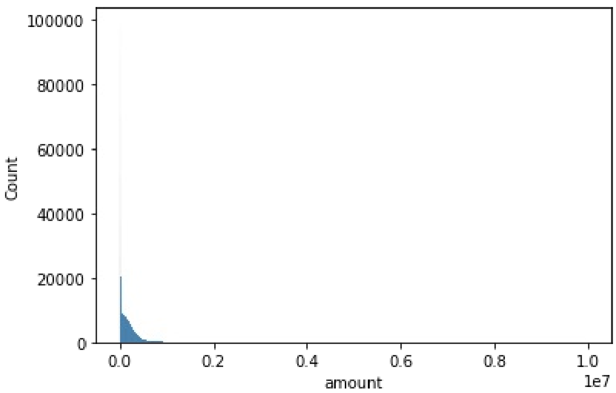
Out[17]:
```
<AxesSubplot:xlabel='step', ylabel='Density'>
```



The above graph indicates the distribution of the step column

In [18]:
```python
# Visualization for amount column

sns.histplot(x= "amount", data =Fraud_D)
```

Out[18]: <AxesSubplot:xlabel='amount', ylabel='Count'>



In [19]:
```python
Fraud_D.head()
```

Out[19]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_transaction
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

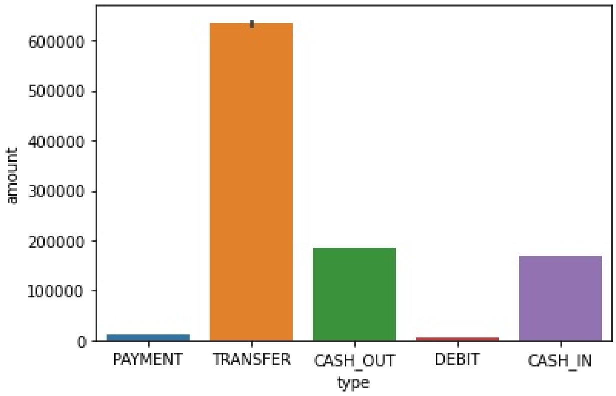
In [20]:
```python
Fraud_D.tail()
```

Out[20]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_tran
1048570	95	CASH_OUT	132557.35	C1179511630	479803.00	347245.65	C435
1048571	95	PAYMENT	9917.36	C1956161225	90545.00	80627.64	M668
1048572	95	PAYMENT	14140.05	C2037964975	20545.00	6404.95	M1355
1048573	95	PAYMENT	10020.05	C1633237354	90605.00	80584.95	M1964
1048574	95	PAYMENT	11450.03	C1264356443	80584.95	69134.92	M677

In [21]:
```
# Bivariate Analysis

sns.barplot(x='type',y='amount',data=Fraud_D)
```

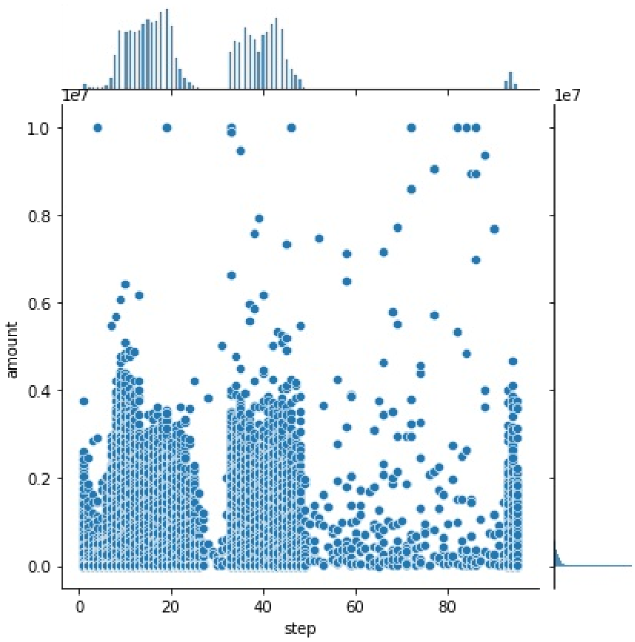
Out[21]: <AxesSubplot:xlabel='type', ylabel='amount'>



In this chart, 'transfer' type has the maximum amount of money being transfered from customers to the recipient. Although 'cash out' and 'payment' are the most common type of transactions

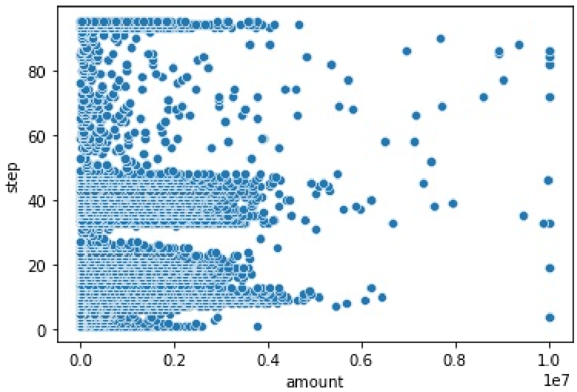
In [22]:
```
# Visualization between step and amount

sns.jointplot(x='step',y='amount',data=Fraud_D)
```

Out[22]: <seaborn.axisgrid.JointGrid at 0x138c56af460>



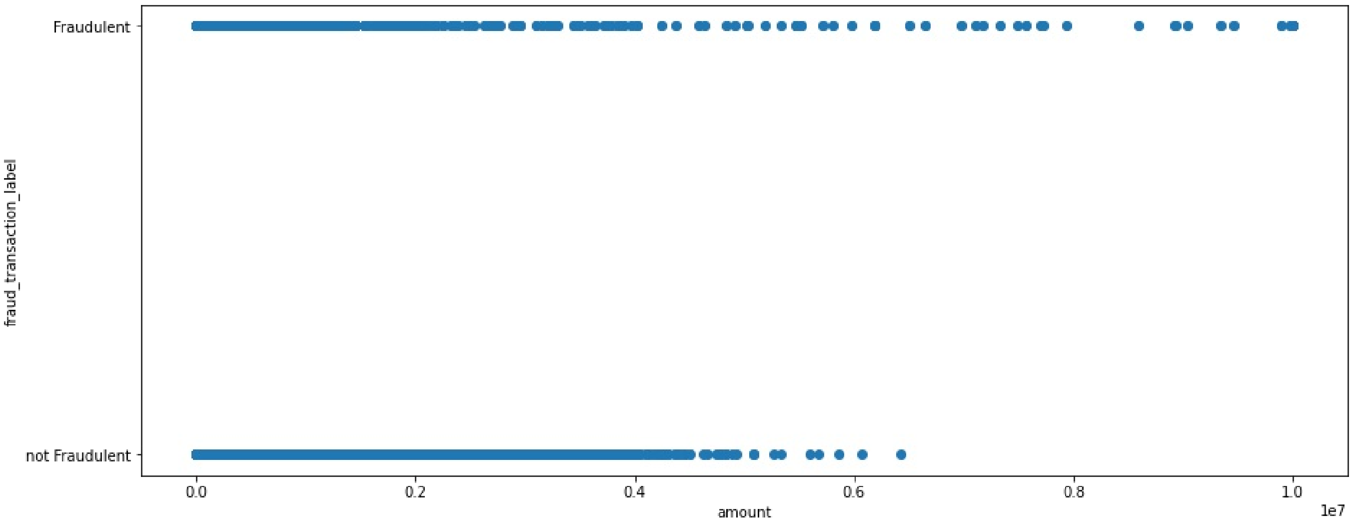
In [23]:
```
sns.scatterplot(x=Fraud_D["amount"], y=Fraud_D["step"])
```

Out[23]: <AxesSubplot:xlabel='amount', ylabel='step'>



In [24]:
```
# Visualization between amount and fraud_transaction_label

plt.figure(figsize=(15,6))
plt.scatter(x='amount',y='fraud_transaction_label',data=Fraud_D)
plt.xlabel('amount')
plt.ylabel('fraud_transaction_label')
```

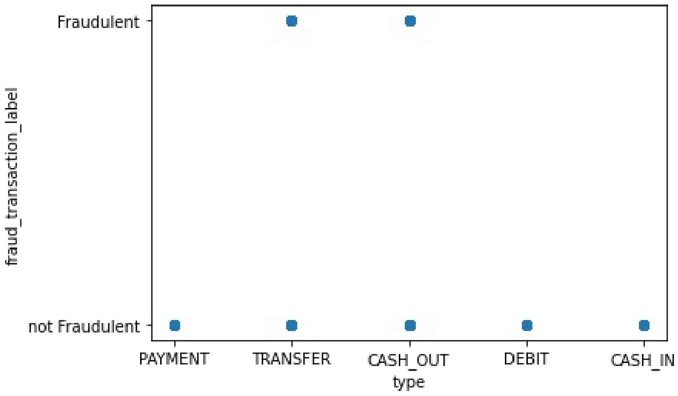
Out[24]: `Text(0, 0.5, 'fraud_transaction_label')`



Although the amount of fraudulent transactions is very low, majority of them are constituted within 0 and 10,000,000 amount.

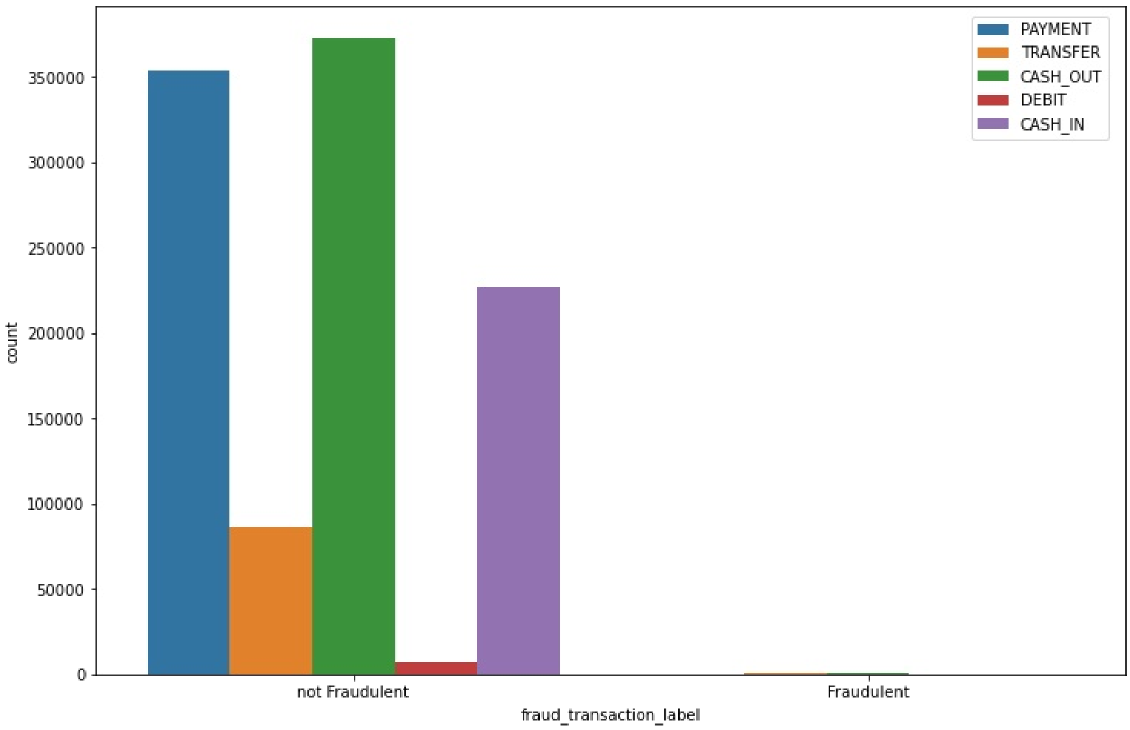
In [25]:
```
# Visualization between type and isfraud_label

plt.scatter(x='type',y='fraud_transaction_label',data=Fraud_D)
plt.xlabel('type')
plt.ylabel('fraud_transaction_label')
```

Out[25]: `Text(0, 0.5, 'fraud_transaction_label')`



In [26]:
```
# Visualization between type and isfraud_label

plt.figure(figsize=(12,8))
sns.countplot(x='fraud_transaction_label',data=Fraud_D,hue='type')
plt.legend(loc=[0.85,0.8])
```

Out[26]: <matplotlib.legend.Legend at 0x13884dbb280>

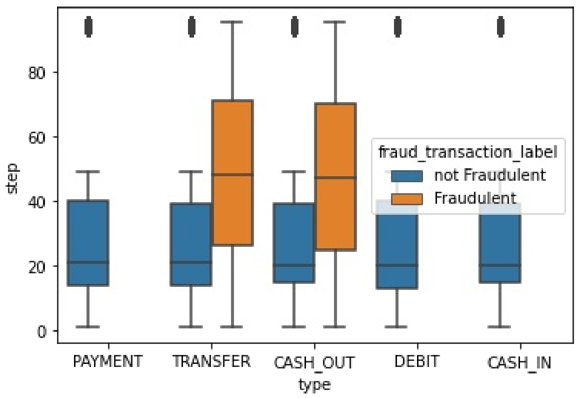


Both the above graphs indicate that transactions of the type 'transfer' and 'cash out' comprise fraudulent transactions

Multivariate Analysis

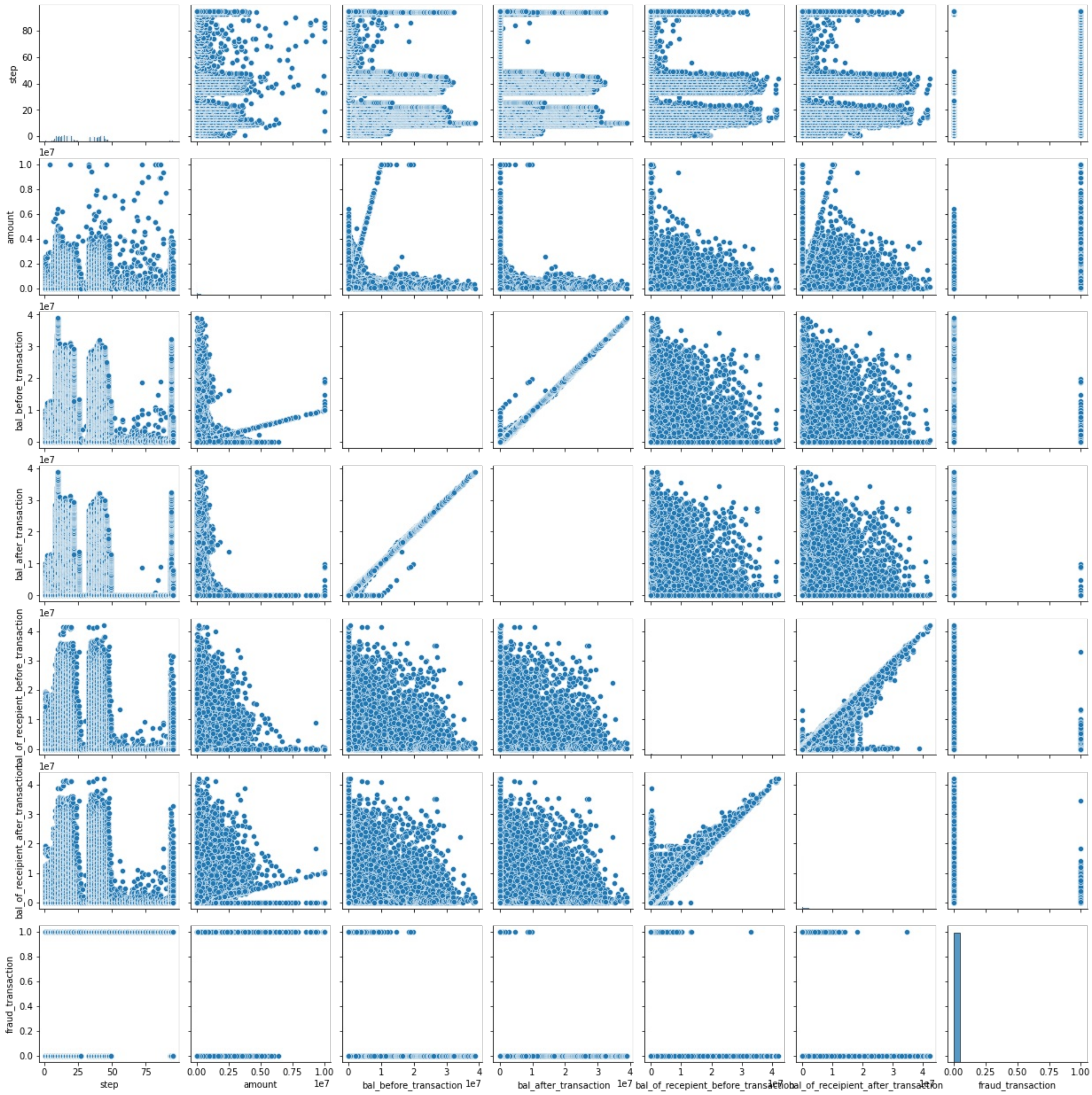
In [27]:
```
# Visualizing btw step,type and isFraud_label

sns.boxplot(x= "type", y= "step", hue ="fraud_transaction_label", data= Fraud_D)
```

Out[27]: <AxesSubplot:xlabel='type', ylabel='step'>



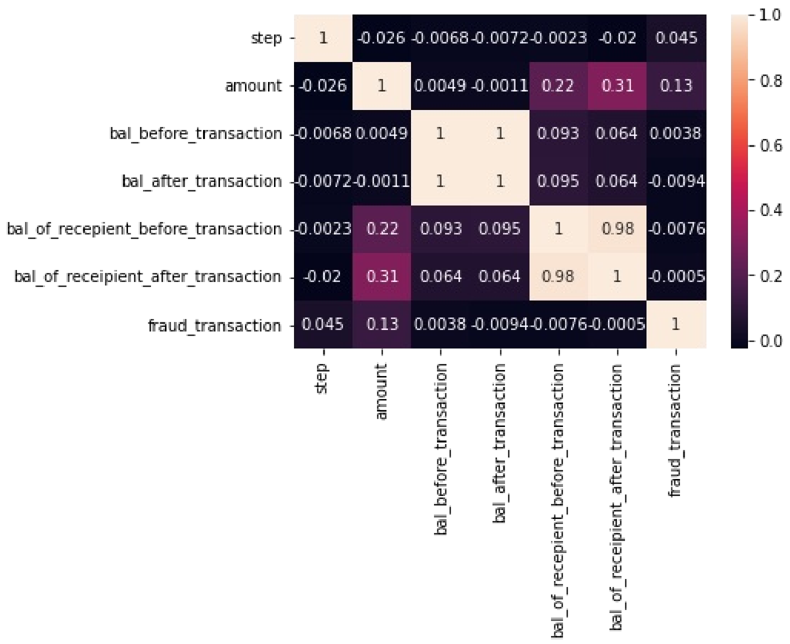
In [28]:
```
sns.pairplot(Fraud_D)
```

Out[28]: <seaborn.axisgrid.PairGrid at 0x138d378f850>



In [29]:
```
# Correlation

corel= Fraud_D.corr()
sns.heatmap(corel, annot =True)
```

Out[29]: <AxesSubplot:>



PERFORMING FEATURE ENGINERRING

Encoding categorical variables

In [30]:
```
# One Hot Encoding
#1. select categorical variables

categorical = ['type']
```

In [31]:
```
#2. use pd.get_dummies() for one hot encoding
#replace pass with your code

categories_dummies = pd.get_dummies(Fraud_D[categorical])

#view what you have done
categories_dummies.head()
```

Out[31]:

	type_CASH_IN	type_CASH_OUT	type_DEBIT	type_PAYMENT	type_TRANSFER
0	0	0	0	1	0
1	0	0	0	1	0
2	0	0	0	0	1
3	0	1	0	0	0
4	0	0	0	1	0

In [32]:
```
#join the encoded variables back to the main dataframe using pd.concat()
#pass both data and categories_dummies as a list of their names
#pop out documentation for pd.concat() to clarify

Fraud_D = pd.concat([Fraud_D,categories_dummies], axis=1)

#check what you have done
print(Fraud_D.shape)
Fraud_D.head()
```

```
(1048575, 16)
```

Out[32]:

	step	type	amount	customer_starting_transaction	bal_before_transaction	bal_after_transaction	recipient_of_transaction
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 [33]:
```
#remove the initial categorical columns now that we have encoded them
#use the list called categorical to delete all the initially selected columns at once
```

```
Fraud_D.drop(categorical, axis = 1, inplace = True)

Fraud_D.drop(columns=['fraud_transaction_label', 'customer_starting_transaction', 'recipient_of_transaction'], i
```

In [34]:
```
Fraud_D.head()
```

Out[34]:

	step	amount	bal_before_transaction	bal_after_transaction	bal_of_recepient_before_transaction	bal_of_receipient_after_transacti
0	1	9839.64	170136.0	160296.36	0.0	
1	1	1864.28	21249.0	19384.72	0.0	
2	1	181.00	181.0	0.00	0.0	
3	1	181.00	181.0	0.00	21182.0	
4	1	11668.14	41554.0	29885.86	0.0	

Model Selection, Training and Validation

Select Target

In [35]:
```
y = Fraud_D.fraud_transaction
```

Selecting Features

In [36]:
```
X = Fraud_D.drop(['fraud_transaction'], axis = 1)
```

In [37]:
```
X
```

Out[37]:

	step	amount	bal_before_transaction	bal_after_transaction	bal_of_recepient_before_transaction	bal_of_receipient_after_t
0	1	9839.64	170136.00	160296.36	0.00	
1	1	1864.28	21249.00	19384.72	0.00	
2	1	181.00	181.00	0.00	0.00	
3	1	181.00	181.00	0.00	21182.00	
4	1	11668.14	41554.00	29885.86	0.00	
...	...	...	...	...	...	
1048570	95	132557.35	479803.00	347245.65	484329.37	
1048571	95	9917.36	90545.00	80627.64	0.00	
1048572	95	14140.05	20545.00	6404.95	0.00	
1048573	95	10020.05	90605.00	80584.95	0.00	
1048574	95	11450.03	80584.95	69134.92	0.00	

1048575 rows × 11 columns

Import ML algorithms and initialize them

In [38]:
```
#import the libraries we will need
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

In [39]:
```
## Train test split( training on 80% while testing is 20%)

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

In [40]:
```
# Initialize each models
LR = LogisticRegression(random_state=42)
KN = KNeighborsClassifier()
DC = DecisionTreeClassifier(random_state=42)
RF = RandomForestClassifier(random_state=42)
```

In [41]:
```
#create list of your model names
models = [LR,KN,DC,RF]
```

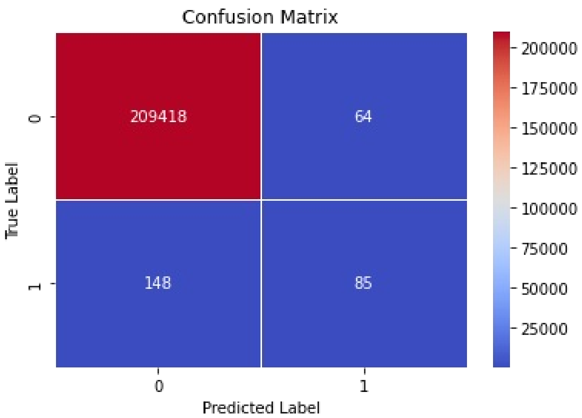
In [42]:
```
def plot_confusion_matrix(y_test,prediction):
    cm_ = confusion_matrix(y_test,prediction)
    plt.figure(figsize = (6,4))
    sns.heatmap(cm_, cmap ='coolwarm', linecolor = 'white', linewidths = 1, annot = True, fmt = 'd')
    plt.title('Confusion Matrix')
    plt.ylabel('True Label')
    plt.xlabel('Predicted Label')
    plt.show()
```

In [43]:
```
from sklearn.metrics import confusion_matrix
```

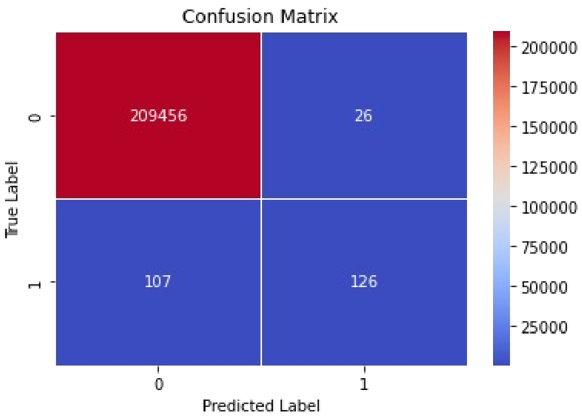
In [44]:
```
#create function to train a model and evaluate accuracy
def trainer(model,X_train,y_train,X_test,y_test):
    #fit your model
    model.fit(X_train,y_train)
    #predict on the fitted model
    prediction = model.predict(X_test)
    #print evaluation metric
    print('\nFor {}, Accuracy score is {} \n'.format(model.__class__.__name__,accuracy_score(prediction,y_test)
    print(classification_report(y_test, prediction)) #use this later
    plot_confusion_matrix(y_test,prediction)
```

In [45]:
```
#loop through each model, training in the process
for model in models:
    trainer(model,X_train,y_train,X_test,y_test)
```

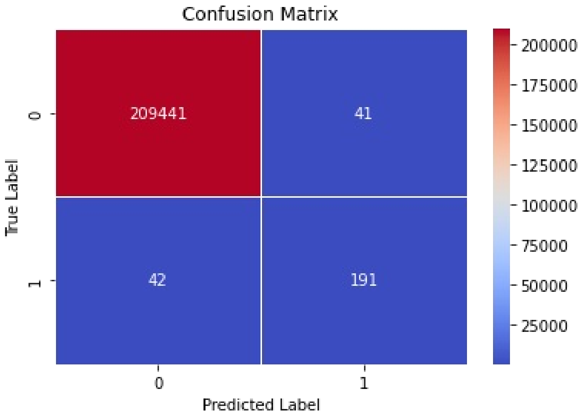
```
For LogisticRegression, Accuracy score is 0.9989891042605441

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209482
           1       0.57      0.36      0.45       233

    accuracy                           1.00    209715
   macro avg       0.78      0.68      0.72    209715
weighted avg       1.00      1.00      1.00    209715
```



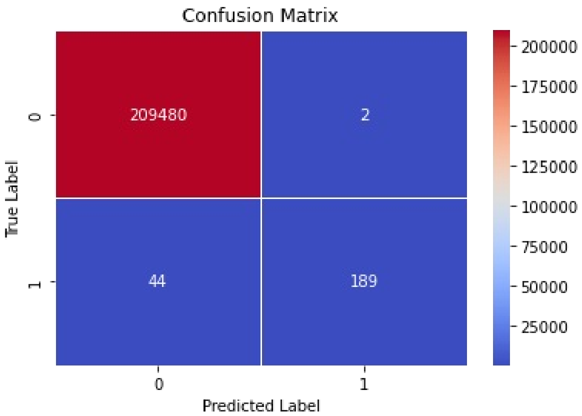

```
For KNeighborsClassifier, Accuracy score is 0.9993658059747753

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209482
           1       0.83      0.54      0.65       233

    accuracy                           1.00    209715
   macro avg       0.91      0.77      0.83    209715
weighted avg       1.00      1.00      1.00    209715
```



```
For DecisionTreeClassifier, Accuracy score is 0.9996042247812508

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209482
           1       0.82      0.82      0.82       233

    accuracy                           1.00    209715
   macro avg       0.91      0.91      0.91    209715
weighted avg       1.00      1.00      1.00    209715
```




```
For RandomForestClassifier, Accuracy score is 0.9997806546980426

              precision    recall  f1-score   support

           0       1.00      1.00      1.00    209482
           1       0.99      0.81      0.89       233

    accuracy                           1.00    209715
   macro avg       0.99      0.91      0.95    209715
weighted avg       1.00      1.00      1.00    209715
```



Interpretation of the result

The Decision Tree model with default parameters yields 99.96% accuracy on training data.

Precision Score: This means that 82% of all the things we predicted came true. that is 82% of clients transactions was detected to be a fraudulent transaction.

Recall Score: In all the actual positives, we only predicted 82% of it to be true.

Random Forest Tree model with default parameters yields 99.97% accuracy on training data.

Precision Score: This means that 99% of all the things we predicted came true. that is 99% of clients transactions was detected to be a fraudulent transaction.

Recall Score: In all the actual positives, we only predicted 81% of it to be true.

Both the Decision Tree and Random Forest models outperform the Logistic Regression and K-Nearest Neighbors model by a wide margin. Since they both have similar recall scores, we should perform a cross-validation of the two models so we may declare which is the best performer with more certainty.

Cross Validation

In [52]:

```
# Importing the library to perform cross-validation
from sklearn.model_selection import cross_validate

# Running the cross-validation on both Decision Tree and Random Forest models; specifying recall as the scoring
DC_scores = cross_validate(DC, X_test, y_test, scoring='recall_macro')
RF_scores = cross_validate(RF, X_test, y_test, scoring='recall_macro')

# Printing the means of the cross-validations for both models
print('Decision Tree Recall Cross-Validation:', np.mean(DC_scores['test_score']))
print('Random Forest Recall Cross-Validation:', np.mean(RF_scores['test_score']))
```

```
Decision Tree Recall Cross-Validation: 0.8645167523613637
Random Forest Recall Cross-Validation: 0.8733484545132477
```

Conclusion

Upon training and evaluating our classification model, we found that the Random Forest model performed the best by a narrow margin.

Therefore, Random Forest performs best with recall cross-validation accuracy of 87% which is important for our problem statement where false negative is our priority

Recommendation

Transaction History and Frequency - if unaccounted transactions occurs frequently we should confirm genuinity of the transaction with the customer

Repeated wrong PIN or Password - We should halt the transaction and alert the customer immediately.

Make customers to change PIN or password often

Instruct user to use own mobile or computers while doing transactions to avoid phishing attacks

Increased cybersecurity for banking websites and mobile applications

Two factor authentication for transaction

Ensure that blossom bank hire a data engineer that will ensure the dataset is accurate, balanced for proper EDA as there are too many outliers in this data set. This will enable the business to build machime learning models that predict outcomes more accurately with better performance.

In [ ]: