Fraud-Transaction-Detection

by Abhishek

In [ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

PREPROCESSING

Read and shape the data:

In [32]:
```
df=pd.read_csv('C:/Users/spike/Downloads/fraud.csv')
df.shape
```

Out[32]:
```
(6362620, 11)
```

In [11]:
```
df.head(200)
```

Out[11]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.00	0.00	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.00	0.00	0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.00	0.00	1
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.00	0.00	1
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.00	0.00	0
...	...	...	...	...	...	...	...	...	...	...
195	1	CASH_OUT	210370.09	C2121995675	0.0	0.00	C1170794006	1442298.03	22190.99	0
196	1	CASH_OUT	36437.06	C2120063568	0.0	0.00	C1740000325	154606.00	1363368.51	0
197	1	CASH_OUT	82691.56	C1620409359	0.0	0.00	C248609774	657983.89	6453430.91	0
198	1	CASH_OUT	338767.10	C691691381	0.0	0.00	C453211571	544481.28	3461666.05	0
199	1	CASH_OUT	187728.59	C264978436	0.0	0.00	C1360767589	394124.51	2107965.39	0

200 rows × 11 columns

In [12]:
```
df.tail(200)
```

Out[12]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFr
6362420	727	TRANSFER	124582.58	C651444933	124582.58	0.0	C1161818914	0.00	0.00	
6362421	727	CASH_OUT	124582.58	C1098290230	124582.58	0.0	C1739564153	320485.06	445067.64	
6362422	727	TRANSFER	263401.81	C806437930	263401.81	0.0	C1469754483	0.00	0.00	
6362423	727	CASH_OUT	263401.81	C850961884	263401.81	0.0	C1203132980	251586.80	514988.60	
6362424	727	TRANSFER	69039.64	C922622756	69039.64	0.0	C417851521	0.00	0.00	
...	...	...	...	...	...	...	...	...	...	
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C776919290	0.00	339682.13	
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C1881841831	0.00	0.00	
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C1365125890	68488.84	6379898.11	
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C2080388513	0.00	0.00	
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C873221189	6510099.11	7360101.63	

200 rows × 11 columns

ANALYSIS

In [13]:
```
df.isnull().values.any()
```

Out[13]:
```
False
```

In [14]:
```
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
```

This is a really big dataset with no NULL values having size over 500MB. This would take some time to train for a normal GPU.

In [16]:
```
legit = len(df[df.isFraud == 0])
fraud = len(df[df.isFraud == 1])
legit_percent = (legit / (fraud + legit)) * 100
fraud_percent = (fraud / (fraud + legit)) * 100

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

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

These results prove that this is a highly unbalanced data as Percentage of Legit transactions= 99.87 % and Percentage of Fraud transactions= 0.13 %.

SO DECISION TREES AND RANDOM FORESTS ARE GOOD METHODS FOR IMBALANCED DATA.

Merchants

In [38]:
```
X = df[df['nameDest'].str.contains('M')]
X.head()
```

Out[38]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFla
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	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0	
5	1	PAYMENT	7817.71	C90045638	53860.0	46042.29	M573487274	0.0	0.0	0	
6	1	PAYMENT	7107.77	C154988899	183195.0	176087.23	M408069119	0.0	0.0	0	

VISUALISATION

In [41]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

CORRELATION HEATMAP

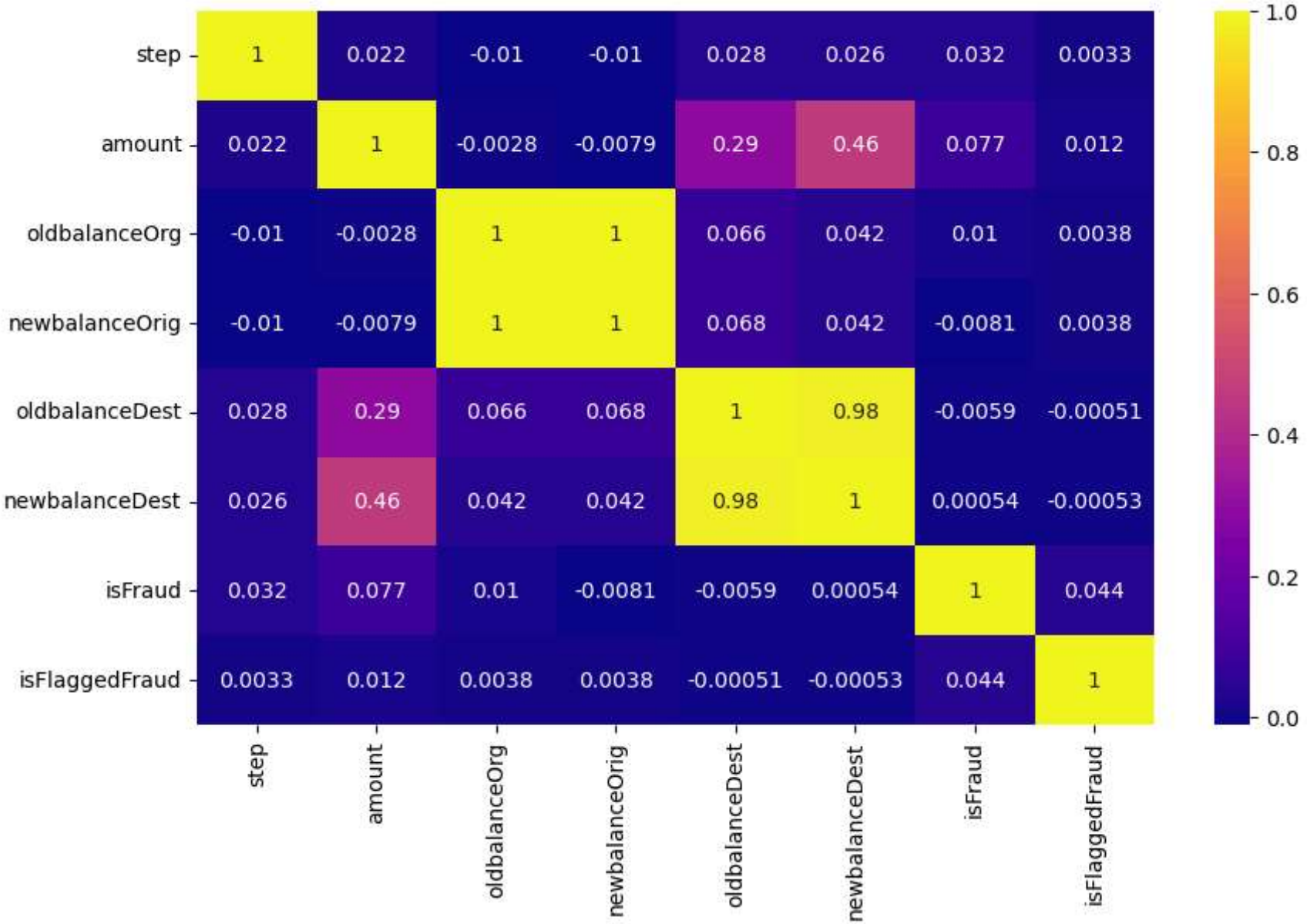
In [19]:
```
corr=df.corr(numeric_only=True)
corr
```

Out[19]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
step	1.000000	0.022373	-0.010058	-0.010299	0.027665	0.025888	0.031578	0.003277
amount	0.022373	1.000000	-0.002762	-0.007861	0.294137	0.459304	0.076688	0.012295
oldbalanceOrg	-0.010058	-0.002762	1.000000	0.998803	0.066243	0.042029	0.010154	0.003835
newbalanceOrig	-0.010299	-0.007861	0.998803	1.000000	0.067812	0.041837	-0.008148	0.003776
oldbalanceDest	0.027665	0.294137	0.066243	0.067812	1.000000	0.976569	-0.005885	-0.000513
newbalanceDest	0.025888	0.459304	0.042029	0.041837	0.976569	1.000000	0.000535	-0.000529
isFraud	0.031578	0.076688	0.010154	-0.008148	-0.005885	0.000535	1.000000	0.044109
isFlaggedFraud	0.003277	0.012295	0.003835	0.003776	-0.000513	-0.000529	0.044109	1.000000

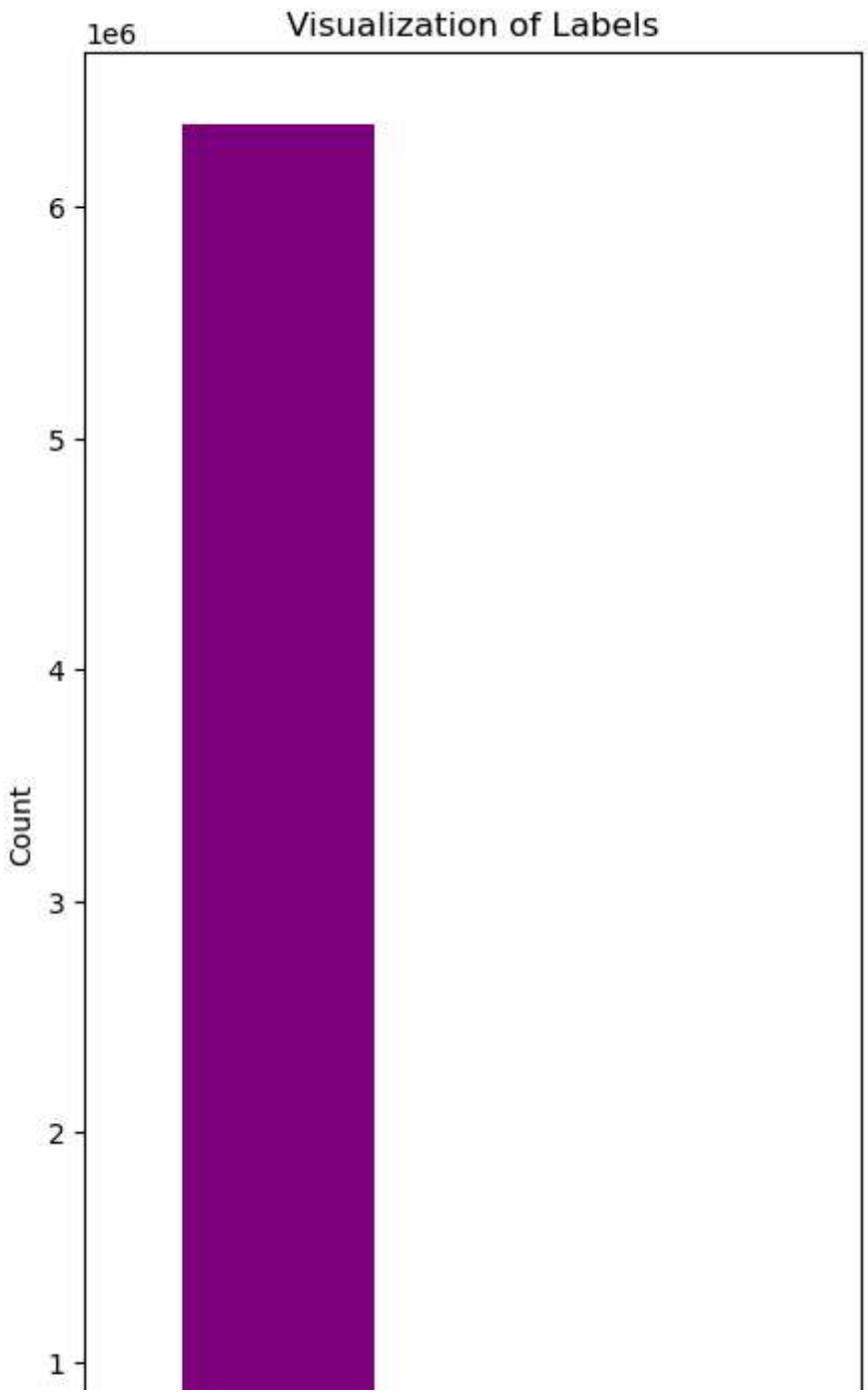
In [54]:

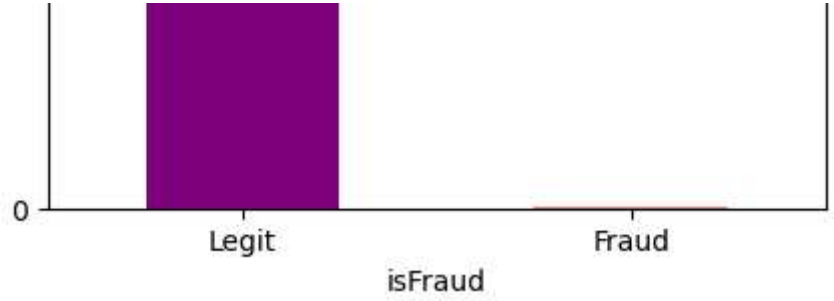
```
plt.figure(figsize=(10, 6))
sns.heatmap(corr, annot=True, cmap='plasma')
plt.show()
```



NUMBER OF LEGIT AND FRAUD TRANSACTIONS

In [57]:
```
import matplotlib.pyplot as plt

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


Visualization of Labels
1e6
6
5
4
3
2
1
Count



PROBLEM SOLVING

In [60]:
```
new_df=df.copy()
new_df.head()
```

Out[60]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFl
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	

LABEL ENCODING

In [63]:
```
objList = new_df.select_dtypes(include = "object").columns
print (objList)
```

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

There Are 3 Attributes With Object Datatype. Thus We Need To Label Encode Them In Order To Check Multicolinearity.

In [65]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
```

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

print (new_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype  
---  ------          -----  
 0   step            int64  
 1   type            int32  
 2   amount          float64
 3   nameOrig        int32  
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        int32  
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64  
 10  isFlaggedFraud  int64  
dtypes: float64(5), int32(3), int64(3)
memory usage: 461.2 MB
None
```

In [22]:
```
new_df.head()
```

Out[22]:

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

MULTICOLINEARITY

VARIANCE INFLATION FACTOR

In [72]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(df):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = df.columns
    vif["VIF"] = [variance_inflation_factor(df.values, i) for i in range(df.shape[1])]

    return(vif)

calc_vif(new_df)
```

Out[72]:

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

We can see that oldbalanceorg and newbalanceorig have too high VIF thus they are highly correlated. Similarly oldbalancedest and newbalancedest. Also namedest is connected to nameorig.

Thus combine these pairs of collinear attributes and drop the individual ones.

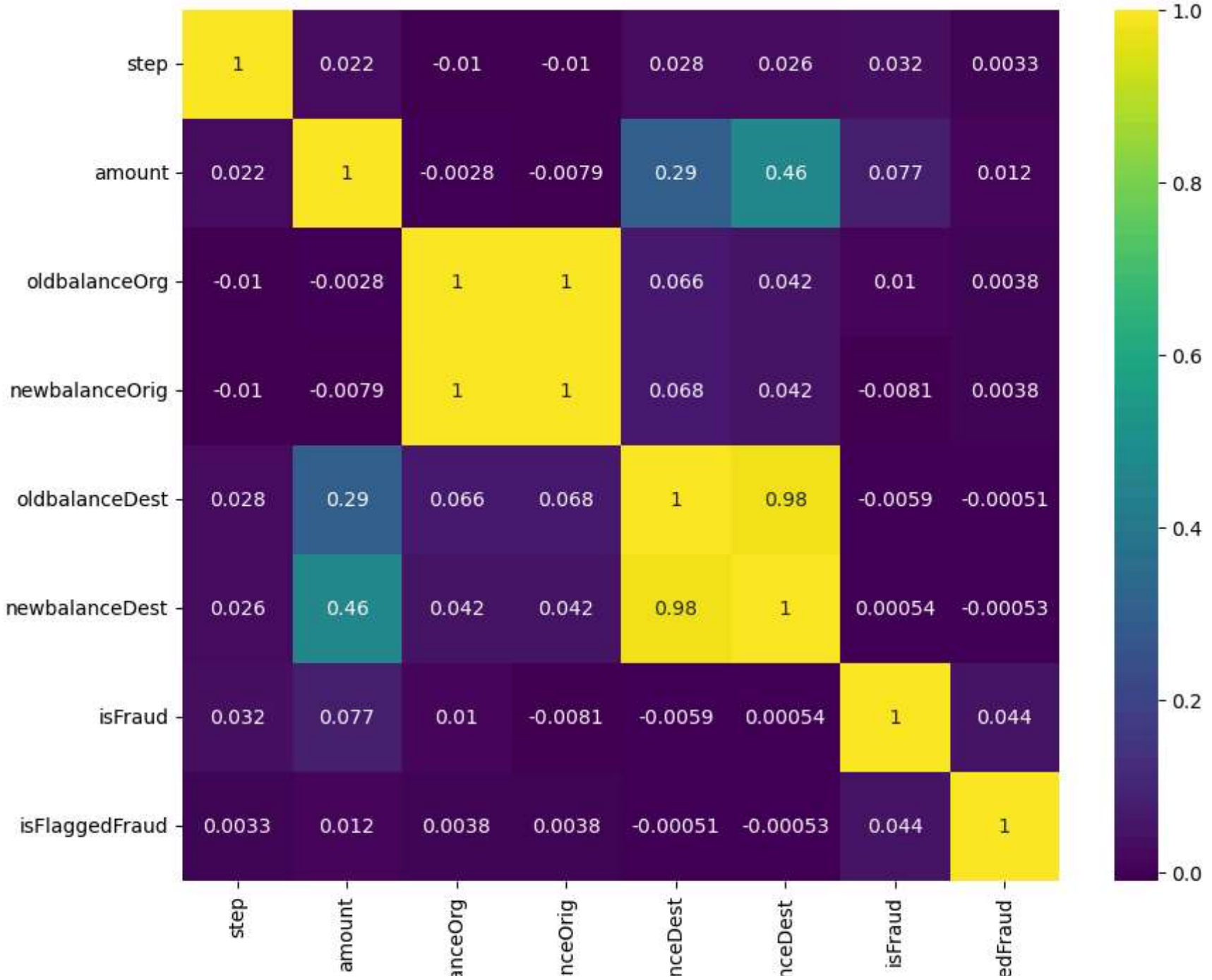
In [26]:
```
new_df['Actual_amount_orig'] = new_df.apply(lambda x: x['oldbalanceOrg'] - x['newbalanceOrig'],axis=1)
new_df['Actual_amount_dest'] = new_df.apply(lambda x: x['oldbalanceDest'] - x['newbalanceDest'],axis=1)
new_df['TransactionPath'] = new_df.apply(lambda x: x['nameOrig'] + x['nameDest'],axis=1)

#Dropping columns
new_df = new_df.drop(['oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest','step','nameOrig','nameDest'],axis=1)

calc_vif(new_df)
```

Out[26]:

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

In [76]:

```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))
sns.heatmap(corr, annot=True, cmap='viridis')
plt.show()
```


step 1 0.022 -0.01 -0.01 0.028 0.026 0.032 0.0033
amount 0.022 1 -0.0028 -0.0079 0.29 0.46 0.077 0.012
oldbalanceOrg -0.01 -0.0028 1 1 0.066 0.042 0.01 0.0038
newbalanceOrig -0.01 -0.0079 1 1 0.068 0.042 -0.0081 0.0038
oldbalanceDest 0.028 0.29 0.066 0.068 1 0.98 -0.0059 -0.00051
newbalanceDest 0.026 0.46 0.042 0.042 0.98 1 0.00054 -0.00053
isFraud 0.032 0.077 0.01 -0.0081 -0.0059 0.00054 1 0.044
isFlaggedFraud 0.0033 0.012 0.0038 0.0038 -0.00051 -0.00053 0.044 1
step
amount
anceOrg
nceOrig
nceDest
nceDest
isFraud
edFraud
1.0
0.8
0.6
0.4
0.2
0.0

How did you select variables to be included in the model?

Using the VIF values and correlation heatmap. We just need to check if there are any two attributes highly correlated to each other and then drop the one which is less correlated to the isFraud Attribute.

MODEL BUILDING

In [86]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
import itertools
from collections import Counter
import sklearn.metrics as metrics
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
```

NORMALIZING (SCALING) AMOUNT

Perform Scaling

In [88]:
```
scaler = StandardScaler()
new_df["NormalizedAmount"] = scaler.fit_transform(new_df["amount"].values.reshape(-1, 1))
new_df.drop(["amount"], inplace= True, axis= 1)

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

TRAIN-TEST SPLIT

Split the data

In [97]:
```
(X_train, X_test, Y_train, Y_test) = train_test_split(X, Y, test_size= 0.3, random_state= 42)

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

```
Shape of X_train:  (4453834, 10)
Shape of X_test:  (1908786, 10)
```

MODEL TRAINIG

DECISION TREE

In [100…
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)

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

RANDOM FOREST

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

Y_pred_rf = random_forest.predict(X_test)
random_forest_score = random_forest.score(X_test, Y_test) * 100
```

EVALUATION

In [48]:
```
print("Decision Tree Score: ", decision_tree_score)
print("Random Forest Score: ", random_forest_score)
```

```
Decision Tree Score:  99.92288292139612
Random Forest Score:  99.95887438403257
```

In [50]:
```
# key terms of Confusion Matrix - DT

print("TP,FP,TN,FN - Decision Tree")
```

```
tn, fp, fn, tp = confusion_matrix(Y_test, Y_pred_dt).ravel()
print(f'True Positives: {tp}')
print(f'False Positives: {fp}')
print(f'True Negatives: {tn}')
print(f'False Negatives: {fn}')

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

# key terms of Confusion Matrix - RF

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

```
TP,FP,TN,FN - Decision Tree
True Positives: 1716
False Positives: 753
True Negatives: 1905598
False Negatives: 719
--------------------------------------------------------------------------------
TP,FP,TN,FN - Random Forest
True Positives: 1712
False Positives: 62
True Negatives: 1906289
False Negatives: 723
```

TP(Decision Tree) ~ TP(Random Forest) so no competetion here.

FP(Decision Tree) >> FP(Random Forest) - Random Forest has an edge

TN(Decision Tree) < TN(Random Forest) - Random Forest is better here too

FN(Decision Tree) ~ FN(Random Forest)

Here Random Forest looks good.

In [52]:
```
# confusion matrix - DT
```

```
confusion_matrix_dt = confusion_matrix(Y_test, Y_pred_dt.round())
print("Confusion Matrix - Decision Tree")
print(confusion_matrix_dt,)

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

# confusion matrix - RF

confusion_matrix_rf = confusion_matrix(Y_test, Y_pred_rf.round())
print("Confusion Matrix - Random Forest")
print(confusion_matrix_rf)
```

```
Confusion Matrix - Decision Tree
[[1905598     753]
 [    719    1716]]
--------------------------------------------------------------------------------------------
Confusion Matrix - Random Forest
[[1906289      62]
 [    723    1712]]
```

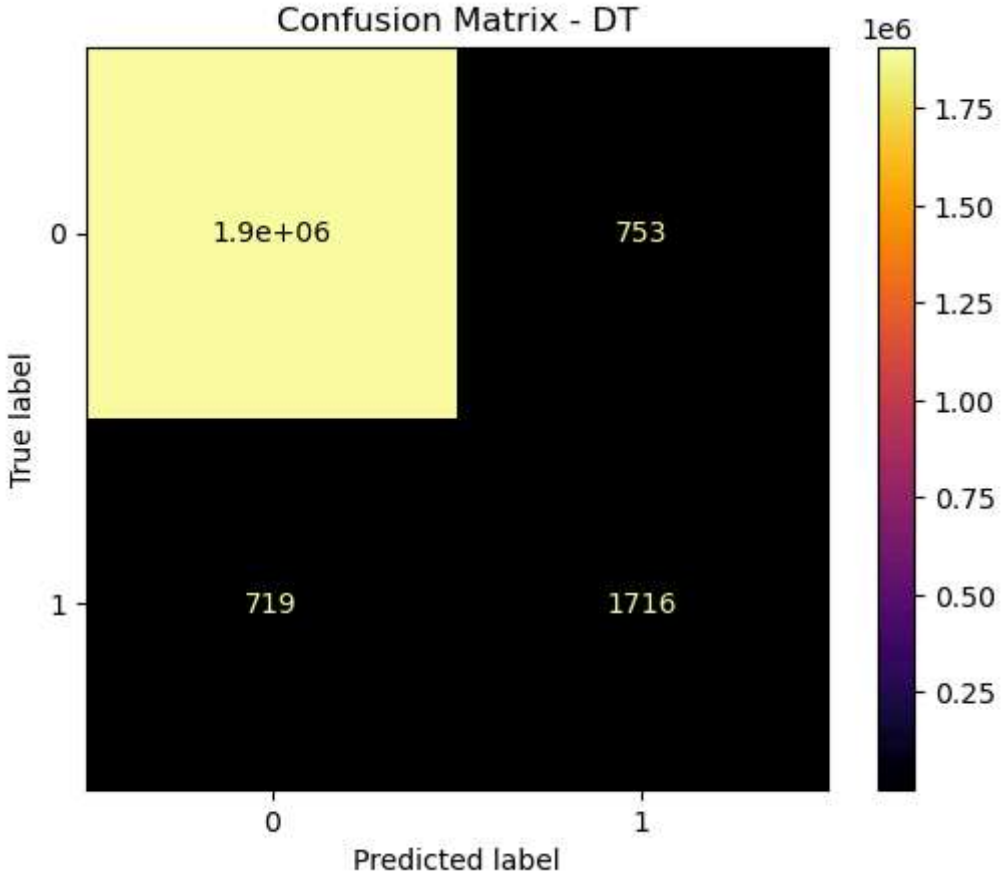
In [54]:

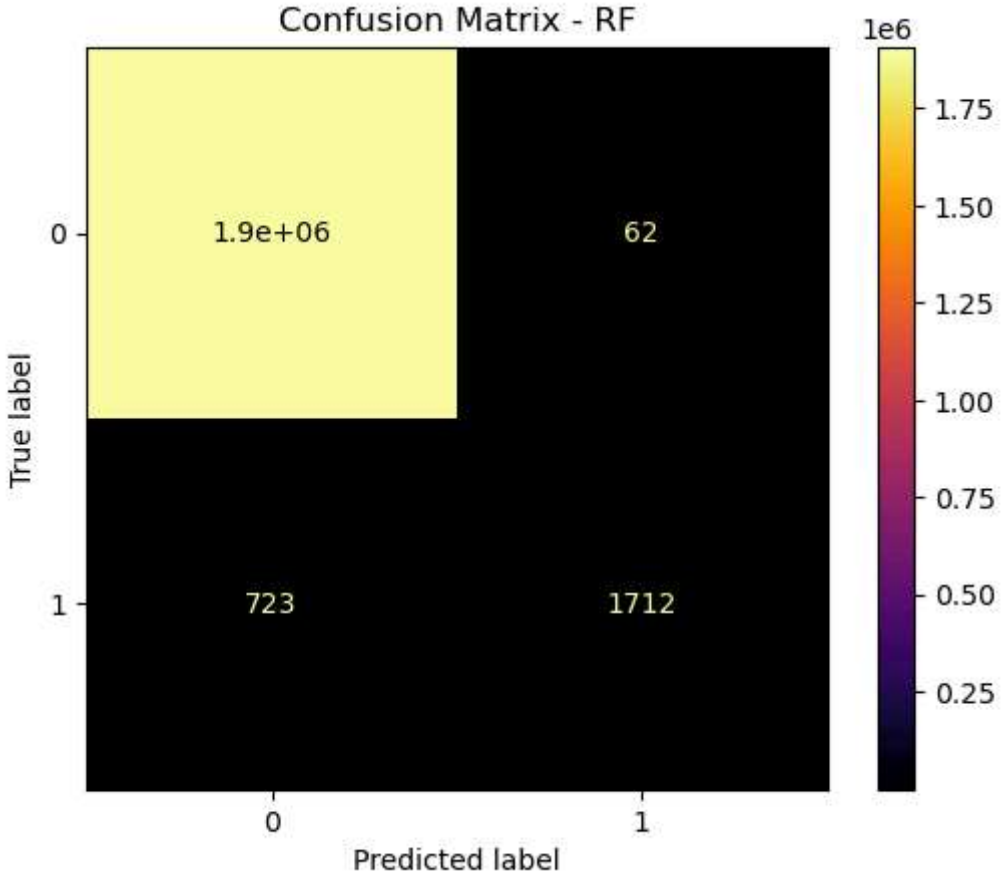
```
# classification report - DT

classification_report_dt = classification_report(Y_test, Y_pred_dt)
print("Classification Report - Decision Tree")
print(classification_report_dt)

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

# classification report - RF

classification_report_rf = classification_report(Y_test, Y_pred_rf)
print("Classification Report - Random Forest")
print(classification_report_rf)
```

```
Classification Report - Decision Tree
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1906351
           1       0.70      0.70      0.70      2435

    accuracy                           1.00   1908786
   macro avg       0.85      0.85      0.85   1908786
weighted avg       1.00      1.00      1.00   1908786

--------------------------------------------------------------------------------
Classification Report - Random Forest
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1906351
           1       0.97      0.70      0.81      2435

    accuracy                           1.00   1908786
   macro avg       0.98      0.85      0.91   1908786
weighted avg       1.00      1.00      1.00   1908786
```

With Such a good precision and hence F1-Score, Random Forest comes out to be better as expected.

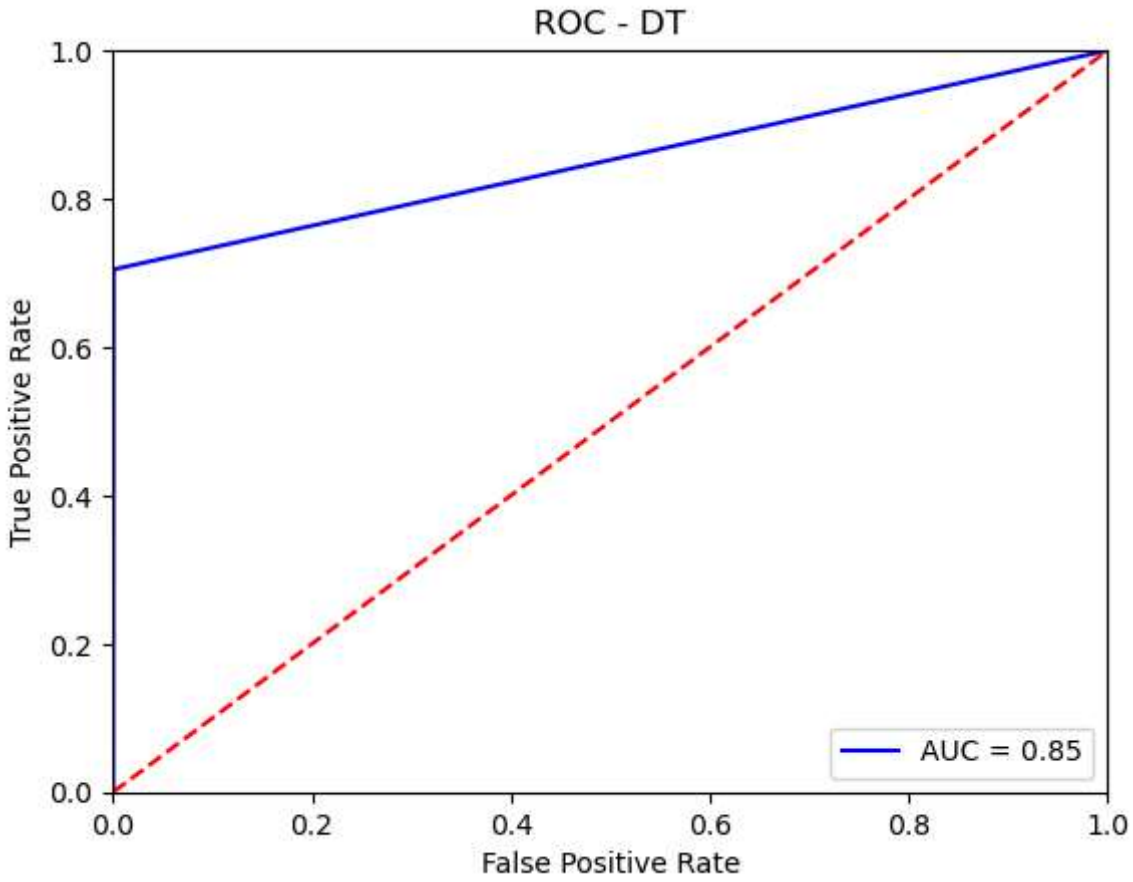
In [68]:
```python
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# Visualizing confusion matrix - DT
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_dt)
disp.plot(cmap='inferno')  # Change 'Blues' to any other colormap
plt.title('Confusion Matrix - DT')
plt.show()

# Visualizing confusion matrix - RF
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_rf)
disp.plot(cmap='inferno')  # Change 'Greens' to any other colormap
plt.title('Confusion Matrix - RF')
plt.show()
```

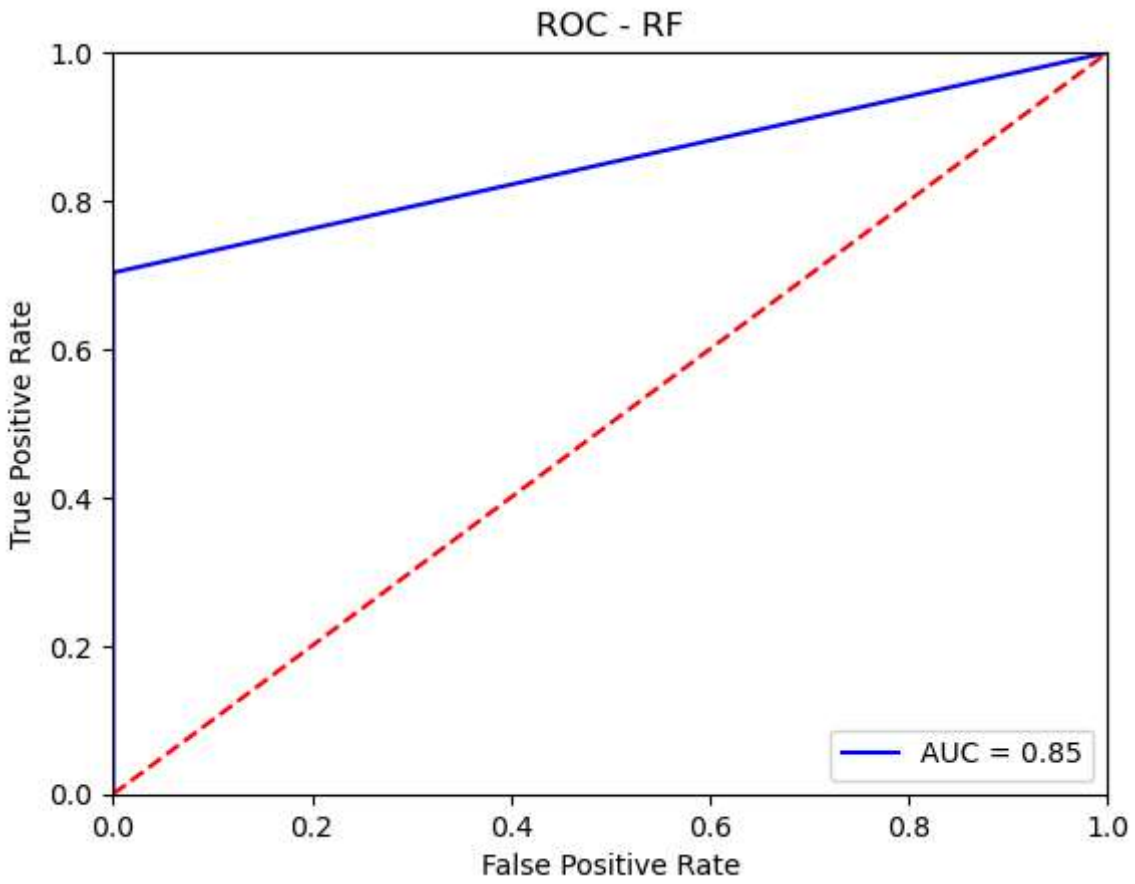

Confusion Matrix - DT
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0
1
True label
1.9e+06
753
719
1716
0
1
Predicted label



In [70]:
```
# AUC ROC - DT
# calculate the fpr and tpr for all thresholds of the classification

fpr, tpr, threshold = metrics.roc_curve(Y_test, Y_pred_dt)
roc_auc = metrics.auc(fpr, tpr)

plt.title('ROC - DT')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
```

```
plt.xlabel('False Positive Rate')
plt.show()

# AUC ROC - RF
# calculate the fpr and tpr for all thresholds of the classification

fpr, tpr, threshold = metrics.roc_curve(Y_test, Y_pred_rf)
roc_auc = metrics.auc(fpr, tpr)

plt.title('ROC - RF')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```


ROC - DT
True Positive Rate
False Positive Rate
AUC = 0.85
0.0
0.2
0.4
0.6
0.8
1.0



THE AUC for both Decision Tree and Random Forest is equal, so both models are pretty good at what they do.

CONCLUSION

We have seen that Accuracy of both Random Forest and Decision Tree is equal, although teh precision of Random Forest is more. In a fraud detection model, Precision is highly important because rather than predicting normal transactions correctly we want Fraud transactions to be predicted correctly and Legit to be left off.If either of the two reasons are not fulfiiled we may catch the innocent and leave the culprit.

This is also one of the reason why Random Forest and Decision Tree are used unstead of other algorithms.

In [ ]:

In [ ]:

CANDIDATE EXPECTATIONS:

1. Data cleaning including missing values, outliers and multi-collinearity.
For data cleaning, I would first check for missing values and remove any rows with missing- values. I would then check for outliers in the amount column and remove any extreme values that do not fall within a certain range. I would also check for multi-collinearity among the variables and remove any highly correlated variables.

2. Describe your fraud detection model in elaboration.
I would use a supervised machine learning model, such as Random Forest or GB classifier, to classify transactions as fraudulent or non-fraudulent. I would also use feature engineering to create new variables that may be useful in detecting fraud, such as the ratio of the transaction amount to the initial balance or amount thrushold.

3. How did you select variables to be included in the model?
I would select variables to be included in the model based on their correlation with the target variable (isFraud) and their importance in detecting fraud. Variables such as type, amount, and the initial and final balances of both the customer and the recipient would be important for the model.

4. Demonstrate the performance of the model by using best set of tools.
I would use metrics such as precision, recall, F1 score, accuracy to evaluate the performance of the Model and roc as well. I would also use cross-validation to ensure that the model is not overfitting to the training
data.

5. What are the key factors that predict fraudulent customer?
The key factors that predict fraudulent customers would likely include high transaction amounts, sudden changes in account balances, and transactions involving multiple parties.

6. Do these factors make sense? If yes, How? If not, How not?
These factors make sense as they are indicative of suspicious or unusual activity that may be indicative of fraud. High transaction amounts and sudden changes in account balances may be indicative of an attempt to steal funds, while transactions involving multiple parties may be indicative of money laundering or other illegal activities.

7. What kind of prevention should be adopted while company update its infrastructure?
To prevent fraud, the company could implement measures such as transaction monitoring, account monitoring, and two-factor authentication. Additionally, the company could also train its employees to identify and report suspicious activity.

8. Assuming these actions have been implemented, how would you determine if they work?
To determine if these actions are effective, the company could track the number of fraudulent transactions before and after the implementation of these measures. Additionally, the company could also conduct regular audits and assessments to identify any areas where the system may be vulnerable to fraud.