In [1]:
```
import numpy as np
import pandas as pd
```

### WRANGLING

In [2]:
```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```
# Get head of the data
Fraud_data = pd.read_csv("Fraud.csv")
Fraud_data.head()
```

Out[3]:

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

In [4]:
```
Fraud_data.shape
```

Out[4]: (6362620, 11)

### ANALYSIS

In [5]:
```
# Check for null values
Fraud_data.isnull().values.any()
```

Out[5]: False

In [6]:
```
# Getting information about data
Fraud_data.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 [7]:
```
# Checking the percentage of missing values
Fraud_data.isnull().sum()
```

Out[7]:
```
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 [8]:
```
# Viewing number of Authorised as well as Fraud transactions

Authorized = len(Fraud_data[Fraud_data.isFraud == 0])
Fraud = len(Fraud_data[Fraud_data.isFraud == 1])
print("Number of Authorized transactions: ", Authorized)
print("Number of Fraud transactions: ", Fraud)
```

```
Number of Authorized transactions:  6354407
Number of Fraud transactions:  8213
```

In [9]:
```
#Viewing percentage of authorised as well as Fraud transactions

Authorized_percent = (Authorized / (Fraud + Authorized)) * 100
Fraud_percent = (Fraud / (Fraud + Authorized)) * 100
print("Percentage of Authorized transactions: {:.4f} %".format(Authorized_p
print("Percentage of Fraud transactions: {:.4f} %".format(Fraud_percent))
```

```
Percentage of Authorized 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.

In [10]:
```
# Merchants
X = Fraud_data[Fraud_data['nameDest'].str.contains('M')]
X.head()
```

Out[10]:

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest | o |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | PAYMENT | 9839.64 | C1231006815 | 170136.0 | 160296.36 | M1979787155 | |
| **1** | 1 | PAYMENT | 1864.28 | C1666544295 | 21249.0 | 19384.72 | M2044282225 | |
| **4** | 1 | PAYMENT | 11668.14 | C2048537720 | 41554.0 | 29885.86 | M1230701703 | |
| **5** | 1 | PAYMENT | 7817.71 | C90045638 | 53860.0 | 46042.29 | M573487274 | |
| **6** | 1 | PAYMENT | 7107.77 | C154988899 | 183195.0 | 176087.23 | M408069119 | |

For merchants there is no information regarding the attribites oldbalanceDest and newbalanceDest.

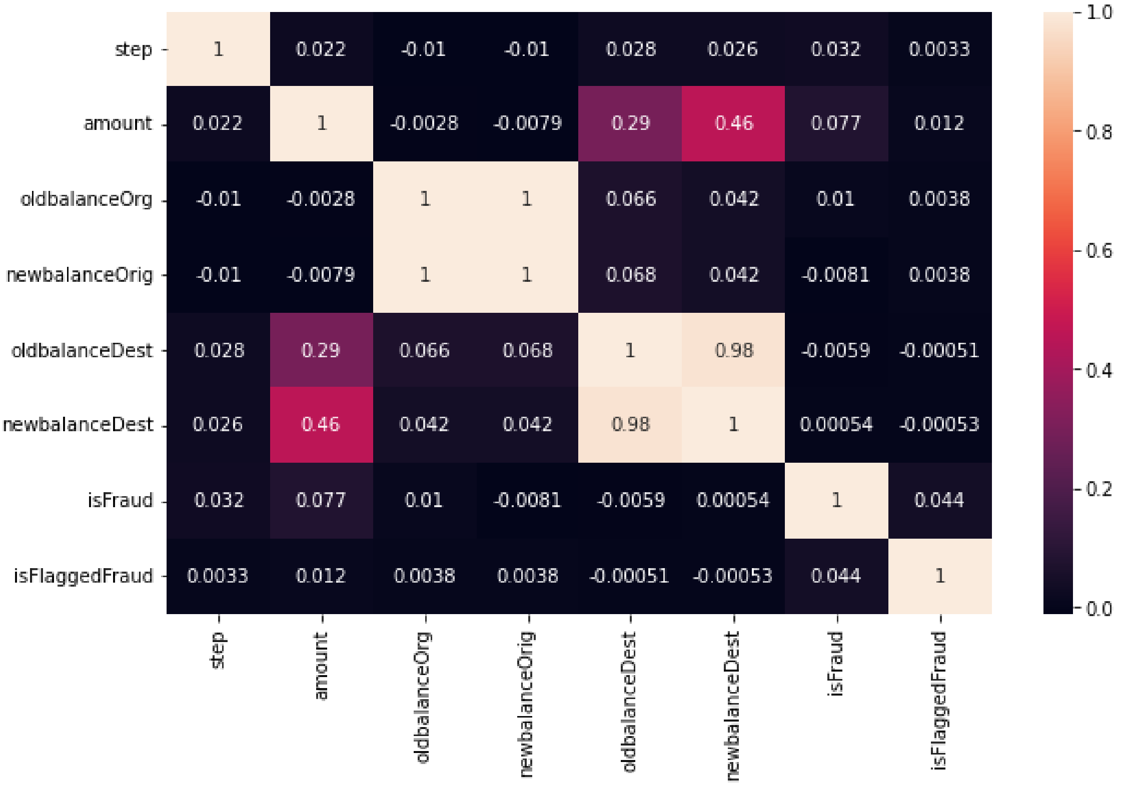
# VISUALISATION

In [11]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

### CORRELATION HEATMAP

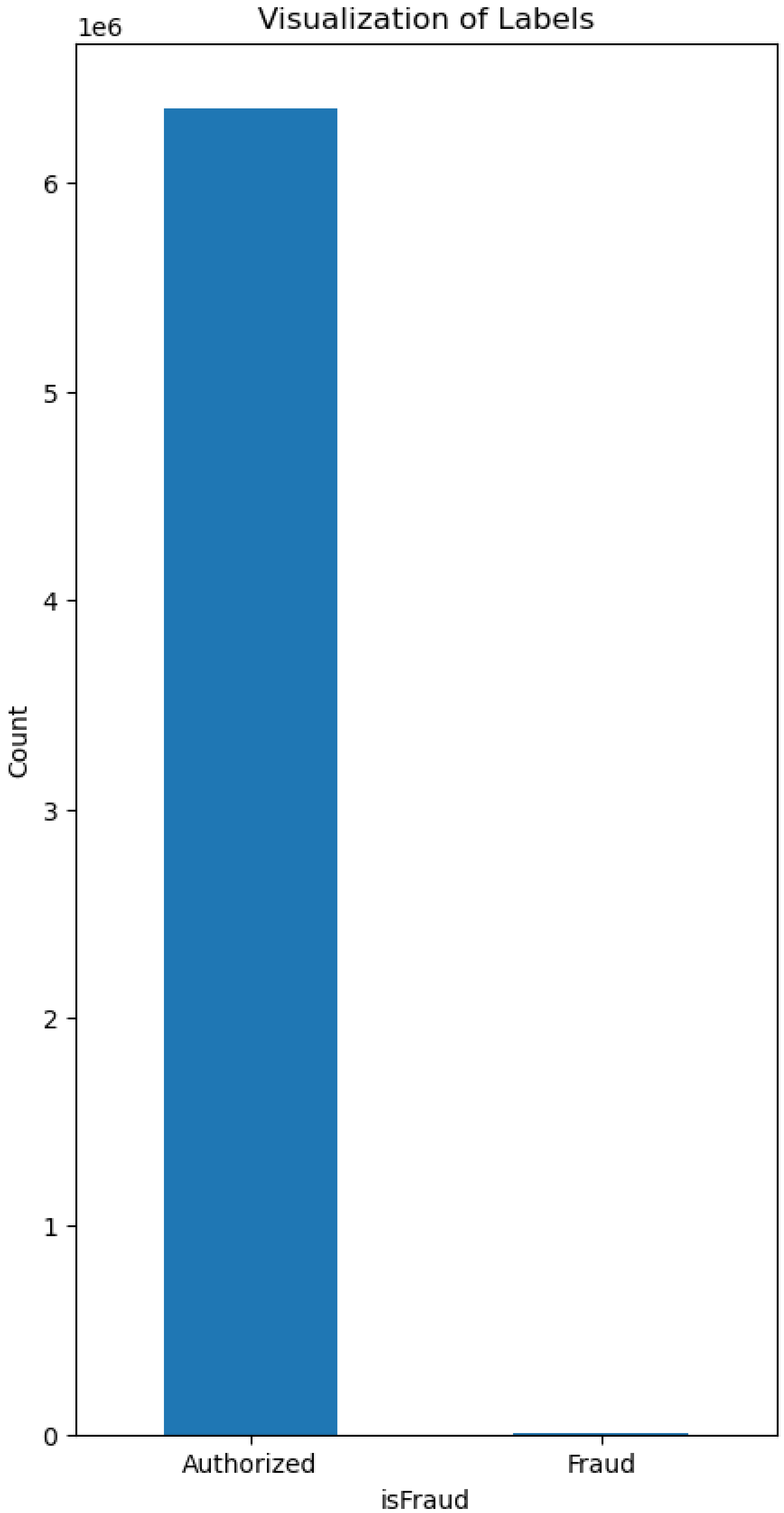
In [10]:
```
corr=df.corr()

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

Out[10]: <AxesSubplot:>




**NUMBER OF LEGIT AND FRAUD TRANSACTIONS**

In [13]:
```
plt.figure(figsize=(5,10))
labels = ["Authorized", "Fraud"]
count_classes = Fraud_data.value_counts(Fraud_data['isFraud'], sort= True)
count_classes.plot(kind = "bar", rot = 0)
plt.title("Visualization of Labels")
plt.ylabel("Count")
plt.xticks(range(2), labels)
plt.show()
```



# PROBLEM SOLVING

In [14]:
```
#creating a copy of original dataset to train and test models

new_Fraud_data=Fraud_data.copy()
new_Fraud_data.head()
```

Out[14]:

| | step | type | amount | nameOrig | oldbalanceOrg | newbalanceOrig | nameDest |
|---|---|---|---|---|---|---|---|
| **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 |

### LABEL ENCODING

In [15]:
```
# Checking how many attributes are dtype: object

objList = new_Fraud_data.select_dtypes(include = "object").columns
print (objList)
```

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

In [ ]:
```
# Create a LabelEncoder object
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for feat in objList:
    new_Fraud_data[feat] = le.fit_transform(new_Fraud_data[feat].astype(str
print(new_Fraud_data.info())
```

THERE ARE 3 ATTRIBUTES WITH Object Datatype. THUS WE NEED TO LABEL ENCODE THEM IN ORDER TO CHECK MULTICOLINEARITY.

In [17]:
```
new_Fraud_data.head()
```

Out[17]:

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

### MULTICOLINEARITY

In [16]:

```
# Import library for VIF (VARIANCE INFLATION FACTOR)

from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(Fraud_data):

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

    return(vif)

calc_vif(new_Fraud_data)
```

Out[16]:

| | 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.

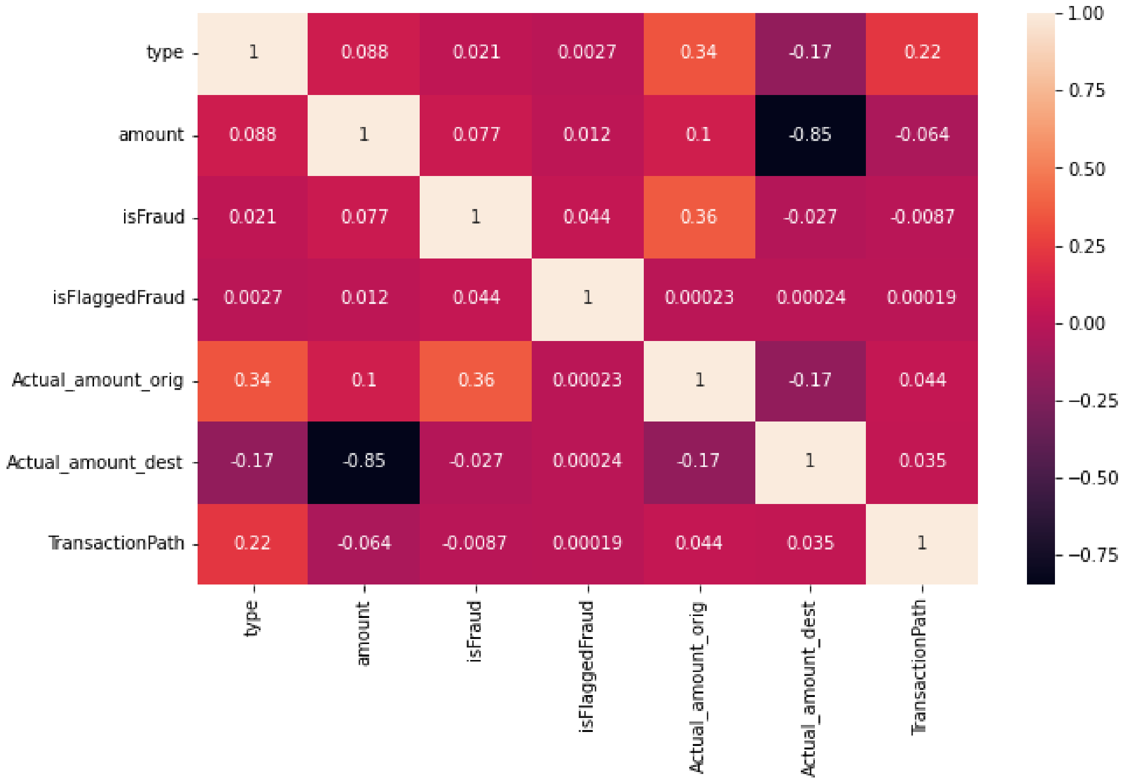
In [17]:
```
new_Fraud_data['Actual_amount_orig'] = new_Fraud_data.apply(lambda x: x['ol
new_Fraud_data['Actual_amount_dest'] = new_Fraud_data.apply(lambda x: x['ol
new_Fraud_data['TransactionPath'] = new_Fraud_data.apply(lambda x: x['nameC

#Dropping columns
new_Fraud_data = new_Fraud_data.drop(['oldbalanceOrg','newbalanceOrig','olc

calc_vif(new_Fraud_data)
```

Out[17]:

| | 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 [18]:
```
corr = new_Fraud_data.corr()

plt.figure(figsize=(10, 6))
sns.heatmap(corr, annot=True)
plt.title('Correlation Matrix')
plt.show()
```

Out[18]: <AxesSubplot:>



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 [19]:
```
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, Confus
```

#### NORMALIZING (SCALING) AMOUNT

In [20]:
```
# Perform Scaling
scaler = StandardScaler()
new_Fraud_data["NormalizedAmount"] = scaler.fit_transform(new_Fraud_data["a
new_Fraud_data.drop(["amount"], inplace= True, axis= 1)

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

I did'nt normalize the complete dataset because it can lead to decrease in accuracy of model.

#### TRAIN-TEST SPLIT

In [21]:
```
# Split the data
(X_train, X_test, Y_train, Y_test) = train_test_split(X, Y, test_size= 0.3,

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

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

#### MODEL TRAINIG

In [22]:
```
# DECISION TREE

decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)

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

In [23]:
```
# RANDOM FOREST

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 [24]:
```
# Print scores of our classifiers

print("Decision Tree Score: ", decision_tree_score)
print("Random Forest Score: ", random_forest_score)
```

```
Decision Tree Score:  99.92361637187196
Random Forest Score:  99.95856004811435
```

In [25]:
```
# 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: 1719
False Positives: 742
True Negatives: 1905609
False Negatives: 716
--------------------------------------------------------------------------
--------------
TP,FP,TN,FN - Random Forest
True Positives: 1712
False Positives: 68
True Negatives: 1906283
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 [26]:
```
# 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)
```

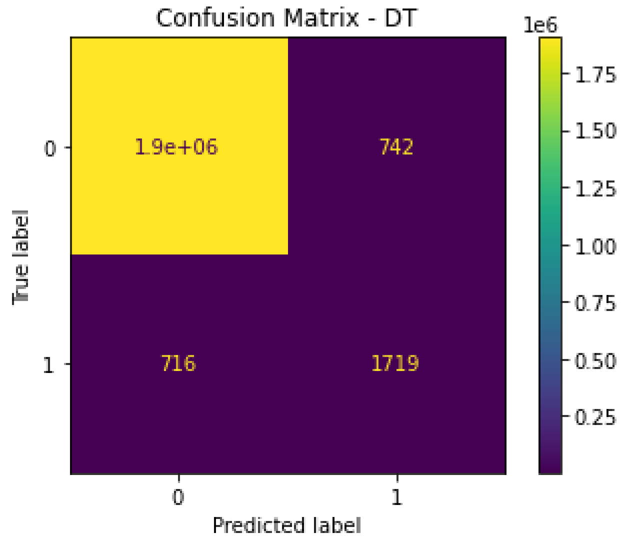
```
Confusion Matrix - Decision Tree
[[1905609     742]
 [    716    1719]]
--------------------------------------------------------------------------
--------------
Confusion Matrix - Random Forest
[[1906283      68]
 [    723    1712]]
```

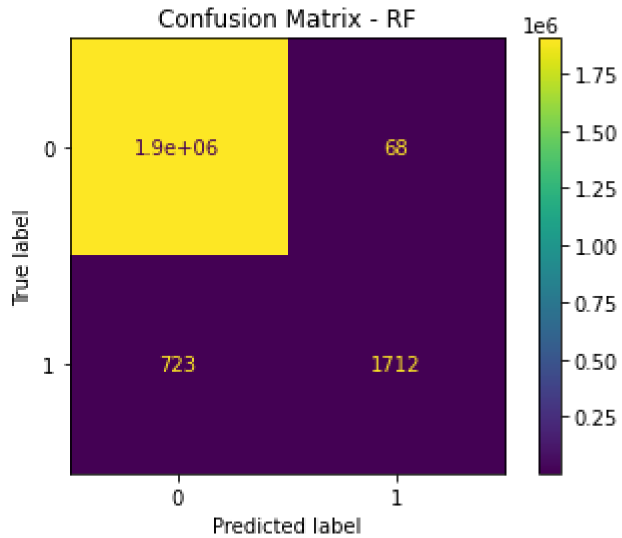
In [27]:
```
# classification report - DT

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      0.71      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       0.96      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
```

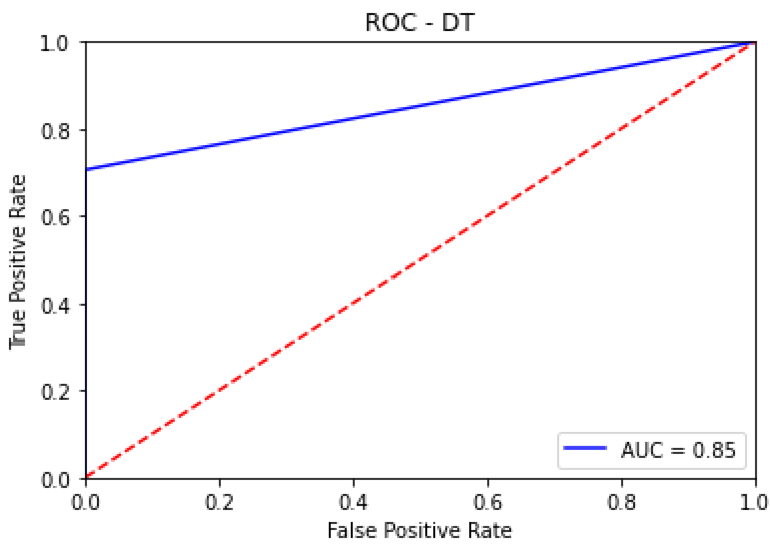
With Such a good precision and hence F1-Score, Random Forest comes out to be better as expected.

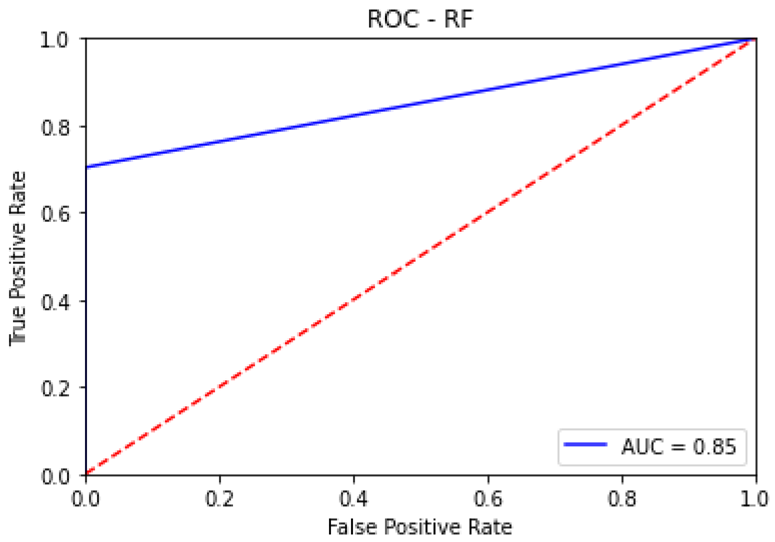
In [28]:
```
# visualising confusion matrix - DT


disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_dt)
disp.plot()
plt.title('Confusion Matrix - DT')
plt.show()

# visualising confusion matrix - RF
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_rf)
disp.plot()
plt.title('Confusion Matrix - RF')
plt.show()
```



In [29]:
```
# 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()
```






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

# CONCLUSION

The equality in accuracy between Random Forest and Decision Tree models notwithstanding, the heightened precision of Random Forest becomes pivotal, especially in fraud detection scenarios. Precision assumes greater significance here as correctly identifying fraud transactions is paramount. Leaving legitimate transactions undetected or falsely flagging innocent ones can have grave consequences. Hence, the preference for Random Forest and Decision Tree algorithms over others stems from their ability to prioritize precision, crucial for accurately distinguishing fraudulent activities from genuine ones.

I selected this model due to the highly unbalanced dataset, where authorized cases vastly outnumber fraud cases at a ratio of 99.87 to 0.13. Random Forest constructs multiple decision trees, which aids the model in comprehending the data more effectively despite its complexity, as decision trees make binary decisions.

Although models such as XGBoost, Bagging, Artificial Neural Networks (ANN), and Logistic Regression may achieve high accuracy, they often fall short in achieving satisfactory precision and recall values.

What are the key factors that predict fraudulent customer?

1. The source of request is secured or not ?
2. Is the name of organisation asking for money is legit or not ?
3. Transaction history of vendors.

What preventive measures should be taken when a company updates its infrastructure?

1. Employ certified applications exclusively for updates.
2. Utilize secure browsing practices, prioritizing reputable websites.
3. Employ secure internet connections, such as VPNs.
4. Ensure regular updates for mobile and laptop security.

5. Exercise caution and refrain from responding to unsolicited communications via calls, SMS, or emails.
6. Promptly contact your bank if you suspect any security breaches or fraudulent activity.

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

1. The bank sends electronic statements.
2. Customers monitor their account activity regularly.
3. It's essential to maintain a record of all payments made.