Import Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
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
```

Import Data

In [2]:

```
df = pd.read_csv('Fraud.csv')
df
```

Out[2]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDe
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M19797871
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M20442822
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C5532640
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C389970
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M12307017
...	...	...	...	...	...	...	
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.00	C77691929
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.00	C18818418
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.00	C13651258
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.00	C20803885
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.00	C87322111

6362620 rows × 11 columns

Data Understanding

In [3]:

```
df.shape
```

Out[3]:

```
(6362620, 11)
```

In [4]:

```
df.isnull().sum()
```

Out[4]:

```
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 [5]:

```
df.dtypes
```

Out[5]:

```
step                int64
type               object
amount            float64
nameOrig           object
oldbalanceOrg     float64
newbalanceOrig    float64
nameDest           object
oldbalanceDest    float64
newbalanceDest    float64
isFraud             int64
isFlaggedFraud      int64
dtype: object
```

In [6]:

```
df.head(100)
```

Out[6]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	old
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	
...	...	...	...	...	...	...	...	
95	1	TRANSFER	710544.77	C835773569	0.0	0.00	C1359044626	
96	1	TRANSFER	581294.26	C843299092	0.0	0.00	C1590550415	
97	1	TRANSFER	11996.58	C605982374	0.0	0.00	C1225616405	
98	1	PAYMENT	2875.10	C1412322831	15443.0	12567.90	M1651262695	
99	1	PAYMENT	8586.98	C1305004711	3763.0	0.00	M494077446	

100 rows × 11 columns

In [7]:

```
df.tail(100)
```

Out[7]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDe
6362520	735	TRANSFER	417103.68	C336307904	417103.68	0.0	C115591528
6362521	735	CASH_OUT	417103.68	C1450763584	417103.68	0.0	C137783051
6362522	735	TRANSFER	92735.71	C1351323617	92735.71	0.0	C41372255
6362523	735	CASH_OUT	92735.71	C786761311	92735.71	0.0	C57018881
6362524	735	TRANSFER	123146.28	C1625883009	123146.28	0.0	C91815439
...	...	...	...	...	...	...	
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.0	C77691929
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.0	C188184183
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.0	C136512589
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.0	C208038851
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.0	C87322118

100 rows × 11 columns

Data Preparation

In [8]:

```
df.isnull().values.any()
```

Out[8]:

```
False
```

In [9]:

```
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
```

In [10]:

```
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 %
```

Here percentage of legit transactions are 99.8% where as percentage of fraud transactions are 0.13%

So, methods for imbalanced data are decision tree and random forest

In [11]:

```python
X = df[df['nameDest'].str.contains('M')]
X.head()
```

Out[11]:

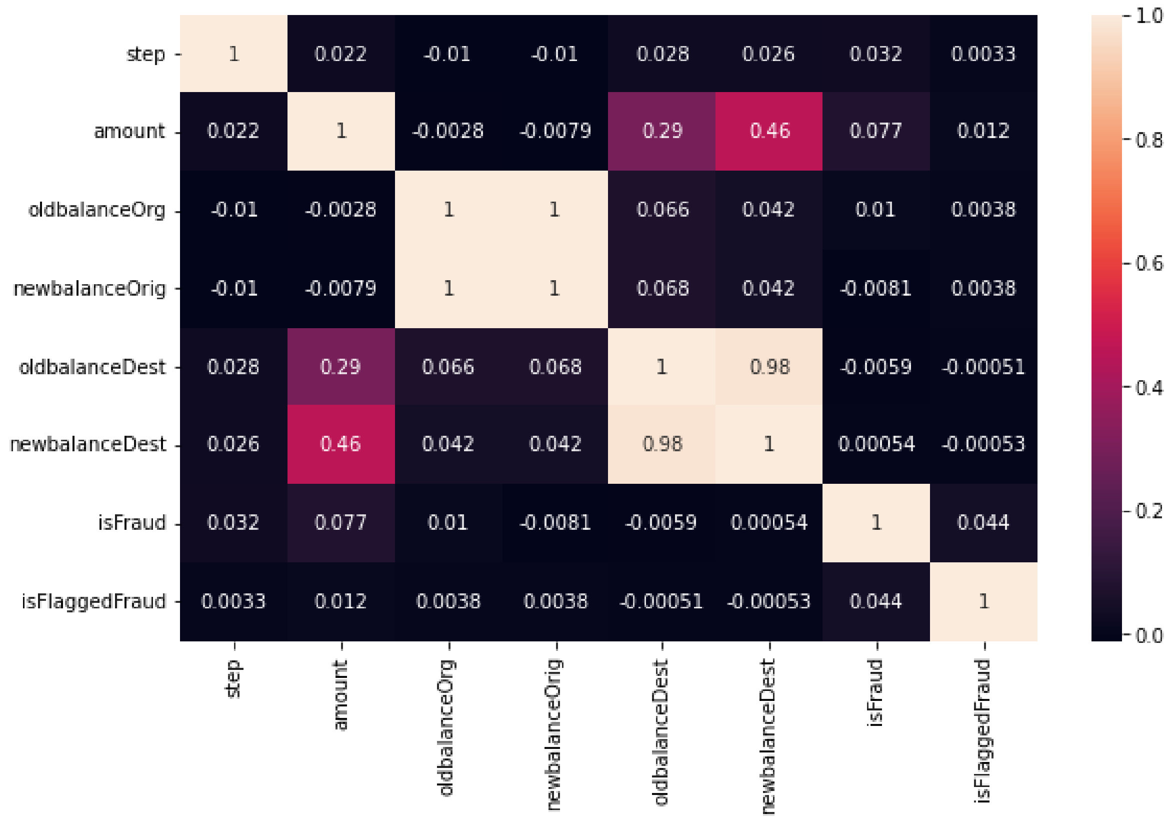
	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbala
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	
5	1	PAYMENT	7817.71	C90045638	53860.0	46042.29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

Data Visualization

Correlation Heatmap

In [12]:

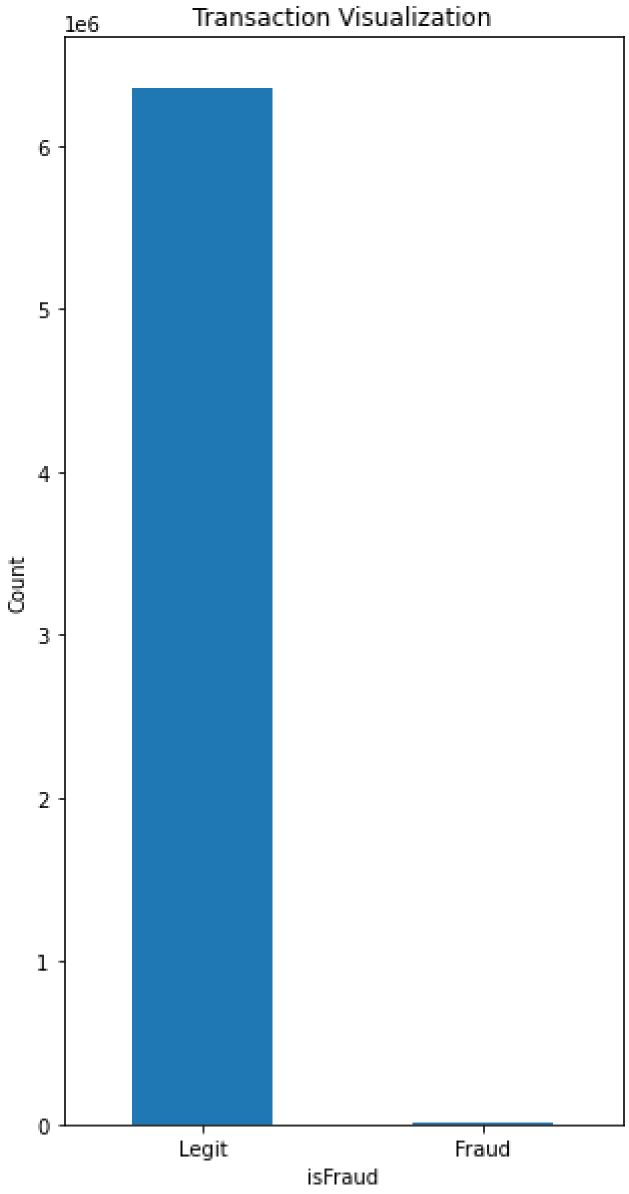
```python
corr = df.corr()

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



Number of legit and Fraud Transactions

In [13]:

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



In [14]:

```
new_df = df.copy()
new_df.head()
```

Out[14]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldba
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]:

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

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

In [16]:

```
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 [17]:

```
new_df.head()
```

Out[17]:

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

MultiColinearity

In [18]:

```
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[18]:

	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

In [19]:

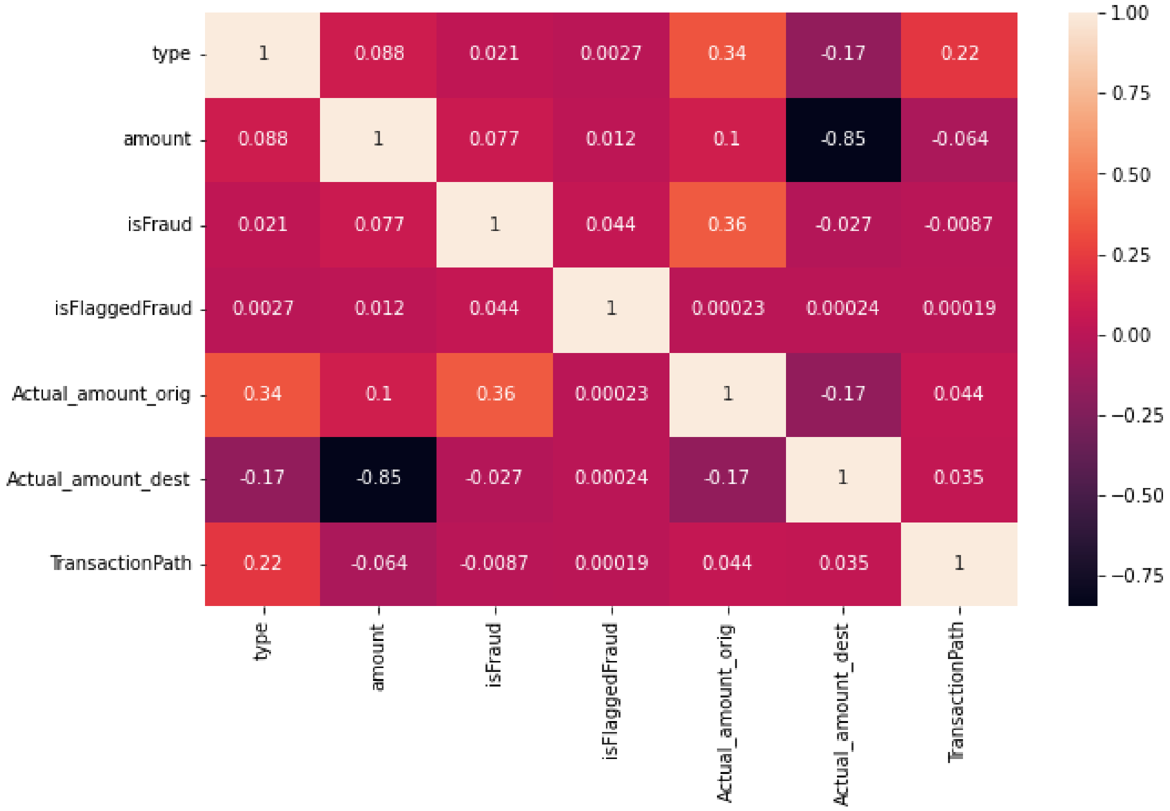
```
new_df['Actual_amount_orig'] = new_df.apply(lambda x: x['oldbalanceOrg'] - x['newbalanceOri
new_df['Actual_amount_dest'] = new_df.apply(lambda x: x['oldbalanceDest'] - x['newbalanceDe
new_df['TransactionPath'] = new_df.apply(lambda x: x['nameOrig'] + x['nameDest'],axis=1)


new_df = new_df.drop(['oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest','s

calc_vif(new_df)
```

Out[19]:

	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 [20]:

```
corr=new_df.corr()

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



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 Buidling

In [21]:

```
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 Amount

In [22]:

```
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

In [23]:

```
(X_train, X_test, Y_train, Y_test) = train_test_split(X, Y, test_size= 0.3, random_state= 4

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 Training

Decison Tree

In [24]:

```
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
```

Model Evaluation

In [26]:

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

```
Decision Tree Score:  99.92319725731433
Random Forest Score:  99.95871721607347
```

In [27]:

```
# 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: 1718
False Positives: 749
True Negatives: 1905602
False Negatives: 717
----------------------------------------------------------------------------
------------
TP,FP,TN,FN - Random Forest
True Positives: 1711
False Positives: 64
True Negatives: 1906287
False Negatives: 724
```

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)

In [28]:

```
# 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
[[1905602     749]
 [    717    1718]]
----------------------------------------------------------------------------
------------
Confusion Matrix - Random Forest
[[1906287      64]
 [    724    1711]]
```

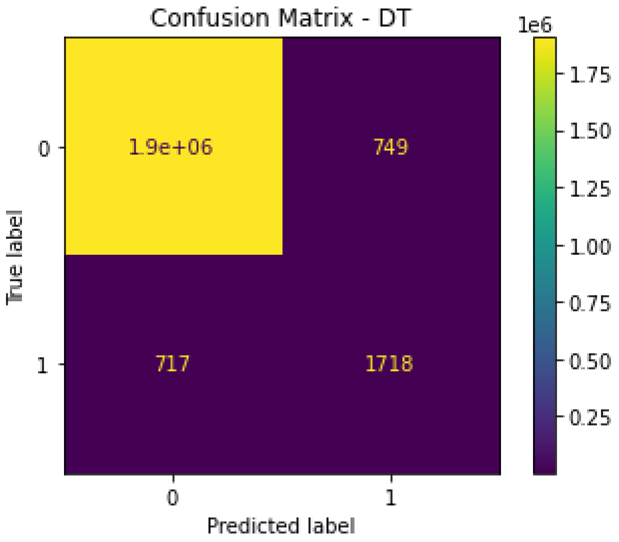
In [29]:

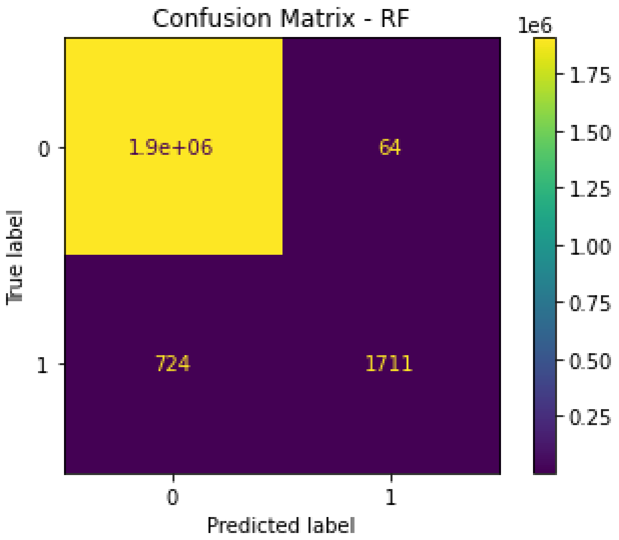
```
# classification report - DT

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
```

In [30]:

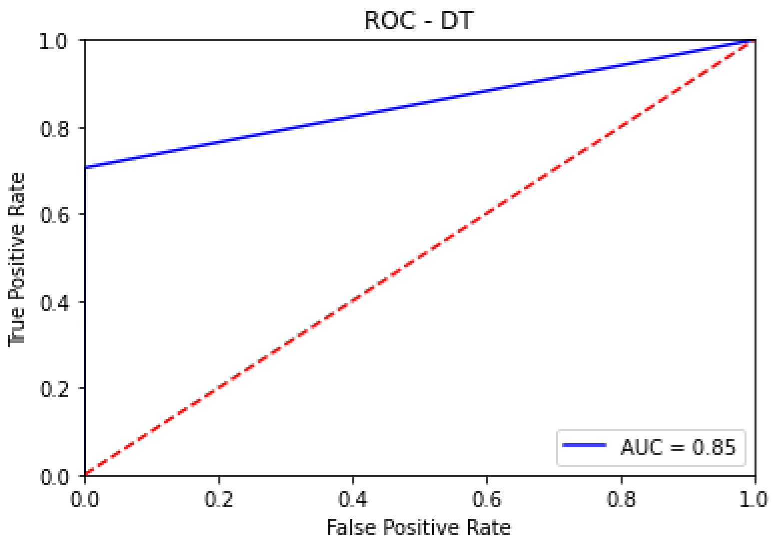
```
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_dt)
disp.plot()
plt.title('Confusion Matrix - DT')
plt.show()

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

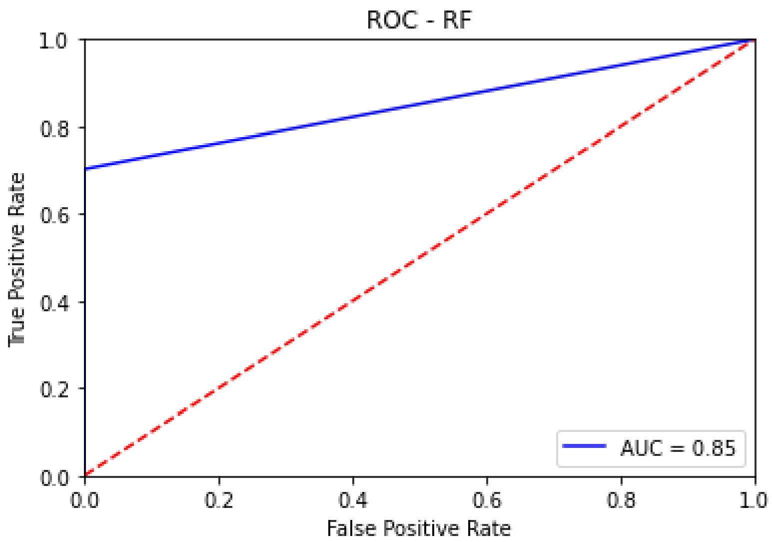




In [31]:

```
# 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()
```





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 2 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.

Also the reason I have chosen this model is because of highly unbalanced dataset (Legit: Fraud :: 99.87:0.13). Random forest makes multiple decision trees which makes it easier (although time taking) for model to understand the data in a simpler way since Decision Tree makes decisions in a boolean way.

Models like XGBoost, Bagging, ANN, and Logistic Regression may give good accuracy but they won't give good precision and recall values.

Q-What are the key factors that predict fraudulent customer?

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

Q-What kind of prevention should be adopted while company update its infrastructure?

1.Use smart vertified apps only.

2.Browse through secured websites.

3.Use secured internet connections (USE VPN).

4.Keep your mobile and laptop security updated.

5.Don't respond to unsolicited calls/SMS(s/E-mails.

6.If you feel like you have been tricked or security compromised, contact your bank immidiately.

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

1.Bank sending E-statements.

2.Customers keeping a check of their account activity.

3.Always keep a log of your payments.