Contents

I. Introduction

Motivation

From traditional to emerging sectors, there is not one single business that is fully immune from fraud. Some studies show that fraud of various kinds could cost businesses 1%-1.75% of their annual sales, this translates to around $200 billion a year!

As one of the most common types of fraudulent activities, digital transaction fraud impacts around 127 million people, or approximately $8 billion in attempted fraudulent charges on Americans. Thus imperative for financial companies to understand the characteristics of a fraudulent transactions and develop predictive models accordingly to flag down potentially risky activities for fraud prevention.

The Dataset

The dataset used in this project is available on kaggle: Synthetic Financial Datasets For Fraud Detection

Context Develop a model for predicting fraudulent transactions for a financial company and use insights from the model to develop an actionable plan. Data for the case is available in CSV format having 6362620 rows and 10 columns.

Content Data for the case is available in CSV format having 6362620 rows and 10 columns.

Data Dictionary:

step - maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).

type - CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.

amount - amount of the transaction in local currency.

nameOrig - customer who started the transaction

oldbalanceOrg - initial balance before the transaction newbalanceOrig - new balance after the transaction

nameDest - customer who is the recipient of the transaction

oldbalanceDest - initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).

newbalanceDest - new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).

isFraud - This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers accounts and try to empty the funds by transferring to another account and then cashing out of the system.

isFlaggedFraud - The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction.

This dataset is presently only one of four on Kaggle with information on the rising risk of digital financial fraud, emphasizing the difficulty in obtaining such data. The main technical challenge it poses to predicting fraud is the highly imbalanced distribution between positive and negative classes in 6 million rows of data.

Objectives

The main goal of this project is to come up with a model that can detect and classify fraudulent transactions effectively. Since the dataset closely resembles real-life financial transactions, if implemented in a real-world production environment, it could help mitigate fraud by detecting and classifying fraudulent activity. Helping to reduce the impact of fraud on businesses and account users, as well as helping to preserve the trust for all actors.

Another goal is to highlight and showcase the power of big-data, data science and machine learning. And how organizations can capitalize on their data to either extract valuable insights, create competitive advantages, or in the context of this study, minimize losses and preserve customer trust.

II. Implementation

In [91]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [92]:
```
# Read the data
df=pd.read_csv("Fraud.csv")
```

```
df.shape
```

Out[92]: (6362620, 11)

We've read the dataset, we can see that this is a quite large dataset with 6.3 million rows.

In [93]:
```
# Get head of the data
df.head(200)
```

Out[93]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameD
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701
...	...	...	...	...	...	...	
195	1	CASH_OUT	210370.09	C2121995675	0.0	0.00	C1170794
196	1	CASH_OUT	36437.06	C2120063568	0.0	0.00	C1740000
197	1	CASH_OUT	82691.56	C1620409359	0.0	0.00	C248609
198	1	CASH_OUT	338767.10	C691691381	0.0	0.00	C453211
199	1	CASH_OUT	187728.59	C264978436	0.0	0.00	C1360767

200 rows × 11 columns

In [94]:
```
# Check for null values
df.isnull().values.any()
```

Out[94]: False

In [95]:
```
# Getting information about data
df.info()
```

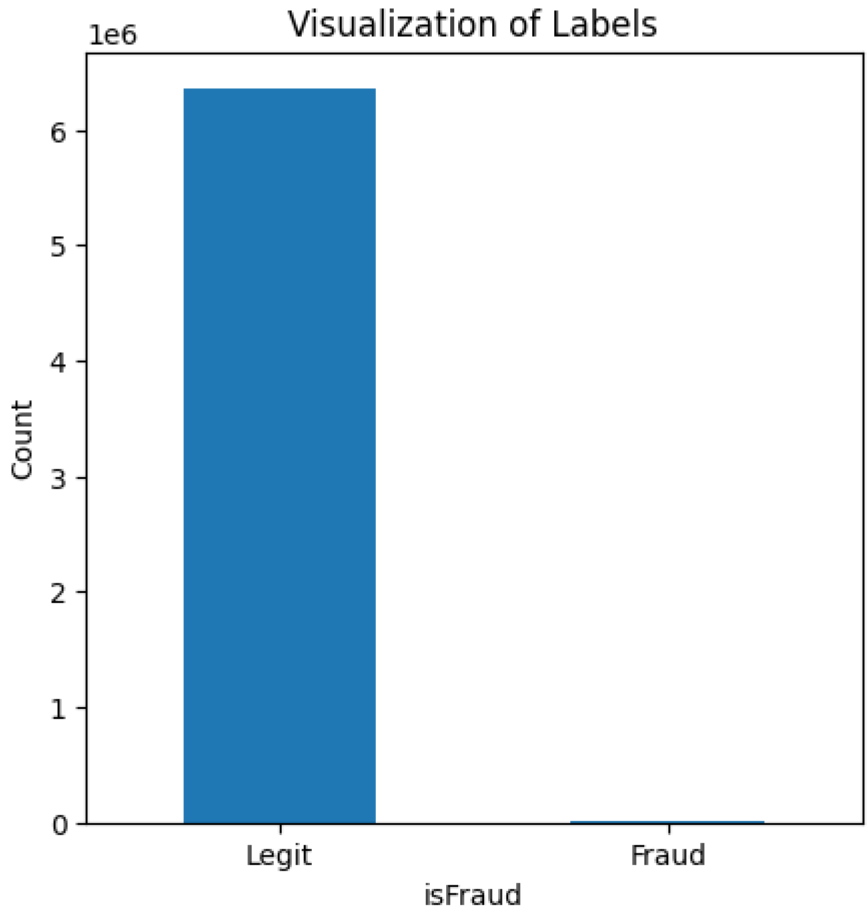
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype  
---  ------          -----  
 0   step            int64  
 1   type            object 
 2   amount          float64
 3   nameOrig        object 
 4   oldbalanceOrg   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
```

We can see this is a quite large dataset, and it does not contian NULL values. The dataset is over 500MB in size.

In [96]:
```
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 %
```

In [97]:
```
plt.figure(figsize=(5,5))
labels = ["Legit", "Fraud"]
count_classes = df.value_counts(df['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()
```



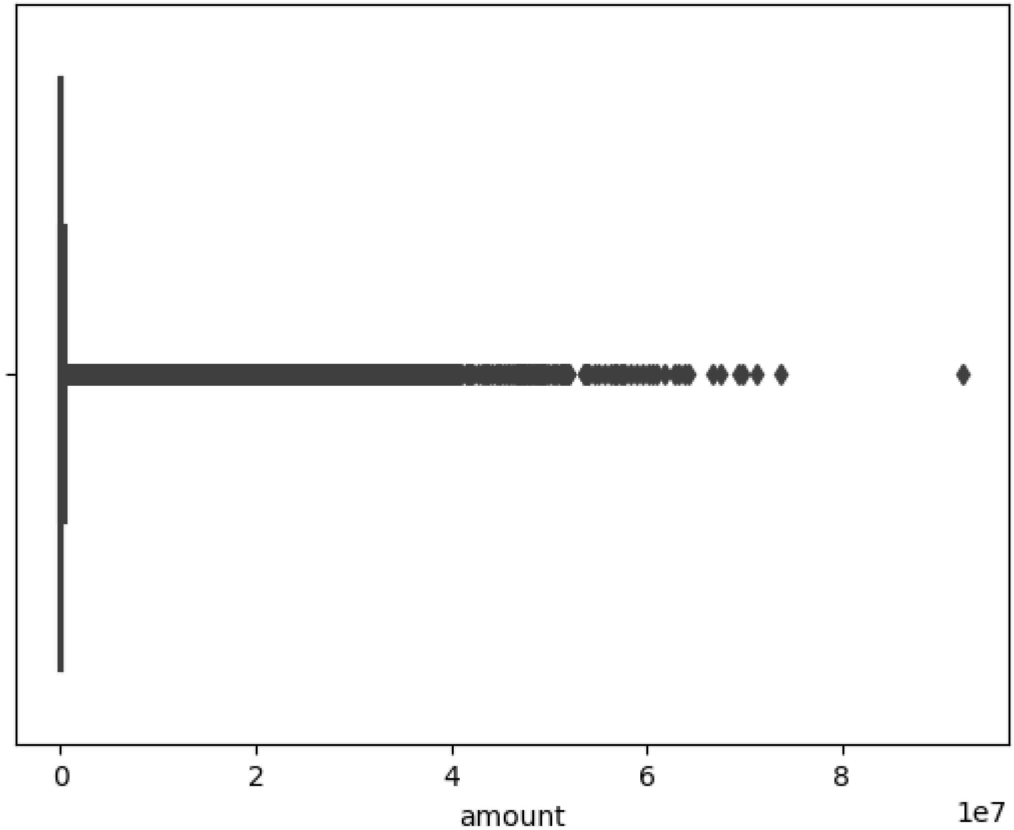
These results prove that this is a highly unbalanced data as Percentage of Legit transactions= 99.87 % and Percentage of Fraud transactions= 0.13 %.

From the summary above we can see that this is a very umbalance dataset, where the target variable contains two classes, where 1 corresponds to fraudulent transactions, and 0 to legitimate transactions. We can see that it is highly umbalanced since fraudulent transactions only represent a 0.13% out of 100% of the data.

This is a problem, since under this circumstances even a DummyClassifier without any training could achieve a extremely high accuracy. To overcome this problem, I will proceed to oversample the minority class, but will do this only after splitting the dataset into training and test sets.

In [98]:
```
sns.boxplot(x=df["amount"])
```

Out[98]:
```
<AxesSubplot: xlabel='amount'>
```

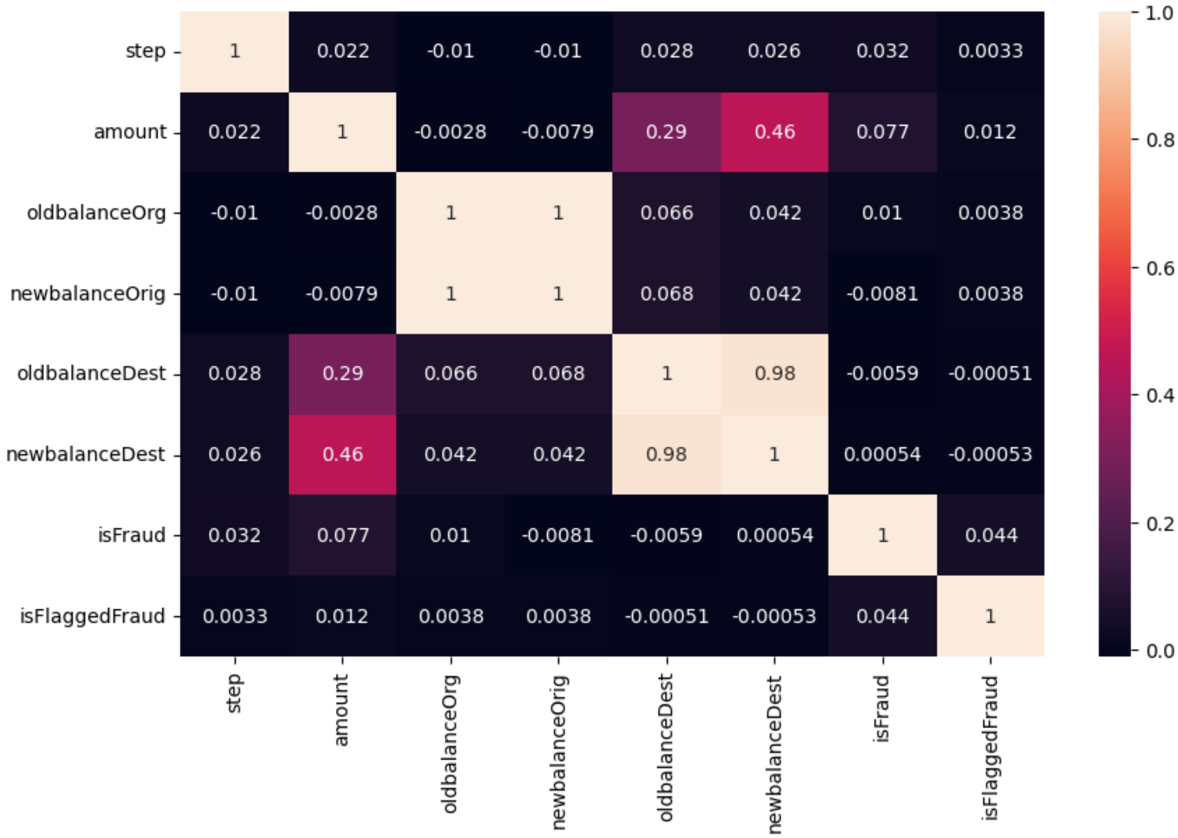


Here we see the boxplot of the distribution of the amount column, where we can spot a large number of outliers.

In [99]:
```
corr=df.corr()

plt.figure(figsize=(10,6))

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

```
/var/folders/r7/v0pshjtd6_gdlcgm_z9_4k1h0000gn/T/ipykernel_4104/1773209120.
py:1: FutureWarning: The default value of numeric_only in DataFrame.corr is
deprecated. In a future version, it will default to False. Select only vali
d columns or specify the value of numeric_only to silence this warning.
  corr=df.corr()
```

Out[99]: `<AxesSubplot: >`



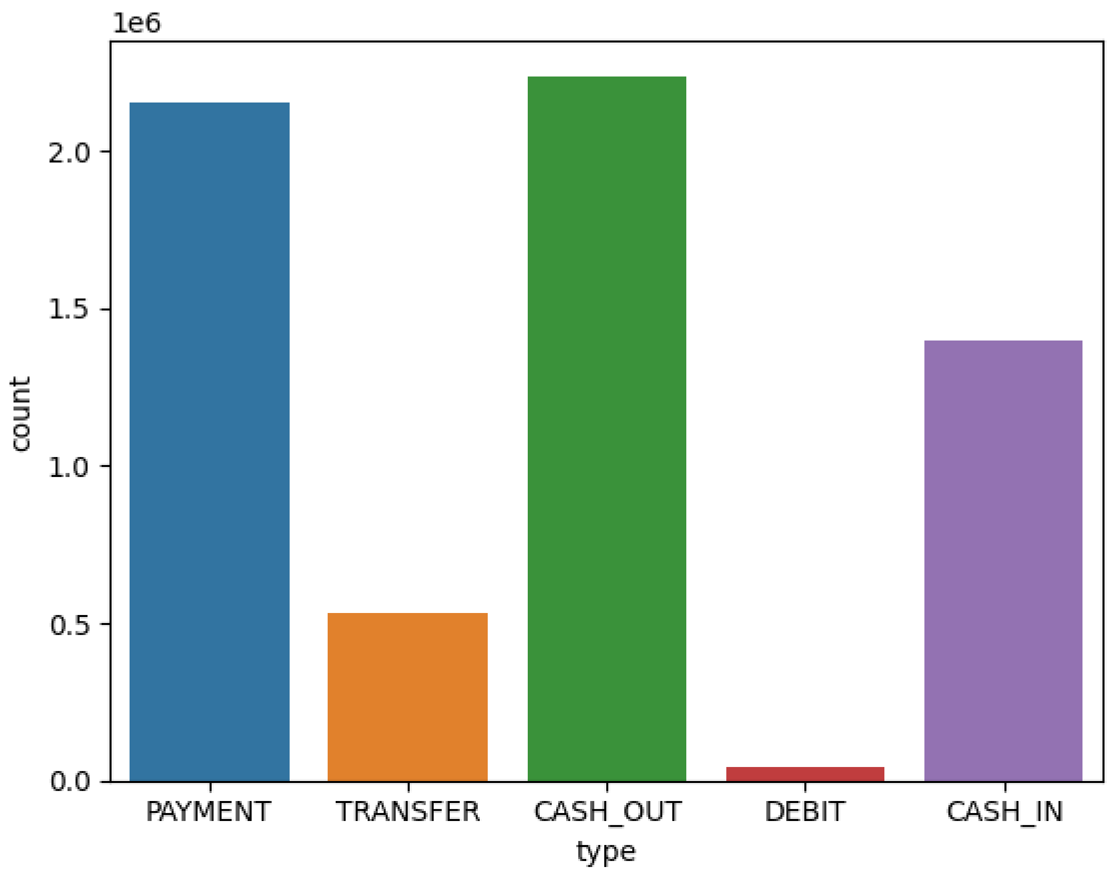
The correlation matrix helps us understand which the strength of the relationship between different variables, specially correltion of the the rest of the features and our target variable 'isFraud'. This values will help us make better informed decisions when selecting the variables to use when training our model.

As this dataset is very large, we are interested in architecting a model that not only is good at predicting the desired outcomes, but that also has a good performance.

DISTRIBUTION OF TYEPES OF TRANSACTIONS

In [100…
```
sns.countplot(x = df['type'])
```

Out[100]:
```
<AxesSubplot: xlabel='type', ylabel='count'>
```



Here we can see the distribution of the different types of transactions that exist in the dataset. being Payment, and cash_out the two larget groups.

In [101…
```
print("Total Unique Values in nameOrig", df['nameOrig'].nunique())
print("Total Unique Values in nameDest", df['nameDest'].nunique())
```

```
Total Unique Values in nameOrig 6353307
Total Unique Values in nameDest 2722362
```

In [102…
```
#creating a copy of original dataset to train and test models
new_df=df.copy()
```

Multicolinearity analysis

In [103…
```
# Checking how many attributes are dtype: object
objList = new_df.select_dtypes(include = "object").columns
print (objList)
```

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

There are three columns with object data type. We need to encode them in order to assess multicolinearity.

In [104…
```
#Label Encoding for object to numeric conversion
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            int64  
 2   amount          float64
 3   nameOrig        int64  
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        int64  
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64  
 10  isFlaggedFraud  int64  
dtypes: float64(5), int64(6)
memory usage: 534.0 MB
None
```

In [105…

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

    return(vif)

calc_vif(new_df)
```

Out[105]:

	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 a very high VIF therefore they are highly correlated. This is true for oldbalanceDest and newbalanceDest as well. Also nameDest is connected to nameOrig.

Therefore we will create a new feature for each group, that will unify the two. After that we will drop the indivual ones.

In [106…

```
new_df['Actual_amount_orig'] = new_df.apply(lambda x: x['oldbalanceOrg'] - x
new_df['Actual_amount_dest'] = new_df.apply(lambda x: x['oldbalanceDest'] -
new_df['TransactionPath'] = new_df.apply(lambda x: x['nameOrig'] + x['nameDe

new_df = new_df.drop(['oldbalanceOrg','newbalanceOrig','oldbalanceDest','new
new_df.head()
```

Out[106]:

	type	amount	isFraud	isFlaggedFraud	Actual_amount_orig	Actual_amount_dest	Tran
0	3	9839.64	0	0	9839.64	0.0	
1	3	1864.28	0	0	1864.28	0.0	
2	4	181.00	1	0	181.00	0.0	
3	1	181.00	1	0	181.00	21182.0	
4	3	11668.14	0	0	11668.14	0.0	

In [107…

```
calc_vif(new_df)
```

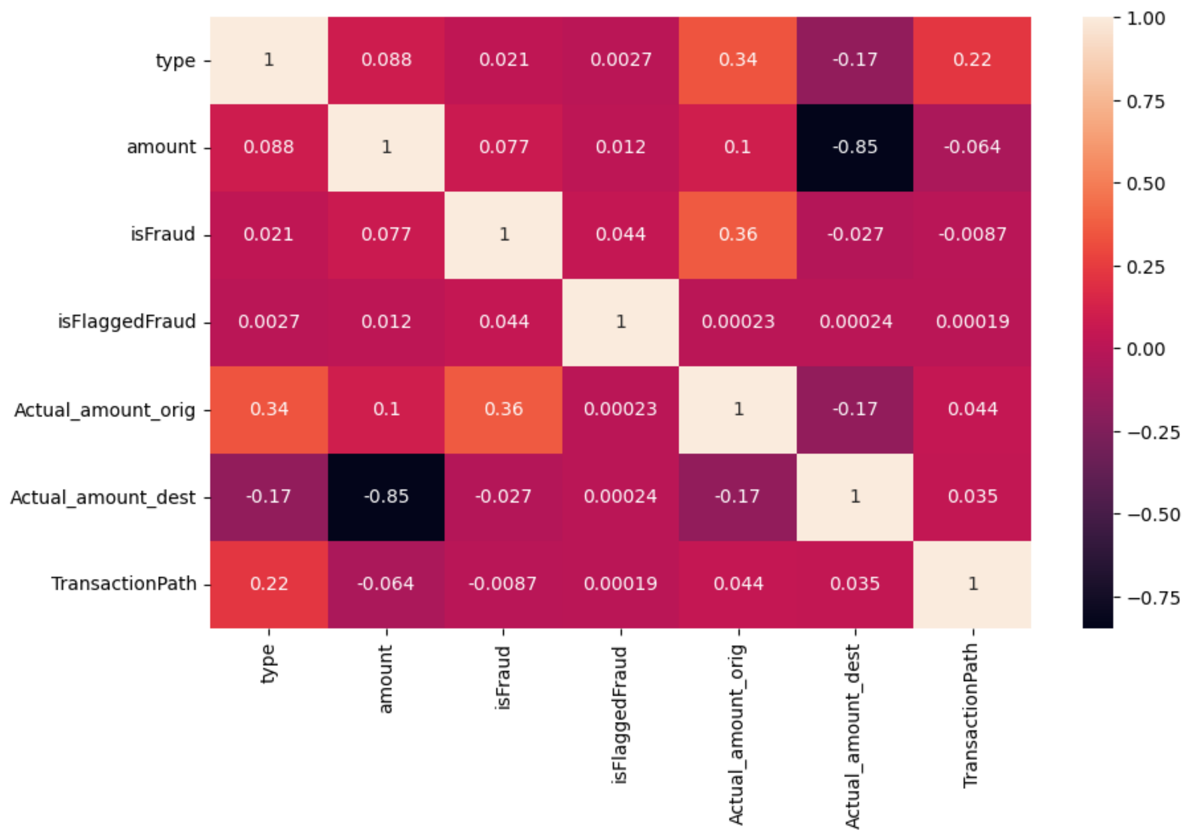
Out[107]:

	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 [108…

```
corr=new_df.corr()

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

Out[108]: <AxesSubplot: >



Building The model

In [109…

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
import itertools
```

```
from collections import Counter
import sklearn.metrics as metrics
from sklearn.metrics import classification_report, confusion_matrix, Confusi
```

NORMALIZING (SCALING) AMOUNT

In [110…

```
print(new_df['amount'].head())
```

```
0     9839.64
1     1864.28
2      181.00
3      181.00
4    11668.14
Name: amount, dtype: float64
```

In [111…

```
# Perform Scaling
scaler = StandardScaler()
new_df["NormalizedAmount"] = scaler.fit_transform(new_df["amount"].values.re
new_df.drop(["amount"], inplace= True, axis= 1)
print(new_df.head())
```

```
   type  isFraud  isFlaggedFraud  Actual_amount_orig  Actual_amount_dest  \
0     3        0               0             9839.64                 0.0
1     3        0               0             1864.28                 0.0
2     4        1               0              181.00                 0.0
3     1        1               0              181.00             21182.0
4     3        0               0            11668.14                 0.0

   TransactionPath  NormalizedAmount
0        2419963.0         -0.281560
1        3922922.0         -0.294767
2        1441841.0         -0.297555
3        6219958.0         -0.297555
4        4274900.0         -0.278532
```

TRAIN-TEST SPLIT

In [112…

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

# 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)

from sklearn.dummy import DummyClassifier

dummy = DummyClassifier(strategy='constant', constant=1).fit(X_train, y_trai
dummy_pred = dummy.predict(X_test)

print('Test score: ', metrics.accuracy_score(y_test, dummy_pred))
```

```
Shape of X_train:  (4453834, 6)
Shape of X_test:  (1908786, 6)
Test score:  0.0012756799347857749
```

Now that we have formatted our data for training, and we've already splitted our data into training and test sets. we will proceed to oversample the minority class.

In [113…
```
from sklearn.utils import resample

# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)

# separate minority and majority classes
not_fraud = X[X.isFraud == 0]
fraud = X[X.isFraud == 1]

print('Counts before oversampling minority class: \n')
print(X.isFraud.value_counts())
# upsample minority
fraud_upsampled = resample(fraud,
                          replace=True, # sample with replacement
                          n_samples=len(not_fraud), # match number in majori
                          random_state=27) # reproducible results

# combine majority and upsampled minority
upsampled = pd.concat([not_fraud, fraud_upsampled])

print('\n\nCounts after oversampling:')
# check new class counts
upsampled.isFraud.value_counts()
```

```
Counts before oversampling minority class:

0    4448056
1       5778
Name: isFraud, dtype: int64


Counts after oversampling:
```

Out[113]:
```
0    4448056
1    4448056
Name: isFraud, dtype: int64
```

After implementing the oversampling strategy, we can see that now the two classes 0 and 1 for the isFraud target variable are balance each with representing 50% of the training data.

MODEL TRAINING

In [114…
```
# LOGISTIC REGRESSOR
y_train = upsampled.isFraud
X_train = upsampled.drop('isFraud', axis=1)

logistic_regressor = LogisticRegression(solver='liblinear', random_state=0)
logistic_regressor.fit(X_train, y_train)
```

```
y_pred_lr = logistic_regressor.predict(X_test)
logistic_regressor_score = logistic_regressor.score(X_test, y_test) * 100
```

III. Evaluation

In [115…
```
# Print scores of our classifiers
print("Logistic Regressor Score: ", logistic_regressor_score)
```

```
Logistic Regressor Score:  95.07100324499447
```

Here we see that our model has a 95% accuracy on the test data.

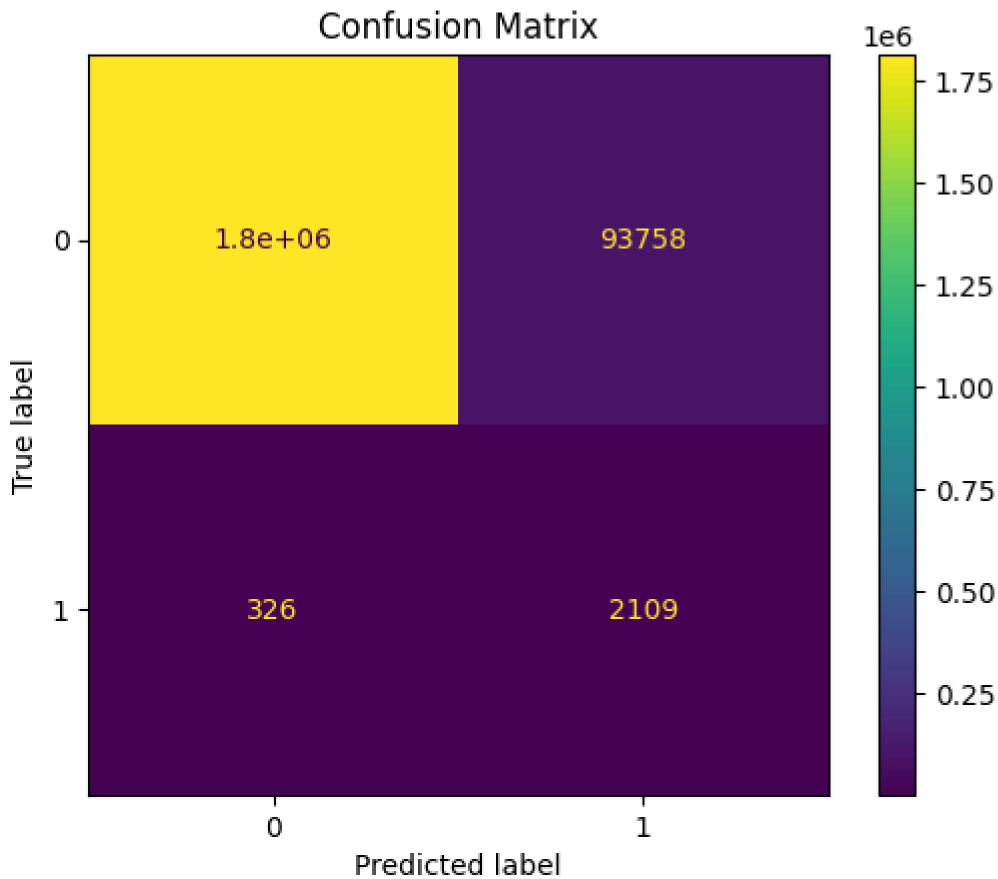
In [116…
```
# classification report
classification_report_lr = classification_report(Y_test, Y_pred_lr)
print("Classification Report",classification_report_lr)
```

```
Classification Report               precision    recall  f1-score   support

           0       1.00      0.95      0.97   1906351
           1       0.02      0.87      0.04      2435

    accuracy                           0.95   1908786
   macro avg       0.51      0.91      0.51   1908786
weighted avg       1.00      0.95      0.97   1908786
```

In [117…
```
# confusion matrix - Linear Regressor
confusion_matrix_lr = confusion_matrix(Y_test, Y_pred_lr.round())
print("Confusion Matrix")
print(confusion_matrix_lr)
```

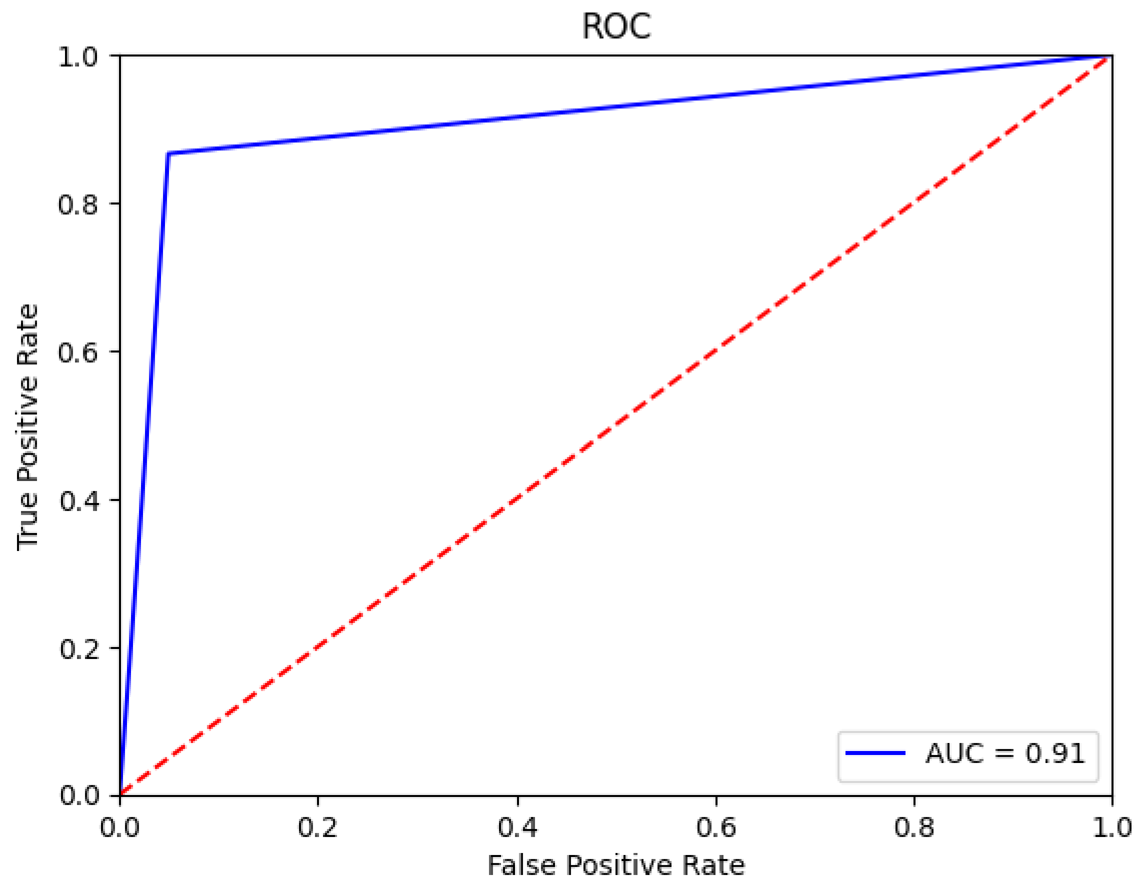
```
Confusion Matrix
[[1812593   93758]
 [    326    2109]]
```

In [118…
```
# visualising confusion matrix - DT
disp = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_lr)
disp.plot()
plt.title('Confusion Matrix')
plt.show()
```



In [119…

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

fpr, tpr, threshold = metrics.roc_curve(y_test, y_pred_lr)
roc_auc = metrics.auc(fpr, tpr)

plt.title('ROC')
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 refers to the area under the curve, where an excellent model has AUC near to the 1 which means it has a good measure of separability. The closer a model's AUC comes to 0 the worse measure of separability between classes.

Our model scores an AUC of 0.91 which is considered to be a great score.

CONCLUSION

Fraud is a big problem in todays world, and machine learning can help mitigate the negative impact it creates on businesses.

Here we have implemented a model that performs really well, further exploration with other models such as random forests, boosting machines or neural networks could be interesting, to compare the performance of those models against our regresor.

One of the greatest advantages of these models is that, once a final architecture has been chosen, the weights of the models can be saved, and the model can be deployed to any financial or e-commerce back-end, and it can help classsify legitimate and fraudulent transactions. What's more, the data produced on these platforms can be collected, processed and tidied to feed it back to the model periodically, so it can continue learning.

Fraudsters are always finding innovative ways to abuse and break services for their own interests. And Machine learning models can identify and learn these patterns.

In [ ]: