In [1]:

```python
# Open the image
from IPython.display import Image
Image(filename="C:/Users/rahul/Downloads/image.jpg")
```

Out[1]:

Online payment fraud detection

We are living in the digital world where people started approaching towards current technologies. They make our work easy and its reliable.

Online payment is one of the scenarios where people started using in recent years. Just one click one tap makes our work easier and faster. As much as we know about the merits of online payment, there are fraudsters who try to loot money from people with different techniques.

With the increase of online payment now-a-days, the online payment fraud has also been rising and it's actually a major concern among the people who are not aware of the current technologies.

Let's analyze about the online payment fraud detection dataset taken from Kaggle and provide insights on this!!

Columns in 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

newbalanceOrig: balance after the transaction

nameDest: recipient of the transaction

oldbalanceDest: initial balance of recipient before the transaction

newbalanceDest: the new balance of recipient after the transaction

isFraud: fraud transaction

Importing Libraries and Datasets

The libraries used are :

Pandas: This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go. Seaborn/Matplotlib: For data visualization. Numpy: Numpy arrays are very fast and can perform large computations in a very short time.

In [2]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```python
df = pd.read_csv("onlinefraud.csv")
df
```

Out[3]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nam
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M19797
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M20442
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C5532
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C389
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M12307
...	...	...	...	...	...	...	
1048570	95	CASH_OUT	132557.35	C1179511630	479803.00	347245.65	C4356
1048571	95	PAYMENT	9917.36	C1956161225	90545.00	80627.64	M6683
1048572	95	PAYMENT	14140.05	C2037964975	20545.00	6404.95	M13551
1048573	95	PAYMENT	10020.05	C1633237354	90605.00	80584.95	M19649
1048574	95	PAYMENT	11450.03	C1264356443	80584.95	69134.92	M6775

1048575 rows × 11 columns

In [4]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 11 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   nameOrig        1048575 non-null  object
 4   oldbalanceOrg   1048575 non-null  float64
 5   newbalanceOrig  1048575 non-null  float64
 6   nameDest        1048575 non-null  object
 7   oldbalanceDest  1048575 non-null  float64
 8   newbalanceDest  1048575 non-null  float64
 9   isFraud         1048575 non-null  int64
 10  isFlaggedFraud  1048575 non-null  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 88.0+ MB
```

In [5]:

```
# view the first few rows of the dataset
df.head()
```

Out[5]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	c
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 [6]:

```
# Drop the 'isFlaggedFraud' column
df.drop('isFlaggedFraud', axis=1, inplace=True)
```

To print the information of the data we can use data.info() command.

In [7]:

```
# view the data types and missing values in each column
df.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   nameOrig        1048575 non-null  object 
 4   oldbalanceOrg   1048575 non-null  float64
 5   newbalanceOrig  1048575 non-null  float64
 6   nameDest        1048575 non-null  object 
 7   oldbalanceDest  1048575 non-null  float64
 8   newbalanceDest  1048575 non-null  float64
 9   isFraud         1048575 non-null  int64  
dtypes: float64(5), int64(2), object(3)
memory usage: 80.0+ MB
```

Let’s see the mean, count , minimum and maximum values of the data

In [8]:

```
# view the summary statistics for each column
df.describe()
```

Out[8]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalar
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048
mean	2.696617e+01	1.586670e+05	8.740095e+05	8.938089e+05	9.781600e+05	1.114
std	1.562325e+01	2.649409e+05	2.971751e+06	3.008271e+06	2.296780e+06	2.416
min	1.000000e+00	1.000000e-01	0.000000e+00	0.000000e+00	0.000000e+00	0.000
25%	1.500000e+01	1.214907e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000
50%	2.000000e+01	7.634333e+04	1.600200e+04	0.000000e+00	1.263772e+05	2.182
75%	3.900000e+01	2.137619e+05	1.366420e+05	1.746000e+05	9.159235e+05	1.149
max	9.500000e+01	1.000000e+07	3.890000e+07	3.890000e+07	4.210000e+07	4.220

Data Visualization

In this section, we will try to understand and compare all columns.

Let's count the columns with different datatypes like Category, Integer, Float.

In [9]:

```
df.dtypes
```

Out[9]:

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

In [10]:

```
print(f"Number of categorical columns:", len(df.select_dtypes(include='object').columns))
print(f"Number of integer columns:", len(df.select_dtypes(include='int').columns))
print(f"Number of float columns:", len(df.select_dtypes(include='float').columns))
```

```
Number of categorical columns: 3
Number of integer columns: 2
Number of float columns: 5
```

In [11]:

```
# Now, let’s have a look at whether this dataset has any null values or not
df.isnull().sum()
```

Out[11]:

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
dtype: int64
```

In [12]:

```
df.isna().sum()
```

Out[12]:

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
dtype: int64
```

Its a good thing that there is no missing values!

In [13]:

```
# Exploring transaction type
df.type.value_counts()
```

Out[13]:

```
CASH_OUT    373641
PAYMENT     353873
CASH_IN     227130
TRANSFER     86753
DEBIT         7178
Name: type, dtype: int64
```

In [14]:

```
a= df.groupby("type").count()["amount"]
a
b= a.sort_values(ascending=False).index[0]
b
```

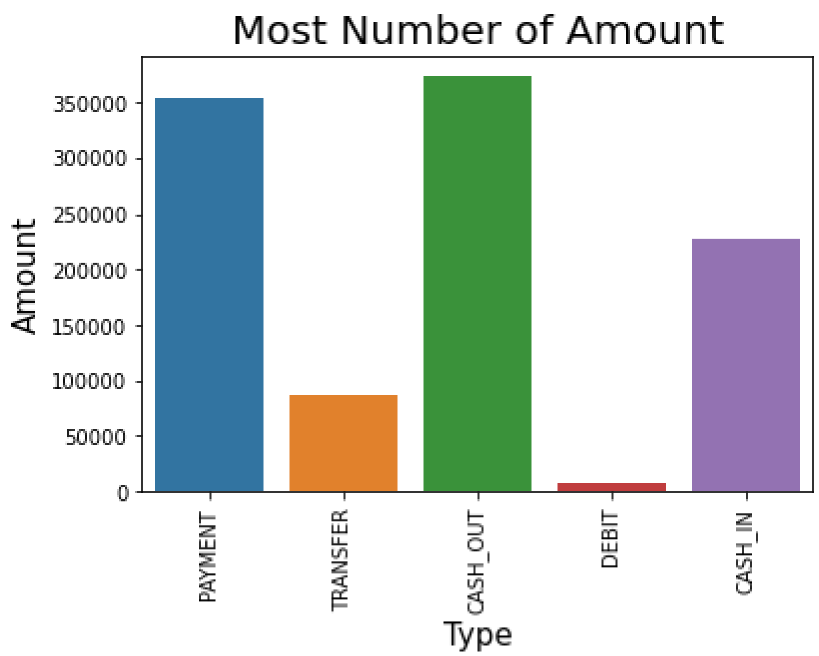
Out[14]:

```
'CASH_OUT'
```

Let’s see the count plot of the Payment type column using Seaborn library.

In [15]:

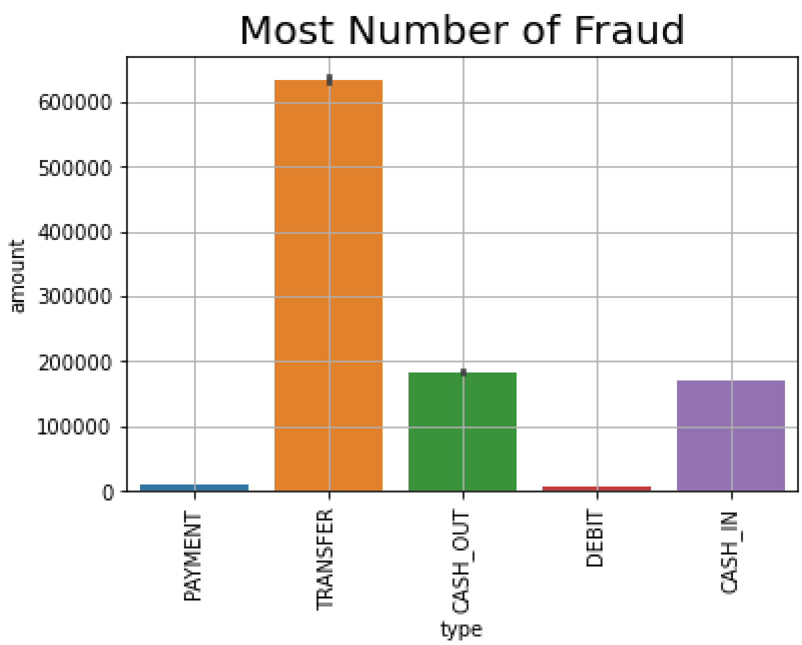
```
sns.countplot(x='type',data=df)
plt.title('Most Number of Amount',fontsize=20)
plt.xlabel('Type',fontsize=15)
plt.ylabel('Amount',fontsize=15)
plt.xticks(rotation=90)
plt.show()
```



We can also use the bar plot for analyzing Type and amount column simultaneously.

In [16]:

```
sns.barplot(x='type', y='amount', data=df)
plt.title('Most Number of Fraud',fontsize=20)
plt.xticks(rotation=90)
plt.grid()
plt.show()
```



Both the graph clearly shows that mostly the type cash_out and transfer are maximum in count and as well as in amount.

Let’s check the distribution of data among both the prediction values.

In [17]:

```
df['isFraud'].value_counts()
```

Out[17]:

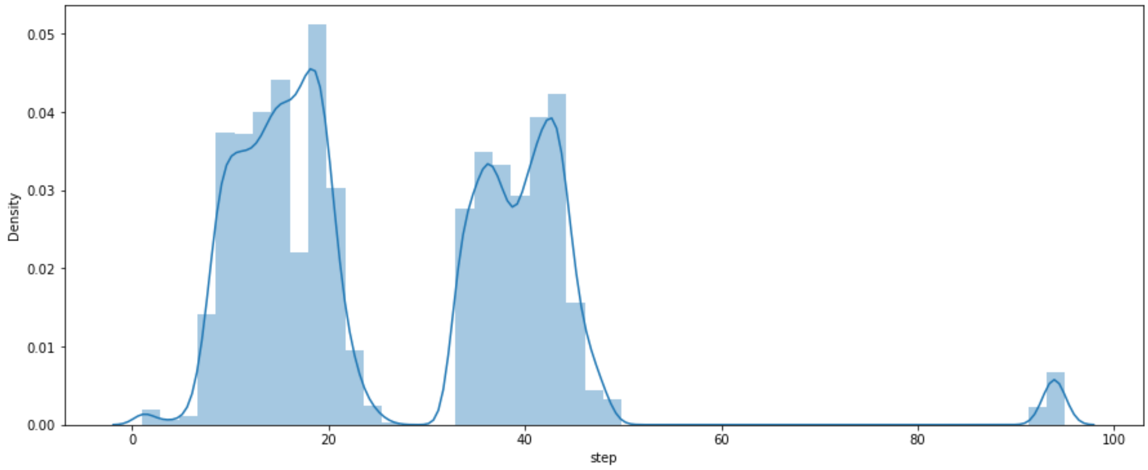
```
0    1047433
1       1142
Name: isFraud, dtype: int64
```

In [18]:

```
plt.figure(figsize=(15, 6))
sns.distplot(df["step"])
```

Out[18]:

<AxesSubplot:xlabel='step', ylabel='Density'>



The graph shows the maximum distribution among 20 to 50 of step.

Now, Let's find the correlation among different features using Heatmap.

In [19]:

```
corr = df.corr()
corr
```

Out[19]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newba
step	1.000000	-0.025996	-0.006780	-0.007180	-0.002251	
amount	-0.025996	1.000000	0.004864	-0.001133	0.215558	
oldbalanceOrg	-0.006780	0.004864	1.000000	0.999047	0.093305	
newbalanceOrig	-0.007180	-0.001133	0.999047	1.000000	0.095182	
oldbalanceDest	-0.002251	0.215558	0.093305	0.095182	1.000000	
newbalanceDest	-0.019503	0.311936	0.064049	0.063725	0.978403	
isFraud	0.045030	0.128862	0.003829	-0.009438	-0.007552	

In [20]:

```
plt.figure(figsize=(12, 6))
sns.heatmap(corr,annot=True,cmap="viridis")
```

Out[20]:

```
<AxesSubplot:>
```

Data Preprocessing

*Encoding of Type column

*Dropping irrelevant columns like nameOrig, nameDest

*Data Splitting

In [21]:

```
type_new = pd.get_dummies(df['type'], drop_first=True)
data_new = pd.concat([df, type_new], axis=1)
data_new.head()
```

Out[21]:

	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

Once we done with the encoding, now we can drop the irrelevant columns. For that, follow the code given below.

In [22]:

```
x= data_new.drop(['isFraud', 'type', 'nameOrig', 'nameDest'], axis=1)
y = data_new['isFraud']
```

Let's check the shape of extracted data.

In [23]:

```
x.shape, y.shape
```

Out[23]:

```
((1048575, 10), (1048575,))
```

In [24]:

```
x
```

Out[24]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
0	1	9839.64	170136.00	160296.36	0.00	0.00
1	1	1864.28	21249.00	19384.72	0.00	0.00
2	1	181.00	181.00	0.00	0.00	0.00
3	1	181.00	181.00	0.00	21182.00	0.00
4	1	11668.14	41554.00	29885.86	0.00	0.00
...	...	...	...	...	...	...
1048570	95	132557.35	479803.00	347245.65	484329.37	616886.72
1048571	95	9917.36	90545.00	80627.64	0.00	0.00
1048572	95	14140.05	20545.00	6404.95	0.00	0.00
1048573	95	10020.05	90605.00	80584.95	0.00	0.00
1048574	95	11450.03	80584.95	69134.92	0.00	0.00

1048575 rows × 10 columns

In [25]:

```
y
```

Out[25]:

```
0          0
1          0
2          1
3          1
4          0
          ..
1048570    0
1048571    0
1048572    0
1048573    0
1048574    0
Name: isFraud, Length: 1048575, dtype: int64
```

Now let's split the data into 2 parts : Training and Testing.

In [26]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.30,random_state=1)
```

In [27]:

```
# checking the new shapes
print("Shape of x_train: ", xtrain.shape)
print("Shape of x_test: ", xtest.shape)
print("Shape of y_train: ", ytrain.shape)
print("Shape of y_test: ", ytest.shape)
```

```
Shape of x_train:  (734002, 10)
Shape of x_test:  (314573, 10)
Shape of y_train:  (734002,)
Shape of y_test:  (314573,)
```

In [28]:

```
# performing standard scaling on the data for better fit

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
xtrain = sc.fit_transform(xtrain)
xtest = sc.transform(xtest)
```

In [29]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report,accuracy_score
```

In [30]:

```
models = []
accuracy = []

models.append(("logreg", LogisticRegression()))
models.append(("DT",DecisionTreeClassifier()))
models.append(("DT-e",DecisionTreeClassifier(criterion="entropy")))

for name, model in models:
    model.fit(xtrain,ytrain)
    ypred = model.predict(xtest)
    ac = accuracy_score(ytest,ypred)
    accuracy.append(ac)

arr = np.array(accuracy)
print(f"Avg Accuracy:- {arr.mean()}")
```

```
Avg Accuracy:- 0.9994691216347239
```

In [31]:

```
models
```

Out[31]:

```
[('logreg', LogisticRegression()),
 ('DT', DecisionTreeClassifier()),
 ('DT-e', DecisionTreeClassifier(criterion='entropy'))]
```

In [32]:

```
accuracy
```

Out[32]:

```
[0.9992052719082694, 0.999548594443897, 0.9996534985520055]
```

VotingClassifier

In [33]:

```
from sklearn.ensemble import VotingClassifier
vc = VotingClassifier(estimators=models,voting="soft")
vc.fit(xtrain,ytrain)
ypred = vc.predict(xtest)

train = vc.score(xtrain,ytrain)
test = vc.score(xtest,ytest)
print(f"Training Accuracy :- {train}\n Testing Accuracy:- {test}\n\n")

print(classification_report(ytest,ypred))
```

```
Training Accuracy :- 1.0
 Testing Accuracy:- 0.9996916455004085


              precision    recall  f1-score   support

           0       1.00      1.00      1.00    314246
           1       0.92      0.77      0.84       327

    accuracy                           1.00    314573
   macro avg       0.96      0.89      0.92    314573
weighted avg       1.00      1.00      1.00    314573
```

In [34]:

```
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
ac = accuracy_score(ytest,ypred)
cm = confusion_matrix(ytest,ypred)
cr = classification_report(ytest,ypred)

print(f"Accuracy :- {ac}\n {cm}\n {cr}")
```

```
Accuracy :- 0.9996916455004085
 [[314223     23]
 [    74    253]]
               precision    recall  f1-score   support

           0       1.00      1.00      1.00    314246
           1       0.92      0.77      0.84       327

    accuracy                           1.00    314573
   macro avg       0.96      0.89      0.92    314573
weighted avg       1.00      1.00      1.00    314573
```

In [35]:

```
def mymodel(model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)

    train = model.score(xtrain,ytrain)
    test = model.score(xtest,ytest)

    print(f"Training Accuracy:- {train}\n Testing Accuracy:- {test}")
    print(classification_report(ytest,ypred))
    return model
```

In [36]:

```
logreg = mymodel(LogisticRegression())
```

```
Training Accuracy:- 0.9992070866291918
 Testing Accuracy:- 0.9992052719082694
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    314246
           1       0.93      0.25      0.40       327

    accuracy                           1.00    314573
   macro avg       0.97      0.63      0.70    314573
weighted avg       1.00      1.00      1.00    314573
```

Conclusion

We have large number of records which are incorrectly flagged as 0. Incorrect flagging might have big impact in future if we don't calculate it properly as it might lead to increase in online payment fraud percentage as people rely more on online payment nowadays. The amount range usually fraudsters target is aroung 1-4 lakhs which is certainly a large sum. Fraudsters focus during cashout and transfer mode type transfer. Fraud is less likely/rare to happen during payment mode transfer though people are using online payment more. There is not much information taken from oldbalanceOrg,newbalanceOrig,nameDest,oldbalanceDest and newbalanceDest columns though they had good positive correlation score

In [ ]:

```

```