Sarvjeet Singh

Email: sarvjeet2606@gmail.com

In [1]:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

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

Data Expolration and Pre-processing

In [3]:

```
df
```

Out[3]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nar
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230
...	...	...	...	...	...	...	
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.00	C776
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.00	C1881
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.00	C1365
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.00	C2080
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.00	C873

6362620 rows × 11 columns

In [4]:

```
df.isna().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.drop_duplicates()
```

Out[5]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nar
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230
...	...	...	...	...	...	...	
6362615	743	CASH_OUT	339682.13	C786484425	339682.13	0.00	C776
6362616	743	TRANSFER	6311409.28	C1529008245	6311409.28	0.00	C1881
6362617	743	CASH_OUT	6311409.28	C1162922333	6311409.28	0.00	C1365
6362618	743	TRANSFER	850002.52	C1685995037	850002.52	0.00	C2080
6362619	743	CASH_OUT	850002.52	C1280323807	850002.52	0.00	C873

6362620 rows × 11 columns

In [4]:

```
df.head()
```

Out[4]:

	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 [5]:

```
df.describe()
```

Out[5]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbala
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.224
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.674
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.000
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.146
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.111
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.561

In [6]:

```
df["type"].value_counts()
```

Out[6]:

```
CASH_OUT    2237500
PAYMENT     2151495
CASH_IN     1399284
TRANSFER     532909
DEBIT         41432
Name: type, dtype: int64
```

In [7]:

```
temp = df[df["type"]=="CASH_IN"]
```

In [8]:

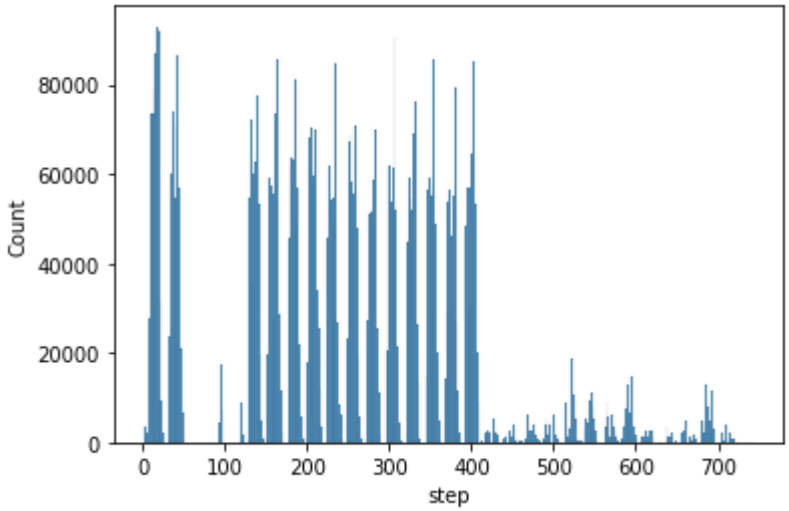
```
temp["isFraud"].value_counts()
```

Out[8]:

```
0    1399284
Name: isFraud, dtype: int64
```

In [9]:

```
sns.histplot(df["step"])
plt.show()
```



Encoding Type column

In [10]:

```
'''CASH_OUT    2237500
PAYMENT     2151495
CASH_IN     1399284
TRANSFER     532909
DEBIT'''

def type(s):
    if s=="CASH_OUT":
        return 0
    elif s=="PAYMENT":
        return 1
    elif s=="CASH_IN":
        return 2
    elif s=="TRANSFER":
        return 3
    elif s=="DEBIT":
        return 4
```

In [11]:

```
df["type"]=df.type.apply(type)
```

In [12]:

```
df["type"].value_counts()
```

Out[12]:

```
0    2237500
1    2151495
2    1399284
3     532909
4      41432
Name: type, dtype: int64
```

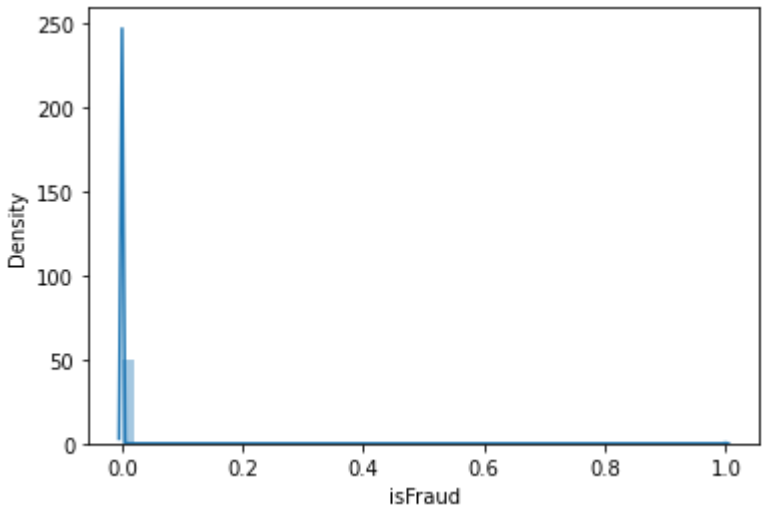
In [48]:

```
sns.distplot(df["isFraud"])
```

```
/home/jeet/.local/lib/python3.6/site-packages/seaborn/distributions.py:255
1: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a fig
ure-level function with similar flexibility) or `histplot` (an axes-level
function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[48]:

```
<AxesSubplot:xlabel='isFraud', ylabel='Density'>
```



In [13]:

```
df.corr()
```

Out[13]:

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceD
step	1.000000	0.012627	0.022373	-0.010058	-0.010299	0.027
type	0.012627	1.000000	0.198987	0.260418	0.270669	0.066
amount	0.022373	0.198987	1.000000	-0.002762	-0.007861	0.294
oldbalanceOrg	-0.010058	0.260418	-0.002762	1.000000	0.998803	0.066
newbalanceOrig	-0.010299	0.270669	-0.007861	0.998803	1.000000	0.067
oldbalanceDest	0.027665	0.066255	0.294137	0.066243	0.067812	1.000
newbalanceDest	0.025888	0.079111	0.459304	0.042029	0.041837	0.976
isFraud	0.031578	0.016171	0.076688	0.010154	-0.008148	-0.005
isFlaggedFraud	0.003277	0.003144	0.012295	0.003835	0.003776	-0.000

In [14]:

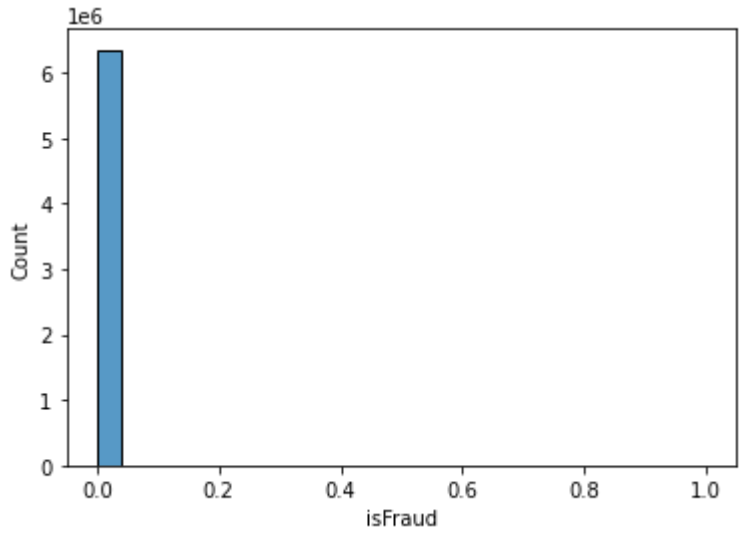
```
df["isFraud"].value_counts()
```

Out[14]:

```
0    6354407
1       8213
Name: isFraud, dtype: int64
```

In [15]:

```
sns.histplot(df["isFraud"])
plt.show()
```



Adding another column *sender_amount = oldbalanceOrg-newbalanceOrig*

In [16]:

```
df["sender_amount"] = df["oldbalanceOrg"]-df["newbalanceOrig"]
```

In [17]:

```
df["sender_amount"].describe()
```

Out[17]:

```
count    6.362620e+06
mean    -2.123056e+04
std      1.466433e+05
min     -1.915268e+06
25%      0.000000e+00
50%      0.000000e+00
75%      1.015044e+04
max      1.000000e+07
Name: sender_amount, dtype: float64
```

Adding another column *dest_amount = oldbalanceDest-newbalanceDest*

In [18]:

```
df["dest_amount"] = df["oldbalanceDest"]-df["newbalanceDest"]
```

In [19]:

```
df["dest_amount"].describe()
```

Out[19]:

```
count    6.362620e+06
mean    -1.242947e+05
std      8.129391e+05
min     -1.056878e+08
25%     -1.491054e+05
50%      0.000000e+00
75%      0.000000e+00
max      1.306083e+07
Name: dest_amount, dtype: float64
```

In [20]:

```
data = df[["isFraud","isFlaggedFraud","type","amount","dest_amount","sender_amount"]]
```

In [21]:

```
data
```

Out[21]:

	isFraud	isFlaggedFraud	type	amount	dest_amount	sender_amount
0	0	0	1	9839.64	0.00	9839.64
1	0	0	1	1864.28	0.00	1864.28
2	1	0	3	181.00	0.00	181.00
3	1	0	0	181.00	21182.00	181.00
4	0	0	1	11668.14	0.00	11668.14
...	...	...	...	...	...	...
6362615	1	0	0	339682.13	-339682.13	339682.13
6362616	1	0	3	6311409.28	0.00	6311409.28
6362617	1	0	0	6311409.28	-6311409.27	6311409.28
6362618	1	0	3	850002.52	0.00	850002.52
6362619	1	0	0	850002.52	-850002.52	850002.52

6362620 rows × 6 columns

In [22]:

```
data.corr()
```

Out[22]:

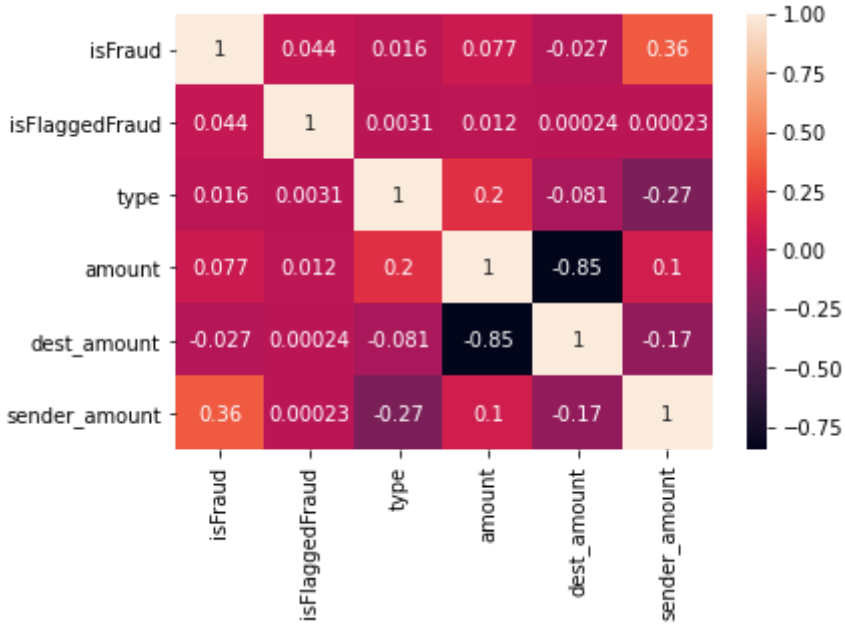
	isFraud	isFlaggedFraud	type	amount	dest_amount	sender_amoun
isFraud	1.000000	0.044109	0.016171	0.076688	-0.027028	0.36247
isFlaggedFraud	0.044109	1.000000	0.003144	0.012295	0.000242	0.00023
type	0.016171	0.003144	1.000000	0.198987	-0.080513	-0.26799
amount	0.076688	0.012295	0.198987	1.000000	-0.845964	0.10233
dest_amount	-0.027028	0.000242	-0.080513	-0.845964	1.000000	-0.16929
sender_amount	0.362472	0.000230	-0.267999	0.102337	-0.169292	1.00000

In [23]:

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

Out[23]:

<AxesSubplot:>



In [24]:

```
data = df[["isFraud","isFlaggedFraud","type","amount","sender_amount"]]
```

In [25]:

```
data
```

Out[25]:

	isFraud	isFlaggedFraud	type	amount	sender_amount
0	0	0	1	9839.64	9839.64
1	0	0	1	1864.28	1864.28
2	1	0	3	181.00	181.00
3	1	0	0	181.00	181.00
4	0	0	1	11668.14	11668.14
...	...	...	...	...	...
6362615	1	0	0	339682.13	339682.13
6362616	1	0	3	6311409.28	6311409.28
6362617	1	0	0	6311409.28	6311409.28
6362618	1	0	3	850002.52	850002.52
6362619	1	0	0	850002.52	850002.52

6362620 rows × 5 columns

Logistic Regression Model

In [26]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [27]:

```
y = data["isFraud"]
X= data[["isFlaggedFraud","type","amount","sender_amount"]]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

In [28]:

```
model = LogisticRegression(random_state=42)
```

In [29]:

```
model.fit(X_train,y_train)
```

Out[29]:

```
LogisticRegression(random_state=42)
```

In [30]:

```
pred1 = model.predict(X_test)
```

In [31]:

```
from sklearn.metrics import recall_score,accuracy_score,f1_score,classification_report
```

In [32]:

```
accuracy_score(y_test,pred1)
```

Out[32]:

```
0.9990989037010959
```

In [33]:

```
recall_score(y_test,pred1)
```

Out[33]:

```
0.3212669683257919
```

In [34]:

```
f1_score(y_test,pred1)
```

Out[34]:

```
0.47592931139549055
```

In [35]:

```
model.score(X_test,y_test)
```

Out[35]:

```
0.9990989037010959
```

Applying Some Standardization

In [36]:

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

In [37]:

```
pipe = make_pipeline(StandardScaler(), LogisticRegression())
pipe.fit(X_train, y_train)
```

Out[37]:

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('logisticregression', LogisticRegression())])
```

In [38]:

```
pipe.score(X_test,y_test)
```

Out[38]:

```
0.9991460540888293
```

In [39]:

```
preds_2 = pipe.predict(X_test)
```

In [40]:

```
recall_score(y_test,preds_2)
```

Out[40]:

```
0.41793500617030027
```

In [41]:

```
f1_score(y_test,preds_2)
```

Out[41]:

```
0.5548880393227745
```

Decision Tree Model

In [42]:

```
from sklearn.tree import DecisionTreeClassifier

decision_tree = DecisionTreeClassifier(max_depth=6)
decision_tree.fit(X_train,y_train)
print("Accuracy of test:",decision_tree.score(X_test,y_test))
```

```
Accuracy of test: 0.9991895372241834
```

In [43]:

```
pred = decision_tree.predict(X_test)
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1906355
           1       0.90      0.41      0.56      2431

    accuracy                           1.00   1908786
   macro avg       0.95      0.70      0.78   1908786
weighted avg       1.00      1.00      1.00   1908786
```

In [44]:

```
f1_score(y_test,pred)
```

Out[44]:

```
0.5623762376237623
```

In [45]:

```
recall_score(y_test,pred)
```

Out[45]:

```
0.4088852324146442
```