IEEE Fraud Detection

0. Project Description

Imagine standing at the check-out counter at the grocery store with a long line behind you and the cashier not-so-quietly announces that your card has been declined. In this moment, you probably aren't thinking about the data science that determined your fate.

Embarrassed, and certain you have the funds to cover everything needed for an epic nacho party for 50 of your closest friends, you try your card again. Same result. As you step aside and allow the cashier to tend to the next customer, you receive a text message from your bank. "Press 1 if you really tried to spend $500 on cheddar cheese."

While perhaps cumbersome (and often embarrassing) in the moment, this fraud prevention system is actually saving consumers millions of dollars per year. Researchers from the IEEE Computational Intelligence Society (IEEE-CIS) want to improve this figure, while also improving the customer experience. With higher accuracy fraud detection, you can get on with your chips without the hassle.

IEEE-CIS works across a variety of AI and machine learning areas, including deep neural networks, fuzzy systems, evolutionary computation, and swarm intelligence. Today they're partnering with the world's leading payment service company, Vesta Corporation, seeking the best solutions for fraud prevention industry, and now you are invited to join the challenge.

In this competition, you'll benchmark machine learning models on a challenging large-scale dataset. The data comes from Vesta's real-world e-commerce transactions and contains a wide range of features from device type to product features. You also have the opportunity to create new features to improve your results.

If successful, you'll improve the efficacy of fraudulent transaction alerts for millions of people around the world, helping hundreds of thousands of businesses reduce their fraud loss and increase their revenue. And of course, you will save party people just like you the hassle of false positives.

Acknowledgements:

Vesta Corporation provided the dataset for this competition. Vesta Corporation is the forerunner in guaranteed e-commerce payment solutions. Founded in 1995, Vesta pioneered the process of fully guaranteed card-not-present (CNP) payment transactions for the telecommunications industry. Since then, Vesta has firmly expanded data science and machine learning capabilities across the globe and solidified its position as the leader in guaranteed ecommerce payments. Today, Vesta guarantees more than $18B in transactions annually.

Header Photo by Tim Evans on Unsplash

https://www.kaggle.com/pyonemyatmaw/ieee-cis-xgb-lgb

1. Loading packages and reading in data

In [1]:
```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from tensorflow import keras
import tensorflow as tf
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import Activation
from tensorflow.python.keras.layers import Input, Dense, Dropout, Flatten, BatchNormalization
from tensorflow.python.keras import regularizers
```

In [2]:
```
train_ID=pd.read_csv('ieee-fraud-detection/train_identity.csv')
train_trans=pd.read_csv('ieee-fraud-detection/train_transaction.csv')
test_ID=pd.read_csv('ieee-fraud-detection/test_identity.csv')
test_trans=pd.read_csv('ieee-fraud-detection/test_transaction.csv')
```

2. looking at the data

In [3]:
```
print(train_ID.shape)
print(train_trans.shape)
print(test_ID.shape)
print(test_trans.shape)
```

```
(144233, 41)
(590540, 394)
(141907, 41)
(506691, 393)
```

2.1 Reduce memory usage

note: we see these are very big datasets. We'll try to reduce the memory usage to lower the chance on a memory error.

In [4]:
```
# reduce memory data function
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.format(end_mem, 100 * (s
tart_mem - end_mem) / start_mem))
    return df
```

In [5]:
```
train_ID = reduce_mem_usage(train_ID)
train_trans = reduce_mem_usage(train_trans)
test_ID = reduce_mem_usage(test_ID)
test_trans = reduce_mem_usage(test_trans)
```

```
Mem. usage decreased to 25.86 Mb (42.7% reduction)
Mem. usage decreased to 542.35 Mb (69.4% reduction)
Mem. usage decreased to 25.44 Mb (42.7% reduction)
Mem. usage decreased to 472.59 Mb (68.9% reduction)
```

2.2. merging the datasets

The 2 datasets (identity and transactions) are joined by transactionID. We merge them, so we get a training dataset and a testing dataset. Not all transactions have a matching identity, so we do a left join (like SQL) on the transactions.

In [6]:
```
train_set = train_trans.merge(train_ID, on='TransactionID', how='left')
test_set = test_trans.merge(test_ID, on='TransactionID', how='left')
```

We will look for differnces in the features between the training and test set.

In [7]:
```
differences = list(set(train_set.columns) - set(test_set.columns))
print("difference is: ", differences)
```

```
difference is:  ['id_21', 'id_31', 'id_19', 'id_11', 'id_16', 'id_08', 'id_05', 'id_12', 'id_38', 'id
_14', 'id_01', 'id_27', 'id_13', 'id_33', 'id_36', 'id_35', 'id_17', 'id_29', 'id_24', 'id_20', 'id_0
4', 'isFraud', 'id_07', 'id_23', 'id_26', 'id_10', 'id_32', 'id_18', 'id_09', 'id_37', 'id_25', 'id_2
2', 'id_06', 'id_03', 'id_02', 'id_34', 'id_15', 'id_30', 'id_28']
```

We see the column names of the id's are spelled different. We will change the column names and make a last control.

In [8]:
```
test_set.rename({'id-01':'id_01','id-02':'id_02','id-03':'id_03','id-04':'id_04','id-05':'id_05', \
                 'id-06':'id_06','id-07':'id_07','id-08':'id_08','id-09':'id_09','id-10':'id_10', \
                 'id-11':'id_11','id-12':'id_12','id-13':'id_13','id-14':'id_14','id-15':'id_15', \
                 'id-16':'id_16','id-17':'id_17','id-18':'id_18','id-19':'id_19','id-20':'id_20', \
                 'id-21':'id_21','id-22':'id_22','id-23':'id_23','id-24':'id_24','id-25':'id_25', \
                 'id-26':'id_26','id-27':'id_27','id-28':'id_28','id-29':'id_29','id-30':'id_30', \
                 'id-31':'id_31', 'id-32':'id_32', 'id-33':'id_33', 'id-34':'id_34', 'id-35':'id_35',
 \
                 'id-36':'id_36', 'id-37':'id_37', 'id-38':'id_38'},axis=1, inplace=True)
differences = list(set(train_set.columns) - set(test_set.columns))
print("difference is: ", differences)
```

```
difference is:  ['isFraud']
```

In [9]:
```
train_set = train_set.head(50000).copy()
test_set = test_set.head(50000).copy()
```

2.3. Changing Dtypes of some features

They tell us that next features are categorical, so we will change them:

Categorical Features - Transaction

- ProductCD
- card1 - card6
- addr1, addr2
- P_emaildomain
- R_emaildomain
- M1 - M9

Categorical Features - Identity

- DeviceType
- DeviceInfo
- id_12 - id_38

The TransactionDT feature is a timedelta from a given reference datetime (not an actual timestamp).

You can read more about the data from this post by the competition host.

In [10]:
```
cat_cols = ['ProductCD', 'card1', 'card2', 'card3', 'card4', 'card5', 'card6', 'addr1', 'addr2', 'P_ema
ildomain',\
            'R_emaildomain', 'M1', 'M2', 'M3', 'M4', 'M5', 'M6', 'M7', 'M8', 'M9', 'DeviceType', 'Device
Info',\
            'id_12', 'id_13', 'id_14', 'id_15', 'id_16', 'id_17', 'id_18', 'id_19', 'id_20', 'id_21', 'i
d_22',\
            'id_23', 'id_24', 'id_25', 'id_26', 'id_27', 'id_28', 'id_29', 'id_30', 'id_31', 'id_32', 'i
d_33',\
            'id_34', 'id_35', 'id_36', 'id_37', 'id_38']
for col in cat_cols:
    train_set[col] = train_set[col].astype('category')
    test_set[col] = test_set[col].astype('category')
```

Now we'll control if it happend.

In [11]:
```
train_set[cat_cols].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Data columns (total 49 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   ProductCD      50000 non-null  category
 1   card1          50000 non-null  category
 2   card2          49303 non-null  category
 3   card3          49997 non-null  category
 4   card4          49994 non-null  category
 5   card5          49765 non-null  category
 6   card6          49997 non-null  category
 7   addr1          47409 non-null  category
 8   addr2          47409 non-null  category
 9   P_emaildomain  41014 non-null  category
 10  R_emaildomain  11660 non-null  category
 11  M1             17967 non-null  category
 12  M2             17967 non-null  category
 13  M3             17967 non-null  category
 14  M4             22662 non-null  category
 15  M5             17826 non-null  category
 16  M6             31329 non-null  category
 17  M7             9224 non-null   category
 18  M8             9224 non-null   category
 19  M9             9224 non-null   category
 20  DeviceType     16068 non-null  category
 21  DeviceInfo     14351 non-null  category
 22  id_12          16456 non-null  category
 23  id_13          12782 non-null  category
 24  id_14          11894 non-null  category
 25  id_15          16069 non-null  category
 26  id_16          15137 non-null  category
 27  id_17          15921 non-null  category
 28  id_18          4853 non-null   category
 29  id_19          15917 non-null  category
 30  id_20          15912 non-null  category
 31  id_21          604 non-null    category
 32  id_22          608 non-null    category
 33  id_23          608 non-null    category
 34  id_24          559 non-null    category
 35  id_25          598 non-null    category
 36  id_26          608 non-null    category
 37  id_27          608 non-null    category
 38  id_28          16069 non-null  category
 39  id_29          16069 non-null  category
 40  id_30          11673 non-null  category
 41  id_31          16048 non-null  category
 42  id_32          11674 non-null  category
 43  id_33          10821 non-null  category
 44  id_34          11640 non-null  category
 45  id_35          16069 non-null  category
 46  id_36          16069 non-null  category
 47  id_37          16069 non-null  category
 48  id_38          16069 non-null  category
dtypes: category(49)
memory usage: 3.4 MB
```

In [12]:
```
test_set[cat_cols].info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Data columns (total 49 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   ProductCD      50000 non-null  category
 1   card1          50000 non-null  category
 2   card2          49501 non-null  category
 3   card3          49998 non-null  category
 4   card4          49975 non-null  category
 5   card5          49877 non-null  category
 6   card6          49998 non-null  category
 7   addr1          45404 non-null  category
 8   addr2          45404 non-null  category
 9   P_emaildomain  42242 non-null  category
 10  R_emaildomain  8858 non-null   category
 11  M1             36955 non-null  category
 12  M2             36955 non-null  category
 13  M3             36955 non-null  category
 14  M4             27196 non-null  category
 15  M5             22316 non-null  category
 16  M6             39377 non-null  category
 17  M7             29849 non-null  category
 18  M8             29851 non-null  category
 19  M9             29851 non-null  category
 20  DeviceType     8753 non-null   category
 21  DeviceInfo     7179 non-null   category
 22  id_12          9029 non-null   category
 23  id_13          8409 non-null   category
 24  id_14          4301 non-null   category
 25  id_15          8766 non-null   category
 26  id_16          7894 non-null   category
 27  id_17          8661 non-null   category
 28  id_18          3226 non-null   category
 29  id_19          8652 non-null   category
 30  id_20          8617 non-null   category
 31  id_21          321 non-null    category
 32  id_22          322 non-null    category
 33  id_23          322 non-null    category
 34  id_24          294 non-null    category
 35  id_25          319 non-null    category
 36  id_26          322 non-null    category
 37  id_27          322 non-null    category
 38  id_28          8758 non-null   category
 39  id_29          8758 non-null   category
 40  id_30          4228 non-null   category
 41  id_31          8708 non-null   category
 42  id_32          4230 non-null   category
 43  id_33          4230 non-null   category
 44  id_34          4307 non-null   category
 45  id_35          8766 non-null   category
 46  id_36          8766 non-null   category
 47  id_37          8766 non-null   category
 48  id_38          8766 non-null   category
dtypes: category(49)
memory usage: 3.3 MB
```

The 'is_fraud' feature exists only of 0 and 1. We'll make a category of it too. (test set has no 'isFraud')

In [13]:
```
train_set['isFraud'] = train_set['isFraud'].astype('uint8').astype('category')
```

2.4. Looking at the descriptives and countplots

In [14]:
```
train_set.iloc[:,0:15].describe(include="all")
```

Out[14]:

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	card4	card5	card6	addr1
count	5.000000e+04	50000.0	5.000000e+04	5.000000e+04	50000	50000.0	49303.0	49997.0	49994	49765.0	49997	47409.0
unique	NaN	2.0	NaN	NaN	5	5446.0	497.0	58.0	4	74.0	4	255.0
top	NaN	0.0	NaN	NaN	W	7919.0	321.0	150.0	visa	226.0	debit	299.0
freq	NaN	48643.0	NaN	NaN	32798	1750.0	4035.0	45067.0	32561	25798.0	34205	4164.0
mean	3.012000e+06	NaN	6.377990e+05	inf	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	1.443390e+04	NaN	3.229399e+05	inf	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	2.987000e+06	NaN	8.640000e+04	2.919922e-01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	2.999500e+06	NaN	3.522868e+05	4.200000e+01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	3.012000e+06	NaN	6.537475e+05	6.850000e+01	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	3.024499e+06	NaN	9.321258e+05	1.200000e+02	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	3.036999e+06	NaN	1.189336e+06	4.828000e+03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [15]:
```
train_set.iloc[:,15:30].describe(include="all")
```

Out[15]:

	P_emaildomain	R_emaildomain	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
count	41014	11660	50000.0	50000.0	50000.000000	50000.0	50000.0	50000.0	50000.0	50000.0	50000.0000	50000.0
unique	59	57	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
top	gmail.com	gmail.com	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
freq	17909	4367	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	inf	inf	0.016235	inf	inf	inf	inf	inf	inf	inf
std	NaN	NaN	inf	inf	0.156006	inf	inf	inf	inf	inf	14.6875	inf
min	NaN	NaN	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0000	0.0
25%	NaN	NaN	1.0	1.0	0.000000	0.0	0.0	1.0	0.0	0.0	0.0000	0.0
50%	NaN	NaN	1.0	1.0	0.000000	0.0	0.0	1.0	0.0	0.0	1.0000	0.0
75%	NaN	NaN	3.0	3.0	0.000000	0.0	1.0	2.0	0.0	1.0	2.0000	1.0
max	NaN	NaN	1892.0	2390.0	16.000000	1166.0	290.0	1166.0	1167.0	1742.0	180.0000	1909.0

In [16]:
```
train_set.iloc[:,30:45].describe(include="all")
```

Out[16]:

	C14	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13
count	50000.0	50000.0	23484.0	24444.0	26829.0	17821.0	4825.0	2356.0	8750.000000	8750.000000	37830.0	15298.0	4097.0	3678.0
mean	inf	inf	inf	inf	inf	inf	inf	inf	inf	0.562988	inf	inf	inf	inf
std	inf	inf	inf	inf	inf	inf	inf	inf	inf	0.315430	inf	inf	inf	inf
min	0.0	0.0	0.0	0.0	-122.0	0.0	-83.0	0.0	0.000000	0.000000	0.0	-33.0	-83.0	0.0
25%	1.0	0.0	28.0	1.0	0.0	1.0	0.0	0.0	5.679688	0.208374	0.0	0.0	0.0	0.0
50%	1.0	0.0	106.0	8.0	32.0	11.0	0.0	0.0	66.875000	0.666504	23.0	30.0	0.0	0.0
75%	2.0	102.0	278.0	28.0	261.0	31.0	51.0	35.0	239.250000	0.791504	210.0	232.0	20.0	0.0
max	784.0	636.0	636.0	616.0	700.0	656.0	708.0	673.0	1135.000000	0.958496	705.0	499.0	498.0	710.0

In [17]:
```
train_set.iloc[:,45:60].describe(include="all")
```

Out[17]:

	D15	M1	M2	M3	M4	M5	M6	M7	M8	M9	V1	V2	V3	V4
count	33499.0	17967	17967	17967	22662	17826	31329	9224	9224	9224	15298.000000	15298.000000	15298.000000	15298.000000
unique	NaN	1	2	2	3	2	2	2	2	2	NaN	NaN	NaN	NaN
top	NaN	T	T	T	M0	F	F	F	F	T	NaN	NaN	NaN	NaN
freq	NaN	17967	16168	14030	14864	9749	17030	7933	5684	7888	NaN	NaN	NaN	NaN
mean	inf	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.999512	1.023438	1.049805	0.812500
std	inf	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.018066	0.168823	0.251221	0.435791
min	-83.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	0.000000	0.000000
25%	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	1.000000	1.000000	1.000000
50%	64.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	1.000000	1.000000	1.000000
75%	319.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	1.000000	1.000000	1.000000
max	709.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	4.000000	5.000000	4.000000

In [18]:
```
train_set.iloc[:,60:75].describe(include="all")
```

Out[18]:

	V6	V7	V8	V9	V10	V11	V12	V13	V14
count	15298.000000	15298.000000	15298.000000	15298.000000	15298.000000	15298.000000	37826.000000	37826.000000	37826.000000
mean	1.027344	1.050781	1.015625	1.029297	0.478760	0.493164	0.497070	0.538086	1.000000
std	0.182861	0.259277	0.138550	0.192139	0.516602	0.547852	0.512207	0.535156	0.022995
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000
50%	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
max	5.000000	5.000000	6.000000	6.000000	3.000000	4.000000	3.000000	4.000000	1.000000

In [19]:
```
train_set.iloc[:,75:90].describe(include="all")
```

Out[19]:

	V21	V22	V23	V24	V25	V26	V27	V28	V29
count	37826.000000	37826.000000	37826.000000	37826.000000	37826.000000	37826.000000	37826.000000	37826.000000	37826.000000
mean	0.099915	0.108826	1.046875	1.079102	0.921875	0.947266	0.002432	0.002512	0.385254
std	0.322021	0.380127	0.317139	0.426514	0.327148	0.404785	0.049286	0.051117	0.512207
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	1.000000
max	5.000000	8.000000	13.000000	13.000000	7.000000	13.000000	1.000000	2.000000	4.000000

In [20]:
```
train_set.iloc[:,90:105].describe(include="all")
```

Out[20]:

	V36	V37	V38	V39	V40	V41	V42	V43	V44
count	26828.000000	26828.000000	26828.000000	26828.000000	26828.000000	26828.000000	26828.000000	26828.000000	26828.000000
mean	0.519531	1.071289	1.111328	0.173950	0.181152	0.999023	0.164551	0.174438	1.070312
std	0.541504	0.390625	0.520020	0.438232	0.463379	0.036102	0.385986	0.427246	0.357178
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000
50%	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000
75%	1.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	1.000000
max	3.000000	11.000000	14.000000	10.000000	10.000000	1.000000	3.000000	6.000000	12.000000

In [21]:
```
train_set.iloc[:,105:120].describe(include="all")
```

Out[21]:

	V51	V52	V53	V54	V55	V56	V57	V58	V59
count	26828.000000	26828.000000	38282.000000	38282.000000	38282.000000	38282.000000	38282.000000	38282.000000	38282.000000
mean	0.160156	0.165527	0.499756	0.541016	1.042969	1.075195	0.111267	0.113586	0.113892
std	0.391602	0.415771	0.510254	0.530762	0.299561	0.407715	0.327881	0.337158	0.346924
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000
max	4.000000	6.000000	5.000000	5.000000	14.000000	15.000000	6.000000	7.000000	7.000000

In [22]:
```
train_set.iloc[:,120:135].describe(include="all")
```

Out[22]:

	V66	V67	V68	V69	V70	V71	V72	V73	V74
count	38282.000000	38282.000000	38282.000000	38282.000000	38282.000000	38282.000000	38282.000000	38282.000000	38282.000000
mean	0.916016	0.936035	0.001436	0.380371	0.396973	0.122864	0.125244	0.114075	0.118713
std	0.314453	0.337646	0.037872	0.514648	0.549805	0.342773	0.351807	0.342285	0.366943
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	1.000000	0.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000
max	5.000000	8.000000	1.000000	4.000000	5.000000	6.000000	7.000000	5.000000	8.000000

In [23]:
```
train_set.iloc[:,135:150].describe(include="all")
```

Out[23]:

	V81	V82	V83	V84	V85	V86	V87	V88	V89
count	33496.000000	33496.000000	33496.000000	33496.000000	33496.000000	33496.000000	33496.000000	33496.000000	33496.000000
mean	0.150879	0.821777	0.862793	0.139404	0.145142	1.053711	1.083984	0.998535	0.002836
std	0.416260	0.443604	0.483154	0.356934	0.379883	0.317139	0.376709	0.034546	0.053741
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	1.000000	0.000000
50%	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	1.000000	0.000000
75%	0.000000	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	1.000000	0.000000
max	8.000000	5.000000	6.000000	3.000000	5.000000	10.000000	10.000000	1.000000	2.000000

In [24]:
```
train_set.iloc[:,150:165].describe(include="all")
```

Out[24]:

	V96	V97	V98	V99	V100	V101	V102	V103	V104
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000
mean	1.108398	0.459961	0.056854	0.604980	0.181396	0.119019	0.389648	0.207520	0.055267
std	4.226562	1.638672	0.257812	1.722656	0.631836	0.609863	2.966797	1.260742	0.317627
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	86.000000	37.000000	5.000000	34.000000	16.000000	16.000000	72.000000	34.000000	7.000000

In [25]:
```
train_set.iloc[:,165:180].describe(include="all")
```

Out[25]:

	V111	V112	V113	V114	V115	V116	V117	V118	V119
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000
mean	1.003906	1.005859	1.003906	1.007812	1.016602	1.010742	1.001953	1.001953	1.001953
std	0.068909	0.076660	0.071594	0.095154	0.142212	0.108948	0.042389	0.044891	0.043304
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
50%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
max	4.000000	4.000000	4.000000	5.000000	5.000000	5.000000	3.000000	3.000000	3.000000

In [26]:
```
train_set.iloc[:,180:195].describe(include="all")
```

Out[26]:

	V126	V127	V128	V129	V130	V131	V132	V133	V134	V135
count	50000.000000	50000.000000	50000.000000	50000.0	50000.0	50000.0	50000.000000	50000.000000	50000.000000	50000.000000
mean	37.912209	127.153961	64.632362	inf	inf	inf	18.988361	49.898525	31.557068	10.702315
std	325.809662	628.438538	425.309204	inf	inf	inf	172.833359	433.363953	289.123474	246.507965
min	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000
75%	0.000000	59.000000	0.000000	0.0	0.0	0.0	0.000000	0.000000	0.000000	0.000000
max	50820.000000	50820.000000	50820.000000	3894.0	4676.0	4676.0	7985.000000	15227.000000	11260.000000	50820.000000

In [27]:
```
train_set.iloc[:,195:210].describe(include="all")
```

Out[27]:

	V141	V142	V143	V144	V145	V146	V147	V148	V149
count	12213.000000	12213.000000	1.221300e+04	1.221300e+04	12213.000	12213.000000	12213.000000	12213.000000	12213.000000
mean	0.024811	0.030701	inf	inf	inf	0.080261	0.082275	0.508301	0.510254
std	0.187988	0.249390	1.349219e+01	1.610938e+01	87.125	0.427246	0.434082	0.541016	0.545898
min	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	2.300000e+01	2.600000e+01	144.000	0.000000	0.000000	1.000000	1.000000
max	5.000000	6.000000	4.900000e+01	5.300000e+01	245.000	7.000000	8.000000	6.000000	6.000000

In [28]:
```
train_set.iloc[:,210:225].describe(include="all")
```

Out[28]:

	V156	V157	V158	V159	V160	V161	V162	V163	V164
count	12213.000000	12213.000000	12213.000000	12213.0	12213.000000	1.221300e+04	1.221300e+04	1.221300e+04	12213.000000
mean	0.514648	0.532715	0.537598	inf	148280.937500	2.871094e+00	3.949219e+00	3.189453e+00	806.819458
std	0.564941	0.598633	0.613770	inf	181355.890625	inf	inf	inf	1014.393921
min	0.000000	0.000000	0.000000	0.0	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000
25%	0.000000	0.000000	0.000000	0.0	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000
50%	0.000000	0.000000	0.000000	0.0	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	0.000000
75%	1.000000	1.000000	1.000000	25632.0	318223.625000	0.000000e+00	0.000000e+00	0.000000e+00	1689.949951
max	9.000000	7.000000	9.000000	44192.0	512182.500000	1.000000e+03	1.300000e+03	1.000000e+03	6300.000000

In [29]:
```
train_set.iloc[:,225:240].describe(include="all")
```

Out[29]:

	V171	V172	V173	V174	V175	V176	V177	V178	V179
count	15251.000000	15251.000000	15251.000000	15251.000000	15251.000000	15251.000000	15251.000000	15251.000000	15251.000000
mean	1.315430	0.076660	0.040710	0.073608	0.090942	1.129883	0.176636	0.467285	0.268799
std	1.220703	0.407959	0.220581	0.304199	0.449707	0.677246	0.791992	3.445312	1.439453
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
25%	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
75%	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	0.000000
max	18.000000	8.000000	3.000000	5.000000	8.000000	13.000000	16.000000	66.000000	29.000000

In [30]:
```
train_set.iloc[:,240:255].describe(include="all")
```

Out[30]:

	V186	V187	V188	V189	V190	V191	V192	V193	V194
count	15251.000000	15251.000000	15251.000000	15251.000000	15251.000000	15251.000000	15251.000000	15251.000000	15251.000000
mean	1.068359	1.448242	0.979492	0.995117	1.083008	1.041992	1.190430	1.113281	0.961426
std	0.514648	4.910156	0.324219	0.505371	0.562500	0.397949	2.039062	1.204102	0.271729
min	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000
25%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
50%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
max	11.000000	85.000000	7.000000	15.000000	12.000000	9.000000	34.000000	23.000000	6.000000

In [31]:
```
train_set.iloc[:,255:270].describe(include="all")
```

Out[31]:

	V201	V202	V203	V204	V205	V206	V207	V208	V209	V210	V211
count	15251.000000	15251.000000	15251.000000	15251.000000	15251.0	15251.0	15251.0	15251.0	15251.0	15251.0	15251.000000
mean	1.045898	60.118832	277.478363	123.974892	inf	inf	inf	inf	inf	inf	20.391985
std	0.637695	954.388489	2985.113281	1361.162476	inf	inf	inf	inf	inf	inf	136.290451
min	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
25%	1.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
50%	1.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
75%	1.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
max	15.000000	101640.000000	101640.000000	101640.000000	2550.0	1500.0	8076.0	1000.0	8048.0	1450.0	6300.000000

In [32]:
```
train_set.iloc[:,270:285].describe(include="all")
```

Out[32]:

	V216	V217	V218	V219	V220	V221	V222	V223	V224
count	15251.000000	15157.000000	15157.000000	15157.000000	15551.000000	15551.000000	15551.000000	15157.000000	15157.000000
mean	37.185570	0.429443	1.700195	0.820801	0.096924	1.172852	1.223633	0.102783	0.878418
std	944.747925	1.446289	14.250000	4.992188	0.635742	1.692383	1.827148	0.702148	9.085938
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	0.000000
max	101640.000000	20.000000	219.000000	79.000000	15.000000	92.000000	92.000000	16.000000	144.000000

In [33]:
```
train_set.iloc[:,285:300].describe(include="all")
```

Out[33]:

	V231	V232	V233	V234	V235	V236	V237	V238	V239
count	15157.000000	15157.000000	15157.000000	15551.000000	15157.000000	15157.000000	15157.000000	15551.000000	15551.000000
mean	0.198486	0.466064	0.277100	1.070312	0.117126	0.319824	0.174927	0.101318	0.104614
std	0.943848	3.005859	1.347656	6.921875	0.483887	2.359375	0.931152	0.459473	0.469727
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	17.000000	42.000000	22.000000	70.000000	7.000000	34.000000	16.000000	11.000000	11.000000

In [34]:
```
train_set.iloc[:,300:315].describe(include="all")
```

Out[34]:

	V246	V247	V248	V249	V250	V251	V252	V253	V254
count	15157.000000	15157.000000	15157.000000	15157.000000	15551.000000	15551.000000	15157.000000	15157.000000	15157.000000
mean	1.110352	1.052734	1.217773	1.126953	0.863770	0.868164	1.078125	1.833008	1.326172
std	0.803223	0.511719	2.435547	1.361328	0.407959	0.421143	0.730957	10.421875	3.812500
min	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	1.000000
25%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
50%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
max	18.000000	12.000000	36.000000	22.000000	8.000000	8.000000	18.000000	163.000000	60.000000

In [35]:
```
train_set.iloc[:,315:330].describe(include="all")
```

Out[35]:

	V261	V262	V263	V264	V265	V266	V267	V268	V269	V270	V271
count	15157.000000	15157.000000	15157.000000	15157.000000	15157.000000	15157.0	15157.0	15157.0	15157.0	15551.0	15551.0
mean	1.235352	1.054688	61.763264	417.893066	162.497574	inf	inf	inf	inf	inf	inf
std	2.992188	1.087891	955.751587	4837.699707	1930.262329	inf	inf	inf	inf	inf	inf
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
25%	1.000000	1.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
50%	1.000000	1.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
75%	1.000000	1.000000	0.000000	0.000000	0.000000	0.0	0.0	0.0	0.0	0.0	0.0
max	49.000000	20.000000	101640.000000	101640.000000	101640.000000	2300.0	27008.0	12552.0	2300.0	1300.0	1300.0

In [36]:
```
train_set.iloc[:,330:345].describe(include="all")
```

Out[36]:

	V276	V277	V278	V279	V280	V281	V282	V283	V284
count	15157.000000	15157.000000	15157.000000	49989.000000	49989.000000	50000.000000	50000.000000	50000.000000	49989.000000
mean	28.513453	96.634651	50.147686	0.301025	0.787598	0.075500	0.832031	1.026367	0.076294
std	931.554932	1292.671753	992.961121	1.090820	4.574219	0.451416	0.853516	2.203125	0.300049
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	1.000000	1.000000	0.000000
max	101640.000000	101640.000000	101640.000000	36.000000	196.000000	10.000000	13.000000	68.000000	5.000000

In [37]:
```
train_set.iloc[:,345:360].describe(include="all")
```

Out[37]:

	V291	V292	V293	V294	V295	V296	V297	V298	V299
count	49989.0	49989.000000	49989.000000	4.998900e+04	49989.000000	50000.000000	49989.000000	49989.000000	49989.000000
mean	inf	1.235352	0.162476	9.985352e-01	0.401611	0.192993	0.059448	0.159180	0.087219
std	inf	3.718750	0.920898	inf	4.210938	1.671875	0.346680	1.255859	0.556152
min	1.0	1.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.0	1.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000
50%	1.0	1.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000
75%	1.0	1.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000
max	874.0	186.000000	36.000000	8.340000e+02	196.000000	61.000000	9.000000	47.000000	19.000000

In [38]:
```
train_set.iloc[:,360:375].describe(include="all")
```

Out[38]:

	V306	V307	V308	V309	V310	V311	V312	V313	V314	V315	V316
count	49989.000000	49989.000000	49989.000000	49989.0	49989.000000	49989.0	49989.0	50000.0	50000.0	50000.0	49989.000000
mean	48.756142	224.192764	104.843536	inf	inf	inf	inf	inf	inf	inf	25.503828
std	575.531189	1634.083374	813.625671	inf	inf	inf	inf	inf	inf	inf	221.025864
min	0.000000	0.000000	0.000000	0.0	0.0000	0.0	0.0	0.0	0.0	0.0	0.000000
25%	0.000000	0.000000	0.000000	0.0	0.0000	0.0	0.0	0.0	0.0	0.0	0.000000
50%	0.000000	0.000000	0.000000	0.0	0.0000	0.0	0.0	0.0	0.0	0.0	0.000000
75%	0.000000	117.000000	20.000000	0.0	78.9375	0.0	0.0	0.0	0.0	0.0	0.000000
max	101640.000000	101640.000000	101640.000000	3162.0	19808.0000	3162.0	3162.0	3956.0	5020.0	3956.0	7985.000000

In [39]:
```
train_set.iloc[:,375:390].describe(include="all")
```

Out[39]:

	V321	V322	V323	V324	V325	V326	V327	V328	V329
count	49989.000000	12268.000000	12268.000000	12268.000000	12268.000000	12268.000000	12268.000000	12268.000000	12268.000000
mean	16.797197	0.262695	0.571289	0.365723	0.031860	0.262939	0.106140	0.103516	0.151001
std	520.974182	0.915527	2.656250	1.336914	0.210693	1.925781	0.707031	0.458008	0.820312
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	101640.000000	16.000000	42.000000	26.000000	5.000000	34.000000	11.000000	7.000000	14.000000

In [40]:
```python
train_set.iloc[:,390:405].describe(include="all")
```

Out[40]:

	V336	V337	V338	V339	id_01	id_02	id_03	id_04	id_05	
count	12268.0	12268.000000	12268.000000	12268.000000	1.645600e+04	16069.000000	7456.000000	7456.000000	15605.000000	1.5605
mean	inf	15.742602	21.362671	16.464481	-inf	133358.390625	0.107544	-0.070801	1.889648	
std	inf	91.084587	131.079559	92.730209	1.403906e+01	123074.875000	0.700195	0.726562	5.230469	1.6046
min	0.0	0.000000	0.000000	0.000000	-1.000000e+02	1120.000000	-9.000000	-13.000000	-30.000000	-1.0000
25%	0.0	0.000000	0.000000	0.000000	-5.000000e+00	52145.000000	0.000000	0.000000	0.000000	-4.0000
50%	0.0	0.000000	0.000000	0.000000	-5.000000e+00	97665.000000	0.000000	0.000000	0.000000	0.0000
75%	0.0	0.000000	0.000000	0.000000	0.000000e+00	171732.000000	0.000000	0.000000	1.000000	0.0000
max	1300.0	1700.000000	3000.000000	1700.000000	0.000000e+00	998295.000000	9.000000	0.000000	52.000000	0.0000

In [41]:
```python
train_set.iloc[:,405:420].describe(include="all")
```

Out[41]:

	id_12	id_13	id_14	id_15	id_16	id_17	id_18	id_19	id_20	id_21	id_22	id_23	id_24	i
count	16456	12782.0	11894.0	16069	15137	15921.0	4853.0	15917.0	15912.0	604.0	608.0	608	559.0	5
unique	2	36.0	21.0	3	2	49.0	12.0	349.0	164.0	136.0	6.0	3	5.0	1
top	NotFound	52.0	-300.0	Found	NotFound	166.0	15.0	410.0	507.0	252.0	14.0	IP_PROXY:TRANSPARENT	15.0	3
freq	14433	6997.0	6565.0	7786	7681	11828.0	2342.0	1693.0	1828.0	200.0	559.0	329	265.0	1

In [42]:
```python
train_set.iloc[:,420:435].describe(include="all")
```

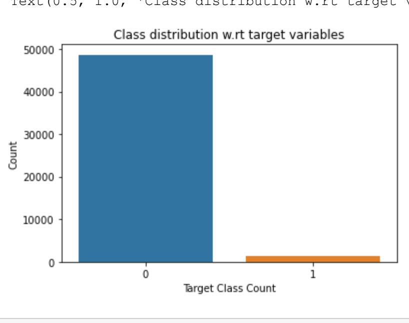
Out[42]:

	id_27	id_28	id_29	id_30	id_31	id_32	id_33	id_34	id_35	id_36	id_37	id_38	DeviceType	DeviceInfo
count	608	16069	16069	11673	16048	11674.0	10821	11640	16069	16069	16069	16069	16068	14351
unique	2	2	2	63	73	4.0	89	3	2	2	2	2	2	585
top	Found	Found	Found	Windows 10	chrome 62.0	24.0	1920x1080	match_status:2	T	F	T	T	desktop	Windows
freq	607	9059	8750	3344	4956	8129.0	2576	9104	11676	14703	12671	13423	10887	6224

3. Data cleaning

In [43]:
```python
counts = train_set['isFraud'].value_counts().values
ax = sns.barplot([0,1], counts)
ax.set(title='Class distribution w.rt target variables', xlabel = 'Target Class Count', ylabel='Count')
```

Out[43]:
```
[Text(0, 0.5, 'Count'),
 Text(0.5, 0, 'Target Class Count'),
 Text(0.5, 1.0, 'Class distribution w.rt target variables')]
```



In [44]:
```python
train_set.isnull().sum()[train_set.isnull().sum() != 0]
```

Out[44]:
```
card2           697
card3             3
card4             6
card5           235
card6             3
               ...
id_36         33931
id_37         33931
id_38         33931
DeviceType    33932
DeviceInfo    35649
Length: 359, dtype: int64
```

3.1. split-up categorical and numerical features

note: We see that there is almost missing data in e every feature. Before we can scale and impute our numerical and one-hot-encode our categorical features, we have to know which are wich ones.

In [45]:
```python
train_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Columns: 434 entries, TransactionID to DeviceInfo
dtypes: category(50), float16(337), float32(45), int32(2)
memory usage: 44.5 MB
```

In [46]:
```python
num_cols = [cname for cname in train_set.columns if
                  train_set[cname].dtype in ['int32', 'float16', 'float32']]

cat_cols = [cname for cname in train_set.columns if
                   train_set[cname].dtype.name == "category"]
```

In [47]:
```python
train_set.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Columns: 434 entries, TransactionID to DeviceInfo
dtypes: category(50), float16(337), float32(45), int32(2)
memory usage: 44.5 MB
```

3.2. Transform the data

- **Categorical data:** Make dummies for every feature.
- **Numerical data:** Make a pipeline with a imputer to fill all missing data. We'll also scale the data.

In [48]:
```python
from imblearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

trainset = train_set.copy()
testset = test_set.copy()

#Preprocessing for categorical data
for col in cat_cols:
    if col != 'isFraud':
        dummies = pd.get_dummies(trainset[col], dummy_na=False, prefix=col)
        trainset = pd.concat([trainset, dummies],axis=1)
        trainset.drop(columns=[col], inplace=True)

        testset = pd.concat([testset, dummies], axis=1)
        testset.drop(columns=[col], inplace=True)

#Preprocessing for numerical data
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

trainset[num_cols] = numeric_transformer.fit_transform(trainset[num_cols])
testset[num_cols] = numeric_transformer.fit_transform(testset[num_cols])
```

In [49]:
```python
trainset.isnull().sum()[trainset.isnull().sum() != 0]
```

Out[49]:
```
Series([], dtype: int64)
```

In [50]:
```python
trainset.shape
```

Out[50]:
```
(50000, 8689)
```

In [51]:
```python
testset.shape
```

Out[51]:
```
(50000, 8688)
```

4. Building auto-encoder

4.1. Train - test - val - split

In [52]:
```python
X_train, X_test = train_test_split(trainset, test_size=0.2, random_state=1234)

y_test = X_test['isFraud']
X_test = X_test.drop(['isFraud'], 1)

X_train = X_train[X_train['isFraud'] == 0]
X_train = X_train.drop(['isFraud'], 1)

X_val, X_test, y_val, y_test = train_test_split(X_test, y_test, test_size=0.4, random_state=123)

X_train = X_train.values
X_val = X_val.values
X_test = X_test.values
```

We'll check the amount of observations that are flagged as fraud, so we know how much observations we are looking for.

In [53]:
```python
y_val.value_counts()
```

Out[53]:
```
0    5846
1     154
Name: isFraud, dtype: int64
```

4.2. Building the auto-encoder model

In [54]:
```python
# the number of features is the input shape
input_dim = X_train.shape[1]

## rate
rate = 0.2

# encoder model
    ## input layer
input_layer = Input(shape=(input_dim, ))

    ## encoder layers
encoder = Dense(128, activation="relu", activity_regularizer=regularizers.l1(10e-5))(input_layer)
Dropout(rate)(encoder)
BatchNormalization()(encoder)
encoder = Dense(64, activation="relu")(encoder)
Dropout(rate)(encoder)
BatchNormalization()(encoder)
encoder = Dense(32, activation = 'relu') (encoder)

# decoder model
decoder = Dense(64, activation = "relu") (encoder)
Dropout(rate)(decoder)
BatchNormalization()(encoder)
decoder = Dense(128, activation='relu')(encoder)
Dropout(rate)(decoder)
BatchNormalization()(encoder)
decoder = Dense(input_dim, activation='sigmoid')(decoder)

# build auto-encoder model
autoencoder = Model(inputs = input_layer, outputs = decoder)
```

4.3. Compiling and fitting the model

In [55]:
```python
#compile and fit
from tensorflow.python.keras.callbacks import ModelCheckpoint
from tensorflow.python.keras.callbacks import EarlyStopping

nb_epoch = 100
batch_size = 29
early_stopping = EarlyStopping(patience=10)
autoencoder.compile(optimizer='adam',
                    loss='mean_squared_error',
                    metrics=['accuracy'])

checkpointer = ModelCheckpoint(filepath="model.h5",
                               verbose=0,
                               save_best_only=True)

history = autoencoder.fit(X_train, X_train,
                          epochs = nb_epoch,
                         batch_size = batch_size,
                         shuffle=True,
                         validation_split=0.2,
                         verbose=1,
                         callbacks=[checkpointer, early_stopping])
```

```
Epoch 1/100
1074/1074 [==============================] - 32s 29ms/step - loss: 0.0613 - accuracy: 4.3939e-04 - va
l_loss: 0.0357 - val_accuracy: 0.0037
Epoch 2/100
1074/1074 [==============================] - 30s 28ms/step - loss: 0.0341 - accuracy: 0.0056 - val_lo
ss: 0.0353 - val_accuracy: 0.0030
Epoch 3/100
1074/1074 [==============================] - 24s 22ms/step - loss: 0.0339 - accuracy: 0.0084 - val_lo
ss: 0.0348 - val_accuracy: 0.0267
Epoch 4/100
1074/1074 [==============================] - 25s 23ms/step - loss: 0.0327 - accuracy: 0.0460 - val_lo
ss: 0.0345 - val_accuracy: 0.0123
Epoch 5/100
1074/1074 [==============================] - 27s 25ms/step - loss: 0.0320 - accuracy: 0.0364 - val_lo
ss: 0.0344 - val_accuracy: 0.0444
Epoch 6/100
1074/1074 [==============================] - 25s 23ms/step - loss: 0.0328 - accuracy: 0.0381 - val_lo
ss: 0.0343 - val_accuracy: 0.0460
Epoch 7/100
1074/1074 [==============================] - 30s 28ms/step - loss: 0.0325 - accuracy: 0.0351 - val_lo
ss: 0.0342 - val_accuracy: 0.0325
Epoch 8/100
1074/1074 [==============================] - 33s 31ms/step - loss: 0.0313 - accuracy: 0.0302 - val_lo
ss: 0.0341 - val_accuracy: 0.0288
Epoch 9/100
1074/1074 [==============================] - 30s 28ms/step - loss: 0.0319 - accuracy: 0.0345 - val_lo
ss: 0.0340 - val_accuracy: 0.0374
Epoch 10/100
1074/1074 [==============================] - 33s 30ms/step - loss: 0.0319 - accuracy: 0.0278 - val_lo
ss: 0.0340 - val_accuracy: 0.0324
Epoch 11/100
1074/1074 [==============================] - 34s 32ms/step - loss: 0.0325 - accuracy: 0.0291 - val_lo
ss: 0.0339 - val_accuracy: 0.0320
Epoch 12/100
1074/1074 [==============================] - 31s 29ms/step - loss: 0.0341 - accuracy: 0.0286 - val_lo
ss: 0.0339 - val_accuracy: 0.0380
Epoch 13/100
1074/1074 [==============================] - 29s 27ms/step - loss: 0.0321 - accuracy: 0.0350 - val_lo
ss: 0.0339 - val_accuracy: 0.0619
Epoch 14/100
1074/1074 [==============================] - 27s 25ms/step - loss: 0.0334 - accuracy: 0.0370 - val_lo
ss: 0.0338 - val_accuracy: 0.0378
Epoch 15/100
1074/1074 [==============================] - 28s 26ms/step - loss: 0.0341 - accuracy: 0.0350 - val_lo
ss: 0.0338 - val_accuracy: 0.0560
Epoch 16/100
1074/1074 [==============================] - 28s 26ms/step - loss: 0.0318 - accuracy: 0.0415 - val_lo
ss: 0.0338 - val_accuracy: 0.0348
Epoch 17/100
1074/1074 [==============================] - 30s 28ms/step - loss: 0.0319 - accuracy: 0.0357 - val_lo
ss: 0.0338 - val_accuracy: 0.0424
Epoch 18/100
1074/1074 [==============================] - 30s 28ms/step - loss: 0.0307 - accuracy: 0.0380 - val_lo
ss: 0.0338 - val_accuracy: 0.0378
Epoch 19/100
1074/1074 [==============================] - 30s 28ms/step - loss: 0.0323 - accuracy: 0.0394 - val_lo
ss: 0.0337 - val_accuracy: 0.0600
Epoch 20/100
1074/1074 [==============================] - 30s 28ms/step - loss: 0.0320 - accuracy: 0.0408 - val_lo
ss: 0.0337 - val_accuracy: 0.0487
Epoch 21/100
1074/1074 [==============================] - 34s 32ms/step - loss: 0.0323 - accuracy: 0.0399 - val_lo
ss: 0.0337 - val_accuracy: 0.0597
Epoch 22/100
1074/1074 [==============================] - 37s 34ms/step - loss: 0.0338 - accuracy: 0.0425 - val_lo
ss: 0.0336 - val_accuracy: 0.0528
Epoch 23/100
1074/1074 [==============================] - 35s 32ms/step - loss: 0.0322 - accuracy: 0.0478 - val_lo
ss: 0.0336 - val_accuracy: 0.0533
Epoch 24/100
1074/1074 [==============================] - 29s 27ms/step - loss: 0.0310 - accuracy: 0.0482 - val_lo
ss: 0.0336 - val_accuracy: 0.0446
Epoch 25/100
1074/1074 [==============================] - 29s 27ms/step - loss: 0.0322 - accuracy: 0.0447 - val_lo
ss: 0.0336 - val_accuracy: 0.0438
Epoch 26/100
1074/1074 [==============================] - 29s 27ms/step - loss: 0.0321 - accuracy: 0.0480 - val_lo
ss: 0.0336 - val_accuracy: 0.0551
Epoch 27/100
1074/1074 [==============================] - 33s 30ms/step - loss: 0.0327 - accuracy: 0.0489 - val_lo
ss: 0.0336 - val_accuracy: 0.0555
Epoch 28/100
1074/1074 [==============================] - 32s 30ms/step - loss: 0.0319 - accuracy: 0.0486 - val_lo
ss: 0.0336 - val_accuracy: 0.0672
Epoch 29/100
1074/1074 [==============================] - 33s 30ms/step - loss: 0.0326 - accuracy: 0.0553 - val_lo
ss: 0.0336 - val_accuracy: 0.0645
Epoch 30/100
1074/1074 [==============================] - 33s 31ms/step - loss: 0.0321 - accuracy: 0.0481 - val_lo
ss: 0.0336 - val_accuracy: 0.0545
Epoch 31/100
1074/1074 [==============================] - 31s 29ms/step - loss: 0.0308 - accuracy: 0.0535 - val_lo
ss: 0.0335 - val_accuracy: 0.0629
Epoch 32/100
1074/1074 [==============================] - 30s 28ms/step - loss: 0.0310 - accuracy: 0.0589 - val_lo
ss: 0.0336 - val_accuracy: 0.0704
Epoch 33/100
1074/1074 [==============================] - 28s 26ms/step - loss: 0.0322 - accuracy: 0.0573 - val_lo
ss: 0.0335 - val_accuracy: 0.0697
Epoch 34/100
1074/1074 [==============================] - 28s 26ms/step - loss: 0.0313 - accuracy: 0.0560 - val_lo
ss: 0.0335 - val_accuracy: 0.0626
Epoch 35/100
1074/1074 [==============================] - 28s 26ms/step - loss: 0.0317 - accuracy: 0.0653 - val_lo
ss: 0.0335 - val_accuracy: 0.0613
Epoch 36/100
1074/1074 [==============================] - 33s 31ms/step - loss: 0.0323 - accuracy: 0.0591 - val_lo
ss: 0.0335 - val_accuracy: 0.0764
Epoch 37/100
1074/1074 [==============================] - 37s 34ms/step - loss: 0.0321 - accuracy: 0.0584 - val_lo
ss: 0.0335 - val_accuracy: 0.0533
Epoch 38/100
1074/1074 [==============================] - 34s 31ms/step - loss: 0.0312 - accuracy: 0.0540 - val_lo
ss: 0.0335 - val_accuracy: 0.0749
Epoch 39/100
1074/1074 [==============================] - 32s 30ms/step - loss: 0.0316 - accuracy: 0.0611 - val_lo
ss: 0.0335 - val_accuracy: 0.0906
Epoch 40/100
1074/1074 [==============================] - 24s 22ms/step - loss: 0.0336 - accuracy: 0.0589 - val_lo
ss: 0.0335 - val_accuracy: 0.0678
Epoch 41/100
1074/1074 [==============================] - 35s 33ms/step - loss: 0.0320 - accuracy: 0.0583 - val_lo
ss: 0.0335 - val_accuracy: 0.0681
Epoch 42/100
1074/1074 [==============================] - 36s 33ms/step - loss: 0.0319 - accuracy: 0.0558 - val_lo
ss: 0.0335 - val_accuracy: 0.0746
Epoch 43/100
1074/1074 [==============================] - 30s 28ms/step - loss: 0.0325 - accuracy: 0.0622 - val_lo
ss: 0.0335 - val_accuracy: 0.0825
Epoch 44/100
1074/1074 [==============================] - 35s 33ms/step - loss: 0.0301 - accuracy: 0.0690 - val_lo
ss: 0.0335 - val_accuracy: 0.0858
Epoch 45/100
1074/1074 [==============================] - 36s 33ms/step - loss: 0.0327 - accuracy: 0.0657 - val_lo
ss: 0.0335 - val_accuracy: 0.0897
Epoch 46/100
1074/1074 [==============================] - 34s 31ms/step - loss: 0.0335 - accuracy: 0.0861 - val_lo
ss: 0.0335 - val_accuracy: 0.0754
Epoch 47/100
1074/1074 [==============================] - 37s 34ms/step - loss: 0.0317 - accuracy: 0.0877 - val_lo
ss: 0.0335 - val_accuracy: 0.0764
Epoch 48/100
1074/1074 [==============================] - 31s 29ms/step - loss: 0.0320 - accuracy: 0.0812 - val_lo
ss: 0.0334 - val_accuracy: 0.0782
Epoch 49/100
1074/1074 [==============================] - 21s 20ms/step - loss: 0.0311 - accuracy: 0.0824 - val_lo
ss: 0.0334 - val_accuracy: 0.0746
Epoch 50/100
1074/1074 [==============================] - 10s 9ms/step - loss: 0.0312 - accuracy: 0.0720 - val_los
s: 0.0334 - val_accuracy: 0.0845
Epoch 51/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0325 - accuracy: 0.0759 - val_los
s: 0.0334 - val_accuracy: 0.0929
Epoch 52/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0324 - accuracy: 0.0791 - val_los
s: 0.0334 - val_accuracy: 0.0948
Epoch 53/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0307 - accuracy: 0.0764 - val_los
s: 0.0334 - val_accuracy: 0.0827
Epoch 54/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0307 - accuracy: 0.0708 - val_los
s: 0.0334 - val_accuracy: 0.0897
Epoch 55/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0328 - accuracy: 0.0896 - val_los
s: 0.0334 - val_accuracy: 0.1127
Epoch 56/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0309 - accuracy: 0.0882 - val_los
s: 0.0334 - val_accuracy: 0.1037
Epoch 57/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0326 - accuracy: 0.0908 - val_los
s: 0.0334 - val_accuracy: 0.1150
Epoch 58/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0304 - accuracy: 0.0870 - val_los
s: 0.0334 - val_accuracy: 0.1021
Epoch 59/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0310 - accuracy: 0.0845 - val_los
s: 0.0334 - val_accuracy: 0.0999
Epoch 60/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0302 - accuracy: 0.0896 - val_los
s: 0.0334 - val_accuracy: 0.1010
Epoch 61/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0319 - accuracy: 0.0959 - val_los
s: 0.0334 - val_accuracy: 0.0832
Epoch 62/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0308 - accuracy: 0.0791 - val_los
s: 0.0334 - val_accuracy: 0.0829
Epoch 63/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0323 - accuracy: 0.0841 - val_los
s: 0.0334 - val_accuracy: 0.0972
Epoch 64/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0319 - accuracy: 0.0823 - val_los
s: 0.0334 - val_accuracy: 0.1083
Epoch 65/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0328 - accuracy: 0.0832 - val_los
s: 0.0334 - val_accuracy: 0.1021
Epoch 66/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0325 - accuracy: 0.0953 - val_los
s: 0.0334 - val_accuracy: 0.1019
Epoch 67/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0331 - accuracy: 0.1001 - val_los
s: 0.0334 - val_accuracy: 0.1152
Epoch 68/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0324 - accuracy: 0.1010 - val_los
s: 0.0334 - val_accuracy: 0.1132
Epoch 69/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0309 - accuracy: 0.0965 - val_los
s: 0.0334 - val_accuracy: 0.1107
Epoch 70/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0305 - accuracy: 0.1023 - val_los
s: 0.0334 - val_accuracy: 0.1075
Epoch 71/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0317 - accuracy: 0.1027 - val_los
s: 0.0334 - val_accuracy: 0.1236
Epoch 72/100
1074/1074 [==============================] - 8s 7ms/step - loss: 0.0308 - accuracy: 0.1029 - val_los
s: 0.0334 - val_accuracy: 0.1141
```

4.4. Predicting/ Reconstructing credit card transactions trainset and testset

In [56]:
```python
predictions_train = autoencoder.predict(X_train)
predictions_val = autoencoder.predict(X_val)
```

4.5. Taking a look at the MSE

In [57]:
```python
from sklearn.metrics import mean_squared_error

mse_val = mean_squared_error(X_val,predictions_val)
mse_train = mean_squared_error(X_train,predictions_train)
print(mse_train)
print(mse_val)
```

```
0.03191511570509188
0.03689375289796546
```

In [58]:
```python
error_train = np.mean(np.power(X_train - predictions_train, 2), axis=1)
limit = np.percentile(error_train, 97.5)
print(limit)
```

```
0.17885801674083593
```

In [59]:
```python
error_val = np.mean(np.power(X_val - predictions_val, 2), axis=1)
fraud = error_val > limit
fraud.sum()
```

Out[59]:
```
175
```

In [60]:
```python
error_val_pd = pd.DataFrame(error_val)
error_val_pd.columns = ['error']
error_val_pd['id'] = range(len(error_val_pd))
error_val_pd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6000 entries, 0 to 5999
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   error   6000 non-null   float64
 1   id      6000 non-null   int32
dtypes: float64(1), int32(1)
memory usage: 70.4 KB
```

In [61]:
```python
from sklearn.metrics import confusion_matrix, precision_recall_curve

confusion_mat = confusion_matrix(y_val, fraud)
print(confusion_mat)
```

```
[[5696  150]
 [ 129   25]]
```

In [62]:
```python
from sklearn.metrics import classification_report
print(classification_report(y_val, fraud))
```

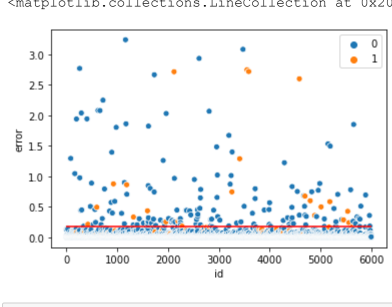
```
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      5846
           1       0.14      0.16      0.15       154

    accuracy                           0.95      6000
   macro avg       0.56      0.57      0.56      6000
weighted avg       0.96      0.95      0.95      6000
```

In [63]:
```python
import seaborn as sns
sns.scatterplot(x=error_val_pd.id, y=error_val_pd.error, hue = np.array(y_val))
plt.hlines(limit, 0, len(error_val), colors='r')
```

Out[63]:
```
<matplotlib.collections.LineCollection at 0x200d1d70250>
```



In [65]:
```python
predictions_test = autoencoder.predict(X_test)
```

In [66]:
```python
error_test = np.mean(np.power(X_test - predictions_test, 2), axis=1)
fraud_test = error_test > limit

confusion_mat = confusion_matrix(y_test, fraud_test)
print(confusion_mat)

print(classification_report(y_test, fraud_test))
```

```
[[3773  103]
 [ 101   23]]
              precision    recall  f1-score   support

           0       0.97      0.97      0.97      3876
           1       0.18      0.19      0.18       124

    accuracy                           0.95      4000
   macro avg       0.58      0.58      0.58      4000
weighted avg       0.95      0.95      0.95      4000
```