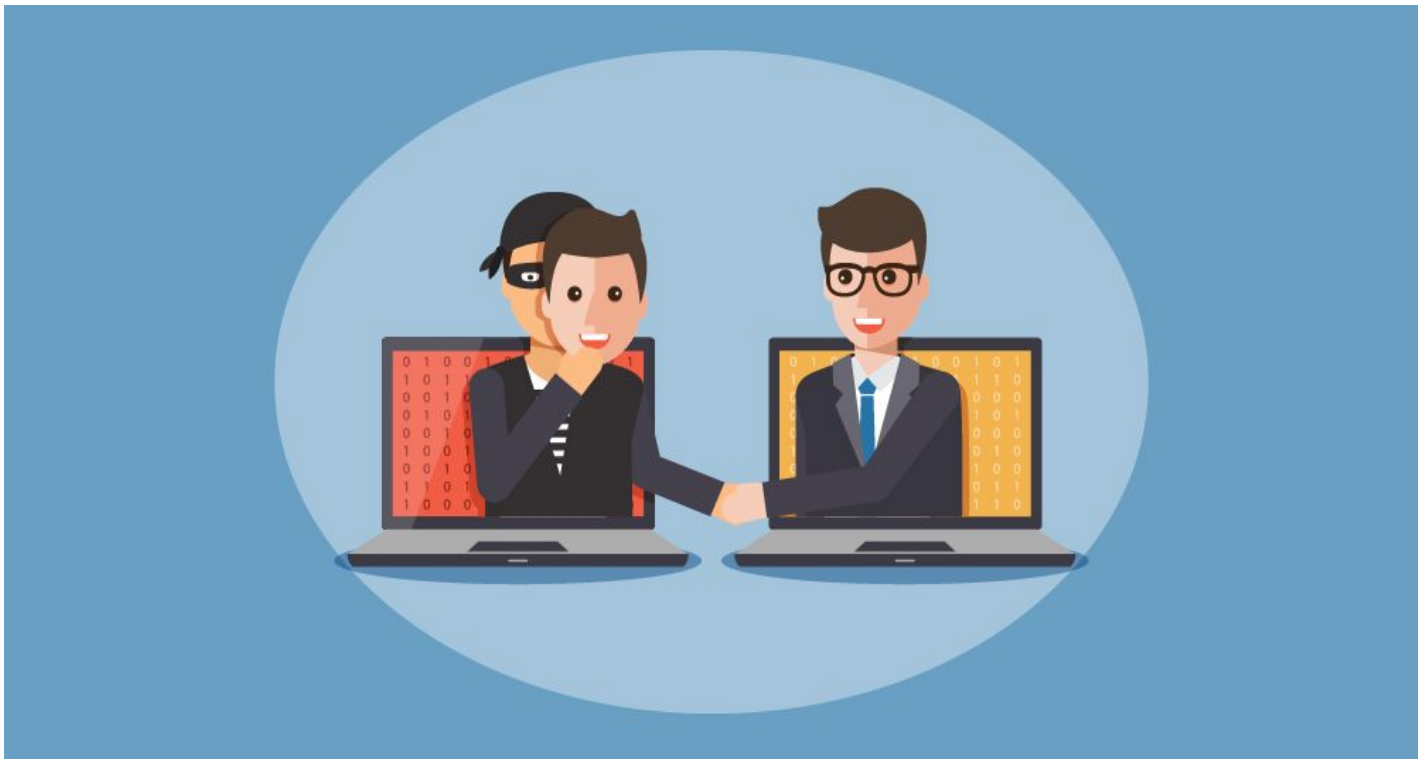
# Project 3



**Subject:**

Transaction Fraud Detection

(Using Autoencoders)

**Author:**

Ghazal Rafiei

Student no. : 97222044

# Gain some inference about the data:

There are 2 files named transaction and identities.

Properties of each file:

Columns of file transaction:

- TransactionDT
- TransactionAMT
- Card1 - card6
- Addr
- Dist
- Email domain and purchaser and seller
- C1-C14
- D1-D15
- M1-M9
- V300-V339
- isFraud

Shape: 590540 rows, 394 columns

Columns of file identity:

- DeviceType
- DeviceInfo
- id01 - d38

Shape: 144233 rows , 40 columns

As the problem definition, CX, DX, MX, and VX columns should be used in training. But ids are about IP, ISP, … which the number itself is not important and if we want to take advantage of them, we should use NLP. So, for now, we put them away.

Also there is column *DeviceInfo* which has too many unique values and it's not sane to use one hot encoding for that.

Is there any null value?

```
[ ] df.isna().sum()

TransactionID        0
isFraud              0
TransactionDT        0
TransactionAmt       0
ProductCD            0
                   ...
V335            508189
V336            508189
V337            508189
V338            508189
V339            508189
Length: 394, dtype: int64
```

```
[ ] df_id.isna().sum()

TransactionID       0
DeviceType       3423
dtype: int64
```

So we have to retrieve null values:
I chose the median method.

Before that, we outer join two dataframes in order not to lose even a single row.
This is the shape after joining: 590540 rows and 395 columns

Then use one hot encoding to have all columns being numeric.

After one hot encoding, there might be created few null columns. So we will omit them in the normalizing process. And this is the shape after everything is done: 590540 rows and 531 columns

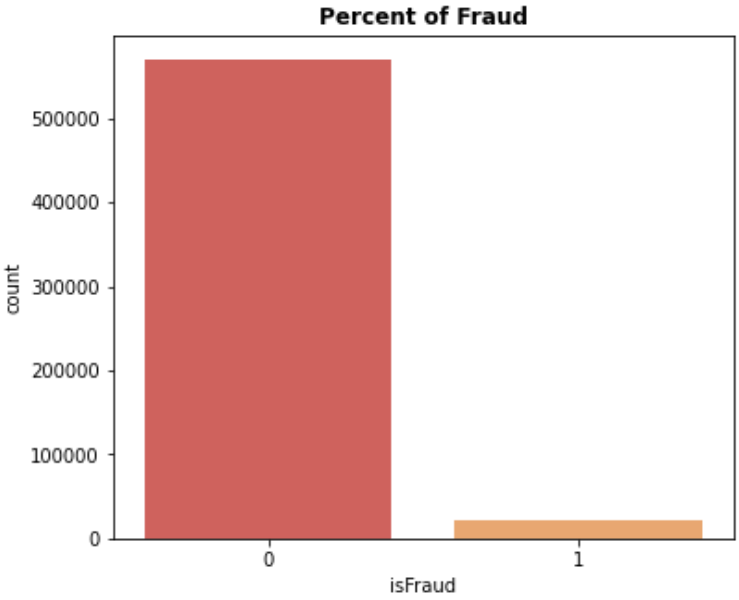
And here's a sample:

| | TransactionDT | TransactionAmt | card1 | card2 | card3 | card5 | addr1 | addr2 | dist1 | dist2 | C1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.000000e+00 | 0.002137 | 0.702653 | 0.435000 | 0.21645 | 0.177215 | 0.398148 | 0.754902 | 0.001847 | 0.003183 | 0.000213 |
| **1** | 6.324658e-08 | 0.000900 | 0.095401 | 0.506667 | 0.21645 | 0.008439 | 0.416667 | 0.754902 | 0.000778 | 0.003183 | 0.000213 |
| **2** | 4.364014e-06 | 0.001840 | 0.199119 | 0.650000 | 0.21645 | 0.278481 | 0.425926 | 0.754902 | 0.027902 | 0.003183 | 0.000213 |

| M4_M1 | M4_M2 | M5_F | M5_T | M6_F | M6_T | M7_F | M7_T | M8_F | M8_T | M9_F | M9_T | DeviceType_desktop | DeviceType_mobile |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.0 | 1.0 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |

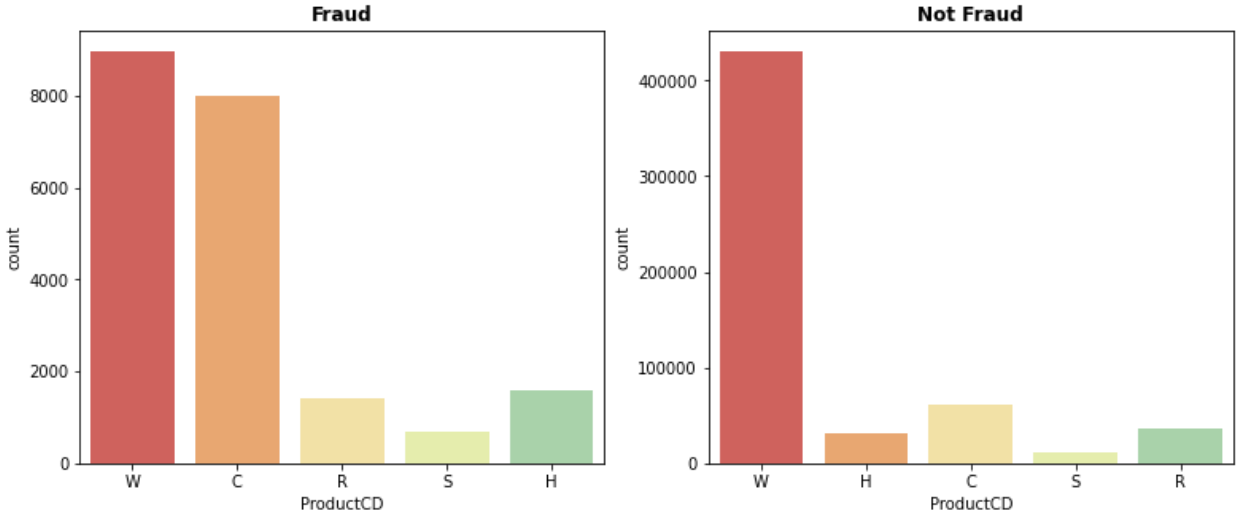
And other columns…

# Visualizations:

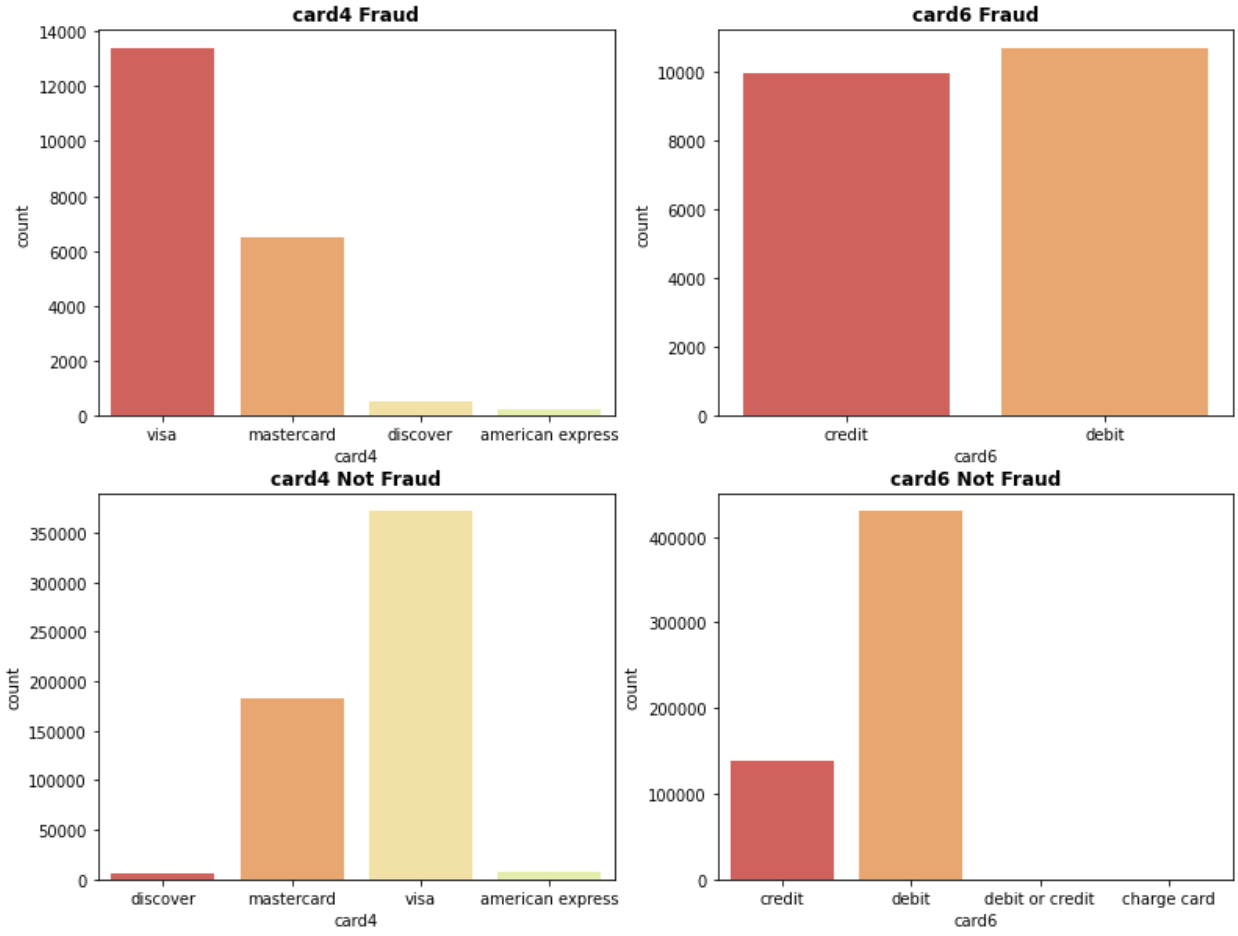


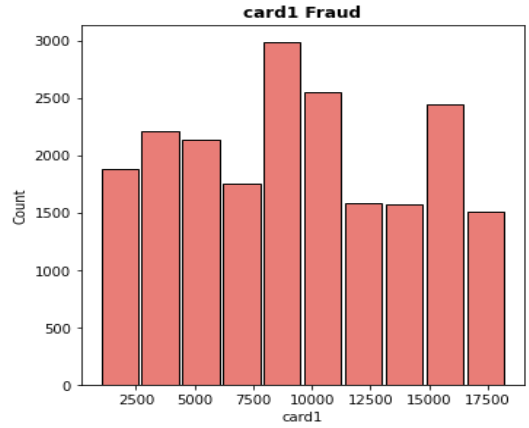

As you can see, there is much less fraud data than not fraud.
So in testing, we shouldn't forget to use equal size of both.

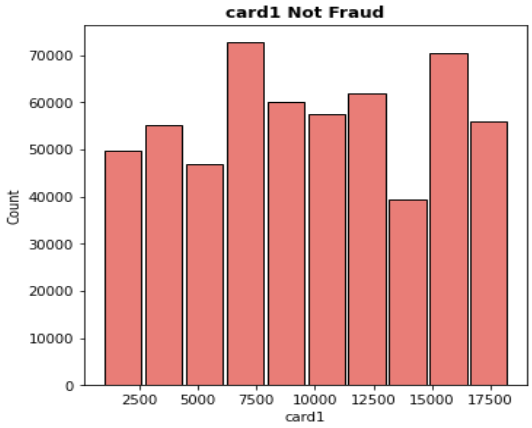
Distribution of Frauds and Not Frauds over kinds of ProductCD:

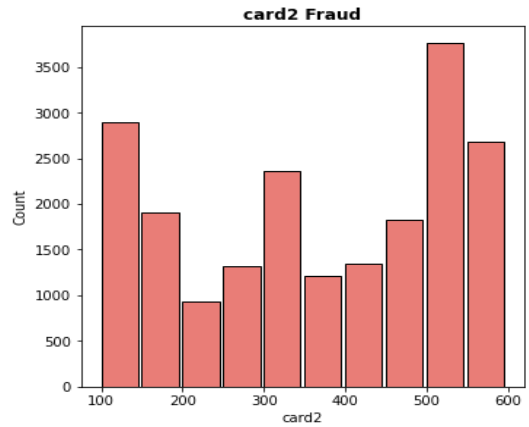


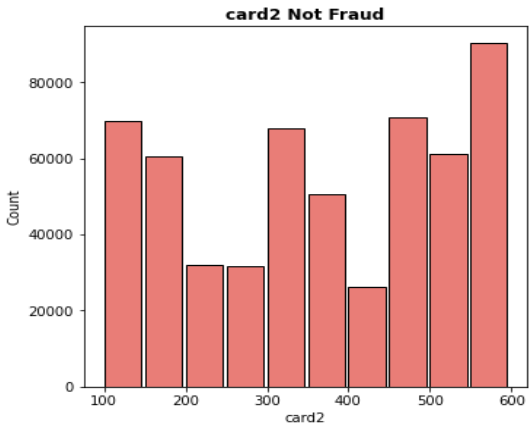
To see if cards are effective or not:



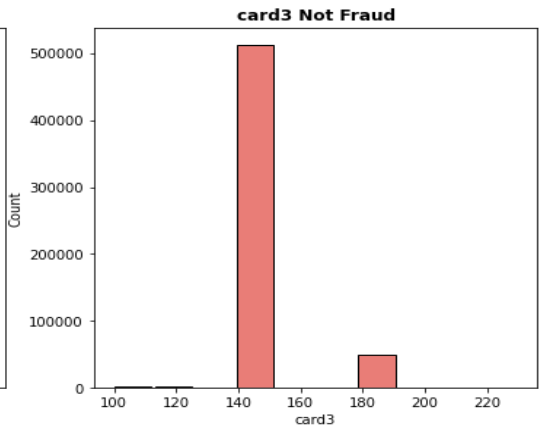

card1 Fraud
Count
card1
0
500
1000
1500
2000
2500
3000
2500
5000
7500
10000
12500
15000
17500

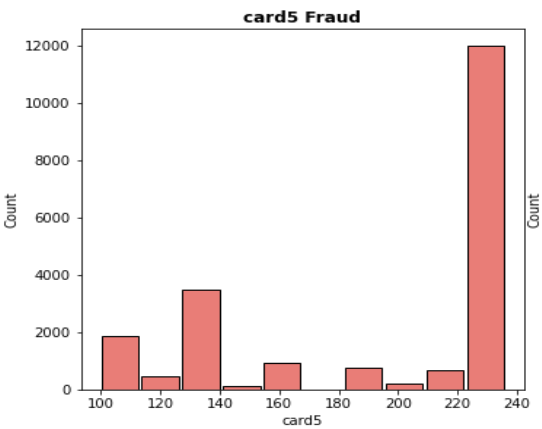

card1 Not Fraud
Count
card1
0
10000
20000
30000
40000
50000
60000
70000
2500
5000
7500
10000
12500
15000
17500

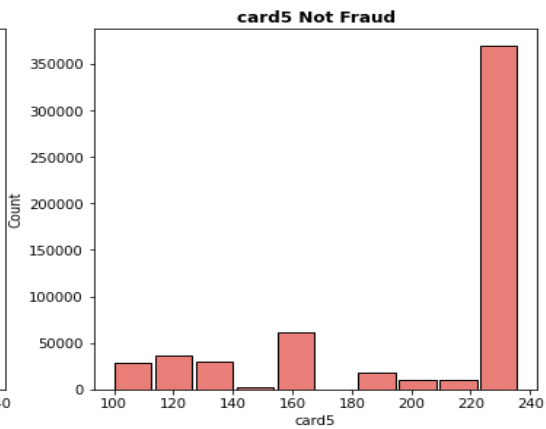

card2 Fraud
Count
card2
0
500
1000
1500
2000
2500
3000
3500
100
200
300
400
500
600


card2 Not Fraud
Count
card2
0
20000
40000
60000
80000
100
200
300
400
500
600

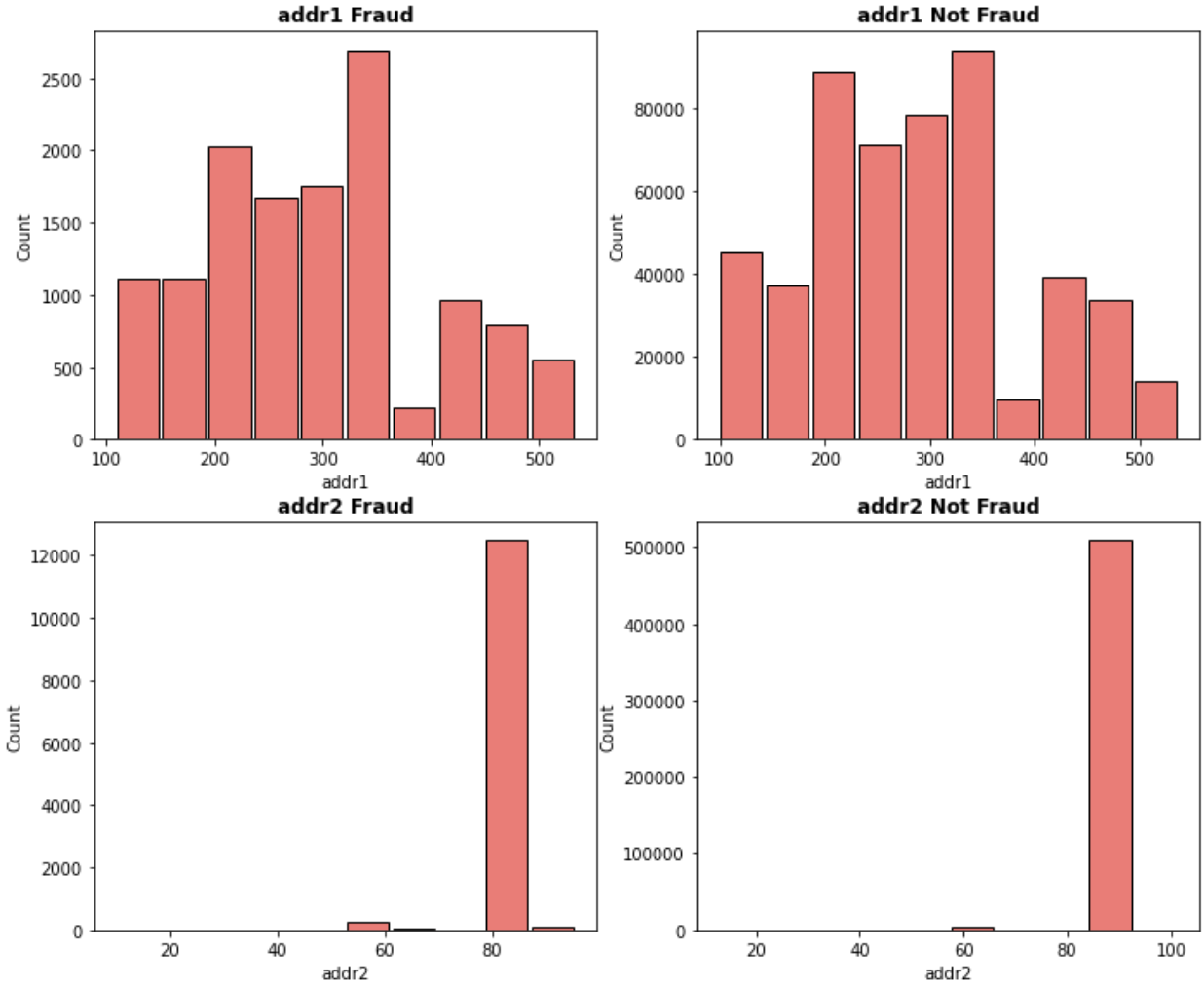

card3 Fraud
Count
card3
0
2000
4000
6000
8000
10000
12000
100
120
140
160
180
200
220


card3 Not Fraud
Count
card3
0
100000
200000
300000
400000
500000
100
120
140
160
180
200
220

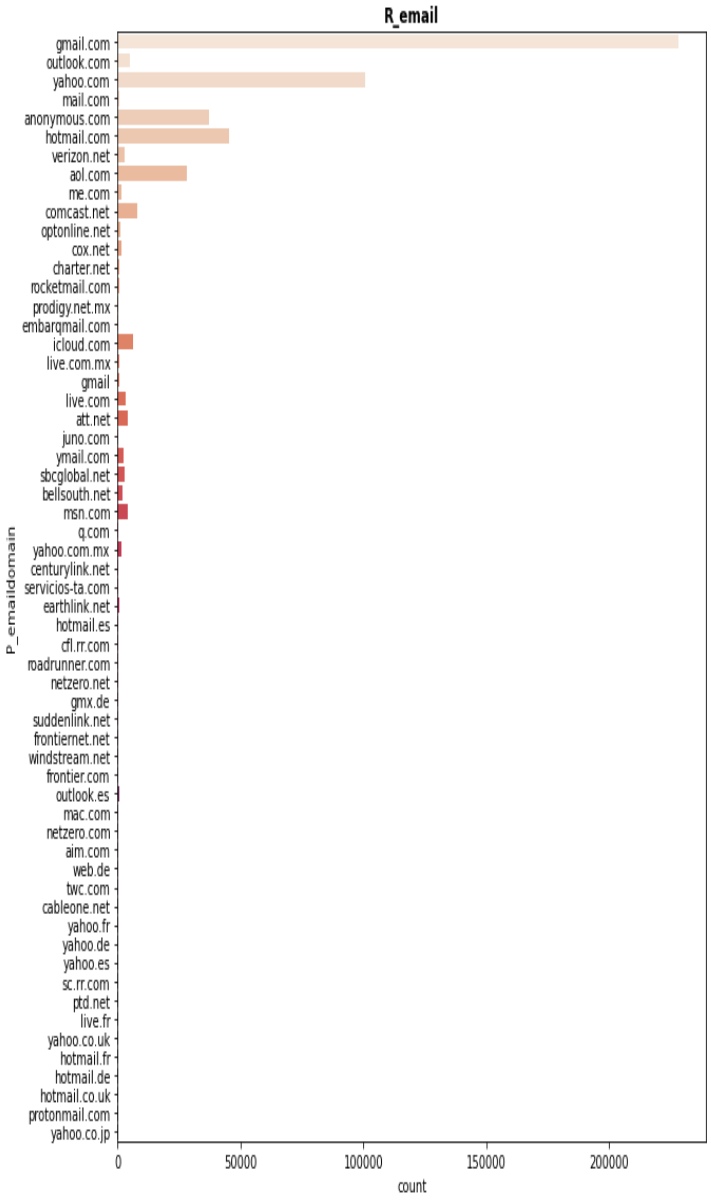

card5 Fraud
Count
card5
0
2000
4000
6000
8000
10000
12000
100
120
140
160
180
200
220
240

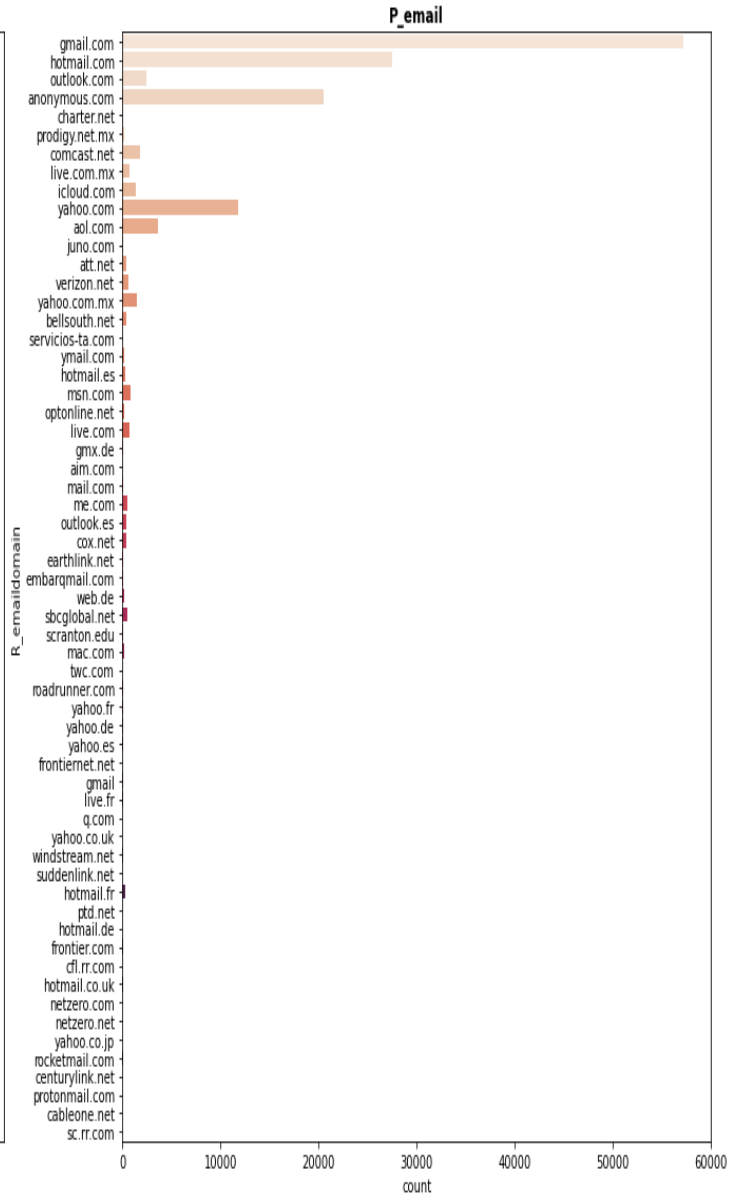

card5 Not Fraud
Count
card5
0
50000
100000
150000
200000
250000
300000
350000
100
120
140
160
180
200
220
240

To see if addr columns are effective in being fraud.

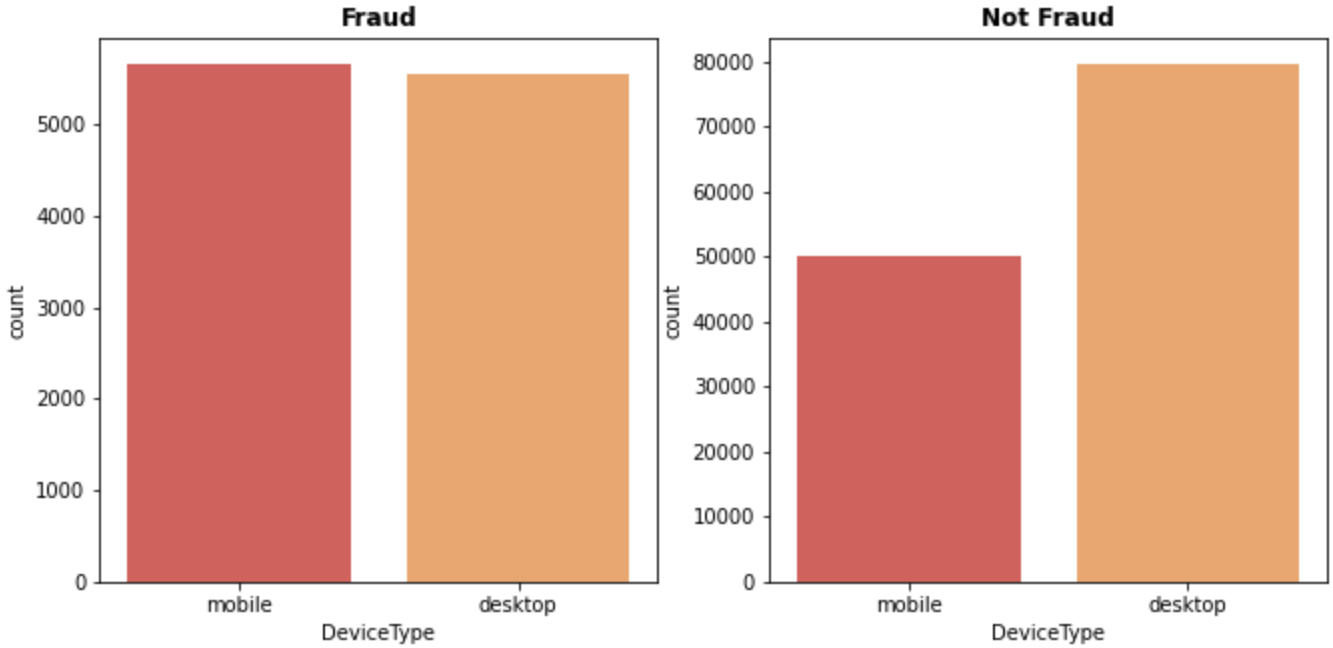


To see if email domains are effective in being fraud.
(for more resolution see the visualization link)

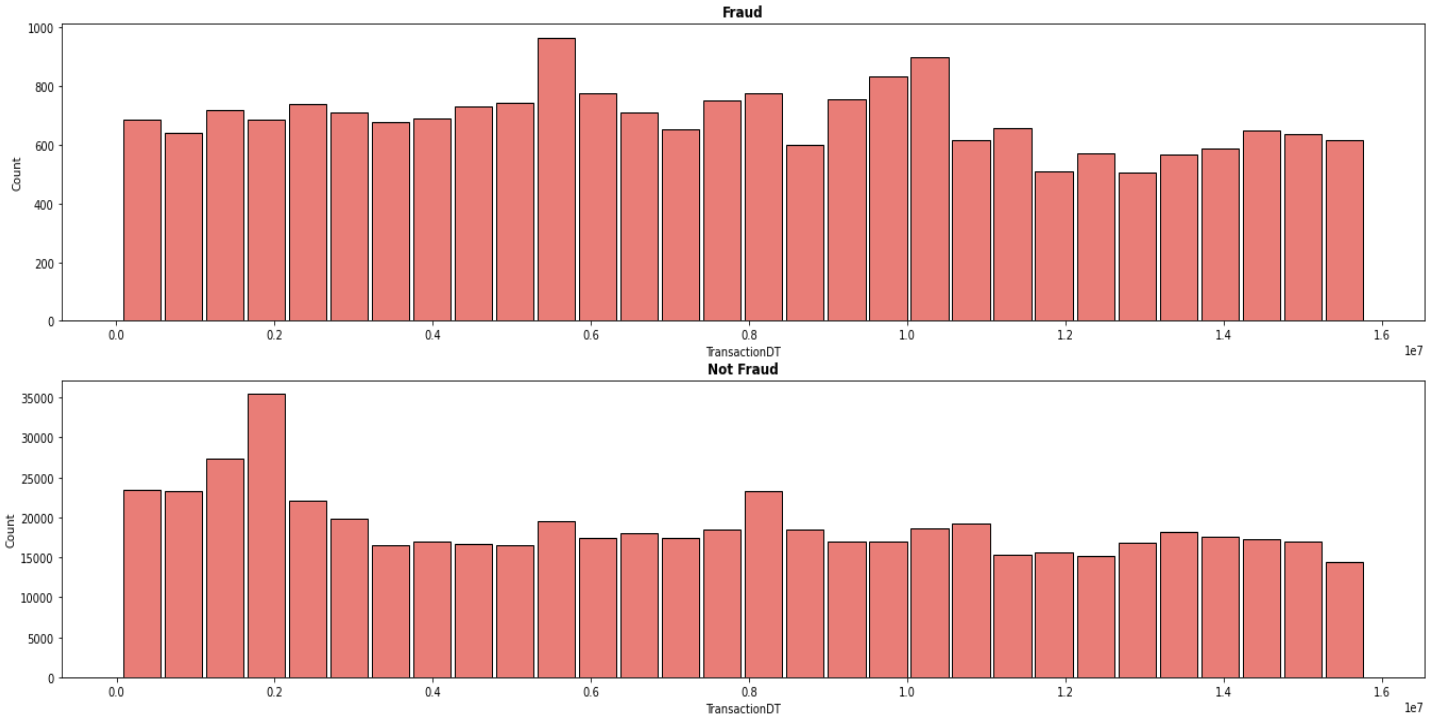


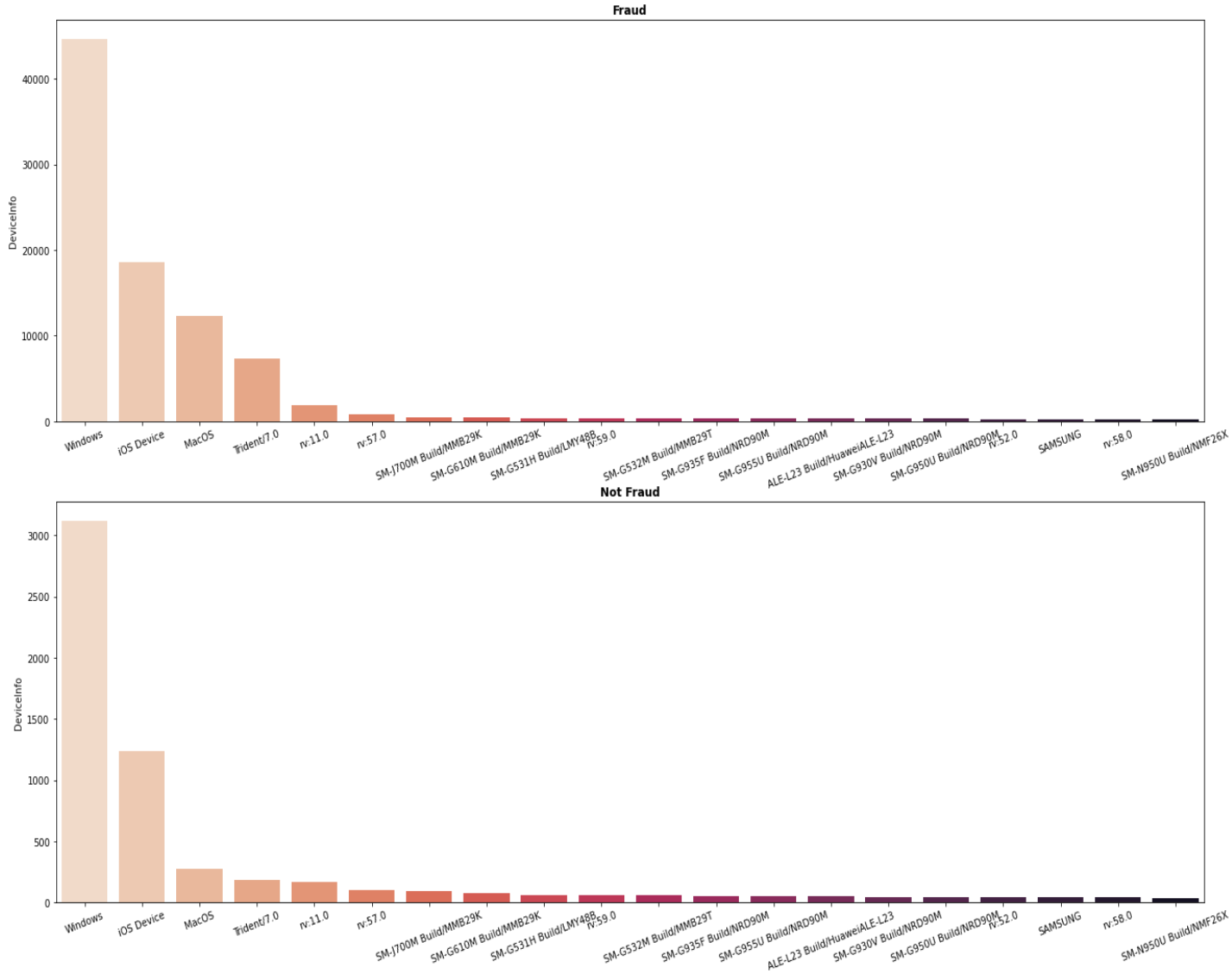
Impact of device type on being Fraud:




Time difference and being Fraud:



Device info which we didn’t use.



## Under Sampling:

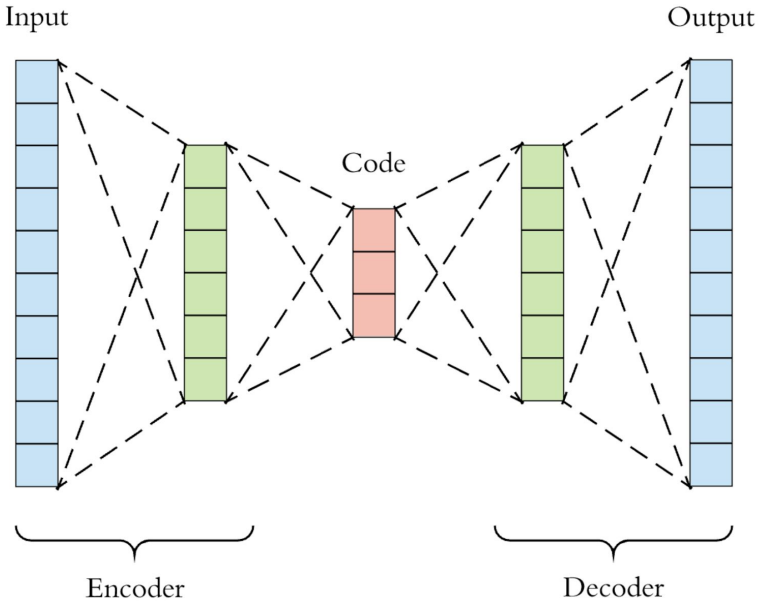
We split the data into three sections:
1- train
2- test
3- validation

## Method:

There's an common approach for anomaly detection which assumes that the data, except fraud, behave in a regular way and the distribution of data has a pattern. But the *fraud* can be anything and mostly are outliers. So this is a different problem than binary classification between fraud and not fraud transactions.

To address this issue, we use autoencoders which can act as an identity function. So if we train the network with only not frauds, the network will imitate not fraud data and then, if we predict the output after giving a fraud data, the loss would be high and that's a point we can distinguish frauds from not frauds.

As you already knew, this a schema of autoencoders:



And this the network we created at first.

```
AutoEncoder(
  (lin1): Linear(in_features=531, out_features=300, bias=True)
  (linb1): BatchNorm1d(300, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin2): Linear(in_features=300, out_features=150, bias=True)
  (linb2): BatchNorm1d(150, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin3): Linear(in_features=150, out_features=75, bias=True)
  (linb3): BatchNorm1d(75, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin4): Linear(in_features=75, out_features=30, bias=True)
  (lin45): Linear(in_features=30, out_features=10, bias=True)
  (linb4): BatchNorm1d(10, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin46): Linear(in_features=10, out_features=30, bias=True)
  (lin5): Linear(in_features=30, out_features=75, bias=True)
  (linb5): BatchNorm1d(75, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin6): Linear(in_features=75, out_features=150, bias=True)
  (lin7): Linear(in_features=150, out_features=300, bias=True)
  (linb6): BatchNorm1d(300, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin8): Linear(in_features=300, out_features=531, bias=True)
  (drop2): Dropout(p=0.03, inplace=False)
)
```

Layer Sizes: 531, 300, 150, 75, 30 , 10, 30, 75, 300, 531

All the activation functions I didn't mention are ReLU and optimizers are Adam with lr = 0.1 because they had better results. But others are tested and shown below.

There are some models I tried but there were more. I didn't bring them all.

1- Optimizer: Adam lr =0.01 (The best)

```
Epoch: 122   Loss: 0.0008
Epoch: 123   Loss: 0.0008
Epoch: 124   Loss: 0.0008
Epoch: 125   Loss: 0.0008
Epoch: 126   Loss: 0.0008  Val_Loss on correct set: 0.0007
Epoch: 127   Loss: 0.0007
Epoch: 128   Loss: 0.0008
Epoch: 129   Loss: 0.0007
Epoch: 130   Loss: 0.0008
Epoch: 131   Loss: 0.0008  Val_Loss on correct set: 0.0007
Epoch: 132   Loss: 0.0008
Epoch: 133   Loss: 0.0008
Epoch: 134   Loss: 0.0008
Epoch: 135   Loss: 0.0008
Epoch: 136   Loss: 0.0008  Val_Loss on correct set: 0.0007
Epoch: 137   Loss: 0.0007
Epoch: 138   Loss: 0.0008
Epoch: 139   Loss: 0.0007
Epoch: 140   Loss: 0.0008
Epoch: 141   Loss: 0.0008  Val_Loss on correct set: 0.0007
Epoch: 142   Loss: 0.0007
Epoch: 143   Loss: 0.0007
Epoch: 144   Loss: 0.0007
Epoch: 145   Loss: 0.0007
Epoch: 146   Loss: 0.0007  Val_Loss on correct set: 0.0007
Epoch: 147   Loss: 0.0008
Epoch: 148   Loss: 0.0007
Epoch: 149   Loss: 0.0007
Epoch: 150   Loss: 0.0007
```

2- Optimizer: SGD, lr = 0.01

```
Epoch: 79   Loss: 0.0085
Epoch: 80   Loss: 0.0086
Epoch: 81   Loss: 0.0085
Epoch: 82   Loss: 0.0085
Epoch: 83   Loss: 0.0086
Epoch: 84   Loss: 0.0086
Epoch: 85   Loss: 0.0084
Epoch: 86   Loss: 0.0085
Epoch: 87   Loss: 0.0084
Epoch: 88   Loss: 0.0085
Epoch: 89   Loss: 0.0084
Epoch: 90   Loss: 0.0085
Epoch: 91   Loss: 0.0084
Epoch: 92   Loss: 0.0084
Epoch: 93   Loss: 0.0084
Epoch: 94   Loss: 0.0084
Epoch: 95   Loss: 0.0084
Epoch: 96   Loss: 0.0084
Epoch: 97   Loss: 0.0085
Epoch: 98   Loss: 0.0084
Epoch: 99   Loss: 0.0084
Epoch: 100   Loss: 0.0084
Epoch: 101   Loss: 0.0085
Epoch: 102   Loss: 0.0084
Epoch: 103   Loss: 0.0083
Epoch: 104   Loss: 0.0083
Epoch: 105   Loss: 0.0084
Epoch: 106   Loss: 0.0084
Epoch: 107   Loss: 0.0083
Epoch: 108   Loss: 0.0083
Epoch: 109   Loss: 0.0083
Epoch: 110   Loss: 0.0083
Epoch: 111   Loss: 0.0083
Epoch: 112   Loss: 0.0083
Epoch: 113   Loss: 0.0083
Epoch: 114   Loss: 0.0082
Epoch: 115   Loss: 0.0082
Epoch: 116   Loss: 0.0082
Epoch: 117   Loss: 0.0082
Epoch: 118   Loss: 0.0082
```

3- Optimizer: SGD, lr = 0.02

```
Epoch: 119   Loss: 0.0098
Epoch: 120   Loss: 0.0099
Epoch: 121   Loss: 0.0099  Val_Loss on correct set: 0.0087
Epoch: 122   Loss: 0.0099
Epoch: 123   Loss: 0.0097
Epoch: 124   Loss: 0.0098
Epoch: 125   Loss: 0.0097
Epoch: 126   Loss: 0.0097  Val_Loss on correct set: 0.0086
Epoch: 127   Loss: 0.0096
Epoch: 128   Loss: 0.0097
Epoch: 129   Loss: 0.0096
Epoch: 130   Loss: 0.0096
Epoch: 131   Loss: 0.0096  Val_Loss on correct set: 0.0086
Epoch: 132   Loss: 0.0096
Epoch: 133   Loss: 0.0096
Epoch: 134   Loss: 0.0096
Epoch: 135   Loss: 0.0096
Epoch: 136   Loss: 0.0094  Val_Loss on correct set: 0.0085
Epoch: 137   Loss: 0.0094
Epoch: 138   Loss: 0.0095
Epoch: 139   Loss: 0.0095
Epoch: 140   Loss: 0.0095
Epoch: 141   Loss: 0.0095  Val_Loss on correct set: 0.0085
Epoch: 142   Loss: 0.0094
Epoch: 143   Loss: 0.0095
Epoch: 144   Loss: 0.0094
Epoch: 145   Loss: 0.0094
Epoch: 146   Loss: 0.0093  Val_Loss on correct set: 0.0084
Epoch: 147   Loss: 0.0094
Epoch: 148   Loss: 0.0094
Epoch: 149   Loss: 0.0094
Epoch: 150   Loss: 0.0093
```

4- Optimizer: SGD, lr =  0.05

```
Epoch: 28  Loss: 0.0086
Epoch: 29  Loss: 0.0086
Epoch: 30  Loss: 0.0087  Val_Loss on correct set: 0.0079
Epoch: 31  Loss: 0.0086
Epoch: 32  Loss: 0.0084
Epoch: 33  Loss: 0.0086
Epoch: 34  Loss: 0.0084
Epoch: 35  Loss: 0.0086  Val_Loss on correct set: 0.0078
Epoch: 36  Loss: 0.0086
Epoch: 37  Loss: 0.0084
Epoch: 38  Loss: 0.0084
Epoch: 39  Loss: 0.0083
Epoch: 40  Loss: 0.0085  Val_Loss on correct set: 0.0078
Epoch: 41  Loss: 0.0084
Epoch: 42  Loss: 0.0085
Epoch: 43  Loss: 0.0083
Epoch: 44  Loss: 0.0083
Epoch: 45  Loss: 0.0083  Val_Loss on correct set: 0.0077
Epoch: 46  Loss: 0.0082
Epoch: 47  Loss: 0.0082
Epoch: 48  Loss: 0.0082
Epoch: 49  Loss: 0.0083
Epoch: 50  Loss: 0.0082  Val_Loss on correct set: 0.0077
Epoch: 51  Loss: 0.0082
Epoch: 52  Loss: 0.0081
Epoch: 53  Loss: 0.0082
Epoch: 54  Loss: 0.0081
Epoch: 55  Loss: 0.0082  Val_Loss on correct set: 0.0076
Epoch: 56  Loss: 0.0081
Epoch: 57  Loss: 0.0081
Epoch: 58  Loss: 0.0080
Epoch: 59  Loss: 0.0080
Epoch: 60  Loss: 0.0080  Val_Loss on correct set: 0.0076
Epoch: 61  Loss: 0.0081
Epoch: 62  Loss: 0.0080
Epoch: 63  Loss: 0.0080
Epoch: 64  Loss: 0.0083
Epoch: 65  Loss: 0.0079  Val_Loss on correct set: 0.0075
```

4- Activatio Functions: Sigmoid and Tanh
Layer Sizes:  531, 400, 300, 150, 75, 30, 10, 30, 75, 150, 300, 400, 531

```
Epoch: 80  Loss: 0.0019  Val_Loss on correct set: 0.0017
Epoch: 81  Loss: 0.0018
Epoch: 82  Loss: 0.0018
Epoch: 83  Loss: 0.0017
Epoch: 84  Loss: 0.0017
Epoch: 85  Loss: 0.0016  Val_Loss on correct set: 0.0015
Epoch: 86  Loss: 0.0016
Epoch: 87  Loss: 0.0079
Epoch: 88  Loss: 0.0070
Epoch: 89  Loss: 0.0061
Epoch: 90  Loss: 0.0052  Val_Loss on correct set: 0.0050
Epoch: 91  Loss: 0.0046
Epoch: 92  Loss: 0.0042
Epoch: 93  Loss: 0.0039
Epoch: 94  Loss: 0.0036
Epoch: 95  Loss: 0.0032  Val_Loss on correct set: 0.0029
Epoch: 96  Loss: 0.0029
Epoch: 97  Loss: 0.0027
Epoch: 98  Loss: 0.0033
Epoch: 99  Loss: 0.0024
Epoch: 100  Loss: 0.0039  Val_Loss on correct set: 0.0025
Epoch: 101  Loss: 0.0022
Epoch: 102  Loss: 0.0021
Epoch: 103  Loss: 0.0021
Epoch: 104  Loss: 0.0022
Epoch: 105  Loss: 0.0020  Val_Loss on correct set: 0.0019
Epoch: 106  Loss: 0.0019
```

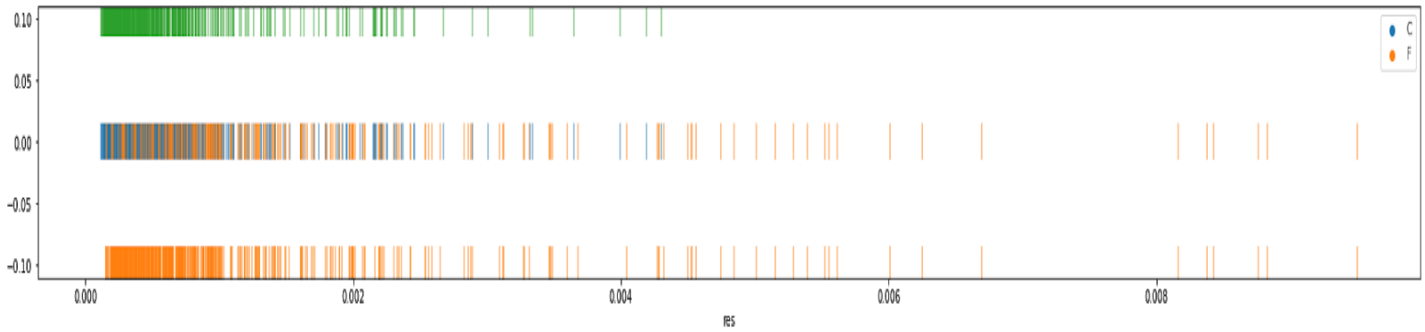
5- 6 neurons bottle neck instead of 10
Layer Sizes: 531, 400, 300, 150, 75, 30, 6, 30, 75, 150, 300, 400, 531

```
Epoch: 73   Loss: 0.0023
Epoch: 74   Loss: 0.0022
Epoch: 75   Loss: 0.0022  Val_Loss on correct set: 0.0020
Epoch: 76   Loss: 0.0036
Epoch: 77   Loss: 0.0026
Epoch: 78   Loss: 0.0024
Epoch: 79   Loss: 0.0023
Epoch: 80   Loss: 0.0022  Val_Loss on correct set: 0.0019
Epoch: 81   Loss: 0.0021
Epoch: 82   Loss: 0.0021
Epoch: 83   Loss: 0.0093
Epoch: 84   Loss: 0.0088
Epoch: 85   Loss: 0.0084  Val_Loss on correct set: 156489565.3896
Epoch: 86   Loss: 0.0081
Epoch: 87   Loss: 0.0079
Epoch: 88   Loss: 0.0076
Epoch: 89   Loss: 0.0074
Epoch: 90   Loss: 0.0073  Val_Loss on correct set: 11583192.1229
Epoch: 91   Loss: 0.0071
Epoch: 92   Loss: 0.0070
Epoch: 93   Loss: 0.0066
Epoch: 94   Loss: 0.0063
Epoch: 95   Loss: 0.0060  Val_Loss on correct set: 424.7983
Epoch: 96   Loss: 0.0058
Epoch: 97   Loss: 0.0073
Epoch: 98   Loss: 0.0058
Epoch: 99   Loss: 0.0054
Epoch: 100   Loss: 0.0052  Val_Loss on correct set: 0.0063
Epoch: 101   Loss: 0.0052
Epoch: 102   Loss: 0.0050
Epoch: 103   Loss: 0.0046
Epoch: 104   Loss: 0.0046
Epoch: 117   Loss: 0.0028
Epoch: 118   Loss: 0.0027
Epoch: 119   Loss: 0.0026
Epoch: 120   Loss: 0.0025  Val_Loss on correct set: 0.0022
Epoch: 121   Loss: 0.0025
Epoch: 122   Loss: 0.0024
Epoch: 123   Loss: 0.0841
Epoch: 124   Loss: 0.0080
Epoch: 125   Loss: 0.0072  Val_Loss on correct set: 0.0072
Epoch: 126   Loss: 0.0066
Epoch: 127   Loss: 0.0060
Epoch: 128   Loss: 0.0054
Epoch: 129   Loss: 0.0050
Epoch: 130   Loss: 0.0047  Val_Loss on correct set: 0.0044
Epoch: 131   Loss: 0.0046
Epoch: 132   Loss: 0.0045
Epoch: 133   Loss: 0.0041
Epoch: 134   Loss: 0.0037
Epoch: 135   Loss: 0.0034  Val_Loss on correct set: 0.0031
Epoch: 136   Loss: 0.0032
Epoch: 137   Loss: 0.0032
Epoch: 138   Loss: 0.0030
Epoch: 139   Loss: 0.0033
Epoch: 140   Loss: 0.0027  Val_Loss on correct set: 0.0025
Epoch: 141   Loss: 0.0027
Epoch: 142   Loss: 0.0025
Epoch: 143   Loss: 0.0025
Epoch: 144   Loss: 0.0024
Epoch: 145   Loss: 0.0023  Val_Loss on correct set: 0.0021
Epoch: 146   Loss: 0.0023
Epoch: 147   Loss: 0.0022
Epoch: 148   Loss: 0.0021
Epoch: 149   Loss: 0.0021
Epoch: 150   Loss: 0.0023  Val_Loss on correct set: 0.0019
```

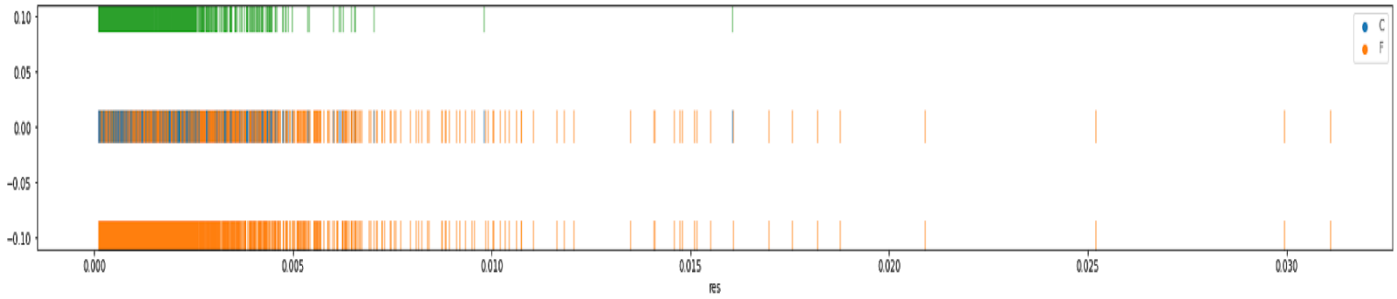
To sum up these models, the first one was the best and achieved the least loss 0.0007. Others are failed because of low convergence speed or not stability in converging or even not accomplishing the result better than the first one.

After training the network, we should see the loss value for frauds and not frauds.

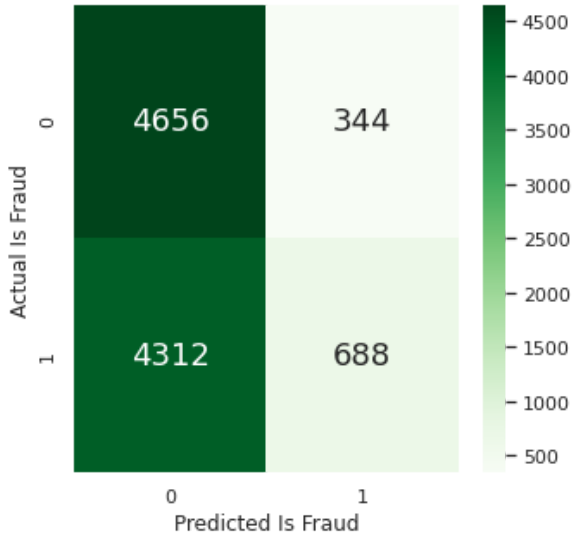
On trained not fraud and test fraud:



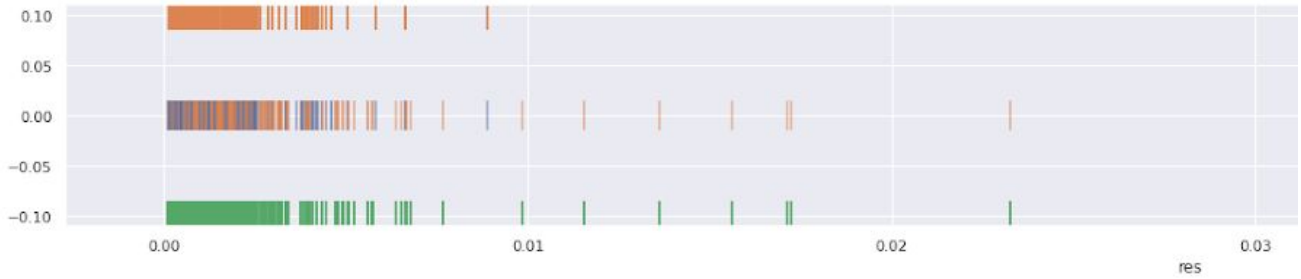

On test and validation fraud and not fraud:

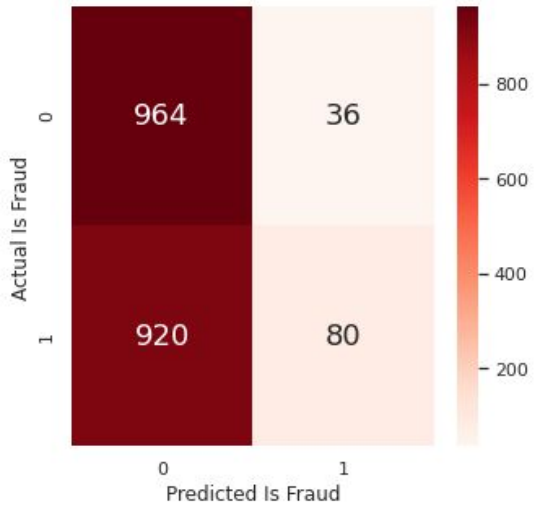



Here I used separator 0.002 which before and after that, we should have not fraud and fraud.



Validation:




As you can see, the final results, our idea didn't go well as we expected. However you can see a slight difference between loss values for frauds(upper) and not frauds(lower).(and the middle plot is for overlapping both plots). For example not fraud data are more compressed to the left while frauds are more scattered and sparse.(just a little :)) )

So we can clearly predict not fraud data, not the frauds. So we need stronger models which separate them all.

## Troubleshooting (In Second Notebook):

I think one of the reasons the model did not work was for having large dimensions. If we used at most 10 or even up to 20 columns, the network could decide sooner and better. Because most of the columns are coming from one hot encoding and that's like a sparse matrix. So if we substitute 0 for all of them, there wouldn't come a huge difference.

So let's work on this idea in extra time!!! :))

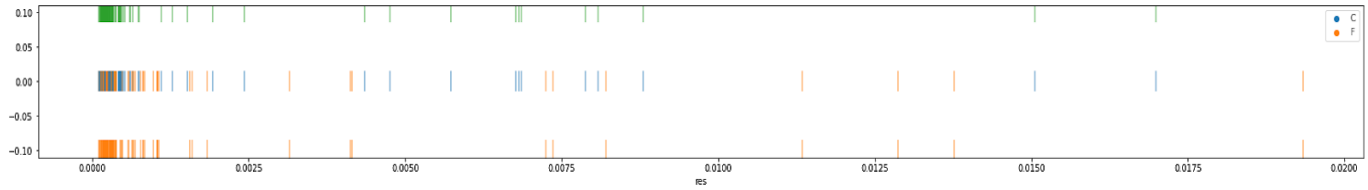
- TransactionID isFraud
- TransactionAmt
- TransactionDT
- ProductCD
- Card1-Card6
- Addr1-addr2
- dist1
- P_emaildomain
- R_emaildomain

```
AutoEncoder(
  (lin1): Linear(in_features=143, out_features=100, bias=True)
  (linb1): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin2): Linear(in_features=100, out_features=60, bias=True)
  (linb2): BatchNorm1d(60, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin3): Linear(in_features=60, out_features=30, bias=True)
  (linb3): BatchNorm1d(30, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin4): Linear(in_features=30, out_features=15, bias=True)
  (lin45): Linear(in_features=15, out_features=6, bias=True)
  (linb4): BatchNorm1d(6, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin46): Linear(in_features=6, out_features=15, bias=True)
  (lin5): Linear(in_features=15, out_features=30, bias=True)
  (linb5): BatchNorm1d(30, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin6): Linear(in_features=30, out_features=60, bias=True)
  (lin7): Linear(in_features=60, out_features=100, bias=True)
  (linb6): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (lin8): Linear(in_features=100, out_features=143, bias=True)
  (drop2): Dropout(p=0.03, inplace=False)
)
```

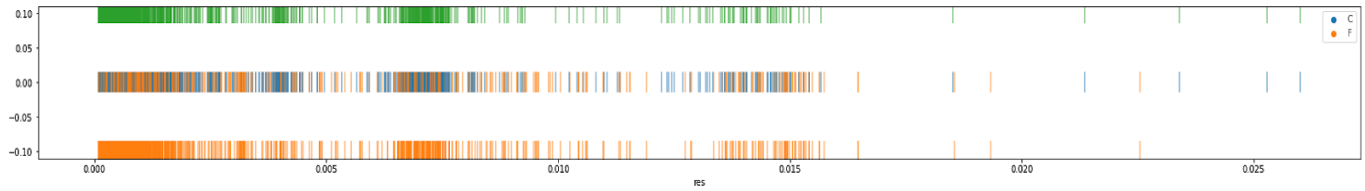
Result with the same network but layers: 143, 100, 60, 30, 15, 6, 15, 30, 60, 100, 143

```
.
Epoch: 80   Loss: 0.0031  Val_Loss on correct set: 0.0009
Epoch: 81   Loss: 0.0043
Epoch: 82   Loss: 0.0030
Epoch: 83   Loss: 0.0031
Epoch: 84   Loss: 0.0029
Epoch: 85   Loss: 0.0030  Val_Loss on correct set: 0.0009
Epoch: 86   Loss: 0.0031
Epoch: 87   Loss: 0.0030
Epoch: 88   Loss: 0.0029
Epoch: 89   Loss: 0.0033
Epoch: 90   Loss: 0.0029  Val_Loss on correct set: 0.0009
Epoch: 91   Loss: 0.0033
Epoch: 92   Loss: 0.0028
Epoch: 93   Loss: 0.0035
Epoch: 94   Loss: 0.0031
Epoch: 95   Loss: 0.0028  Val_Loss on correct set: 0.0009
Epoch: 96   Loss: 0.0028
Epoch: 97   Loss: 0.0029
Epoch: 98   Loss: 0.0031
Epoch: 99   Loss: 0.0029
Epoch: 100   Loss: 0.0029  Val_Loss on correct set: 0.0009
```

Train:




Validation:




Data is not separated very well, again!