Credit Card Fraud Detection

Context

It is important that credit card companies are able to recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Content

The datasets contains transactions made by credit cards in September 2013 by european cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-senstive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Dataset Link https://www.kaggle.com/mlg-ulb/creditcardfraud/data (https://www.kaggle.com/mlg-ulb/creditcardfraud/data)

In [8]:

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

In [9]:

```
df = pd.read_csv("creditcard.csv")
df.head()
```

Out[9]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [8]:

```
df.tail()
```

Out[8]:

	Time	V1	V2	V3	V4	V5	V6	V7
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

5 rows × 31 columns

In [9]:

```
df.shape
```

Out[9]:

```
(284807, 31)
```

In [10]:

```
#For value counts of both the classes
x = df['Class'].value_counts()
print("Value Counts:")
print(x)
y = df.shape
fraudper = (x[1]/y[0])*100
print(fraudper,"% of fraud points only")
```

```
Value Counts:
0    284315
1       492
Name: Class, dtype: int64
0.172748563062 % of fraud points only
```

Extremely Unbalanced Dataset

In [6]:

```
df.describe()
```

Out[6]:

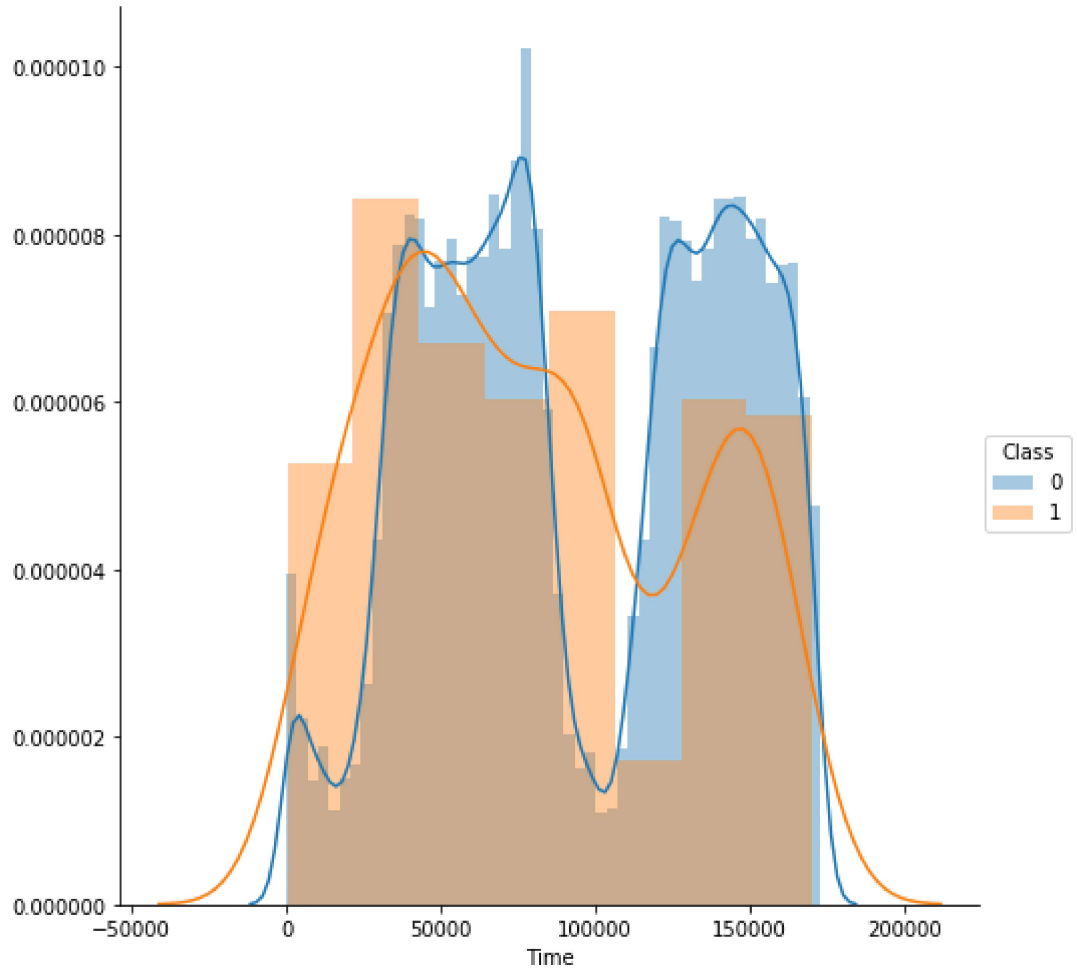
	Time	V1	V2	V3	V4	V
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+0
mean	94813.859575	1.165980e-15	3.416908e-16	-1.373150e-15	2.086869e-15	9.604066e-1
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+0
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+0
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-0
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-0
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-0
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+0

8 rows × 31 columns

Dataset has all numerical features of which amount and time is given rest all 28 features are anonymized Principal Components transformed usign PCA. The output class can be either 1(in case of fraud) and otherwise 0.

In [7]:

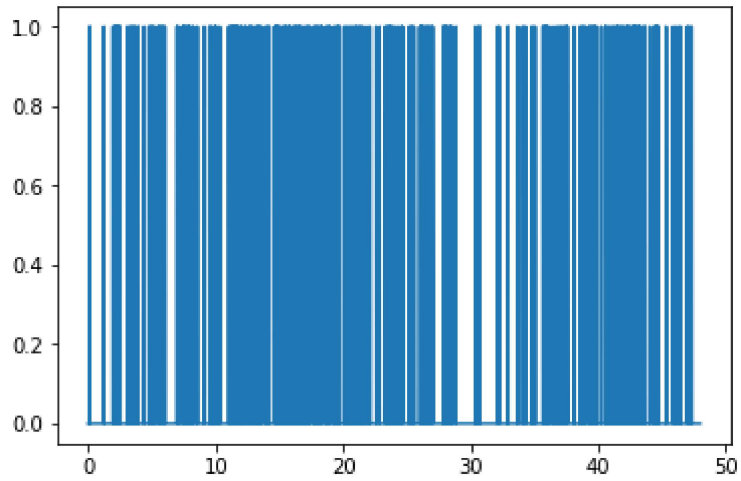
```
#Plotting according to Amount
(sns.FacetGrid(df,hue="Class",size=7)
 .map(sns.distplot,"Time")
 .add_legend())

plt.show()
```



Using time we can't distinguish and find the fraudulent transactions as it happens anytime of the day

In [8]:

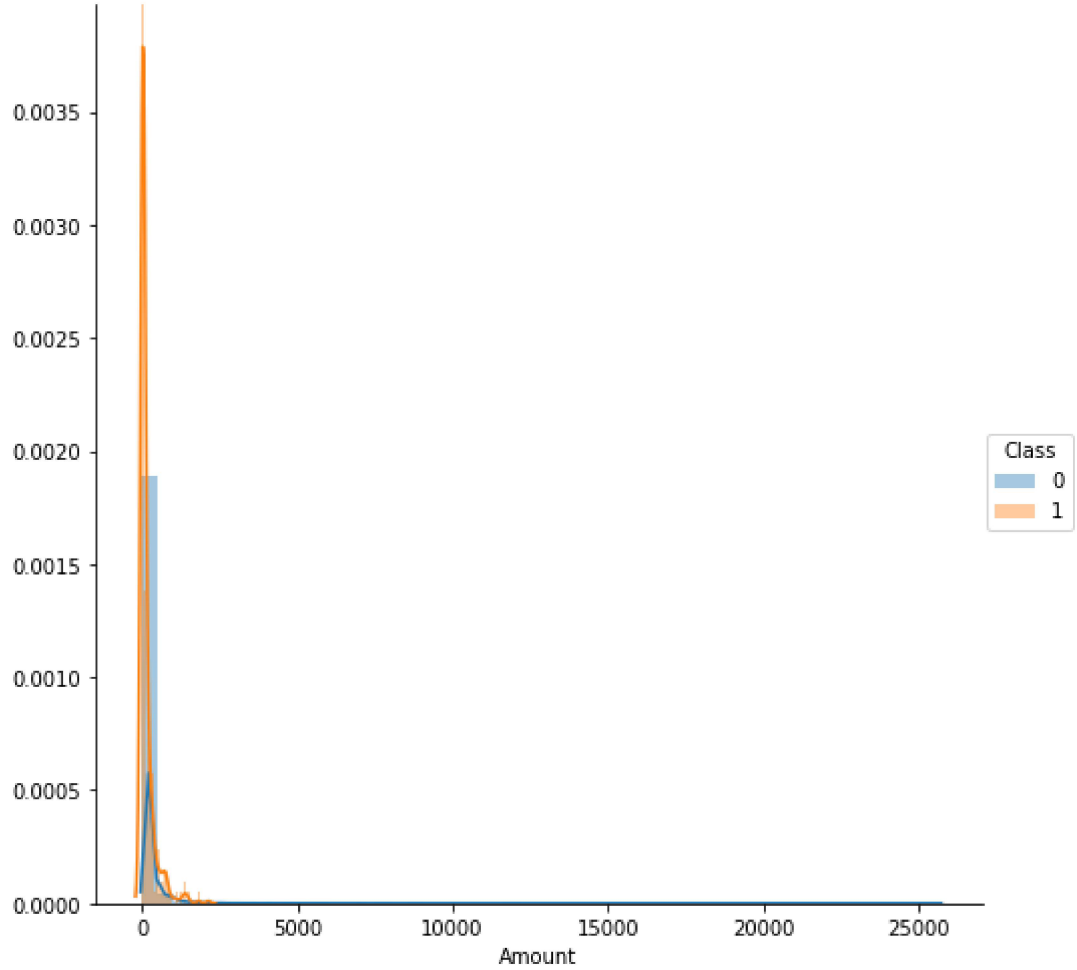
```
def sectohours(sec):
    hours = sec/(60*60)
    return hours
hours = sectohours(df['Time'])
plt.plot(hours,df['Class'])
plt.show()
```



Frauds happened between all the hours and most of them did not happen at a particular time

In [9]:

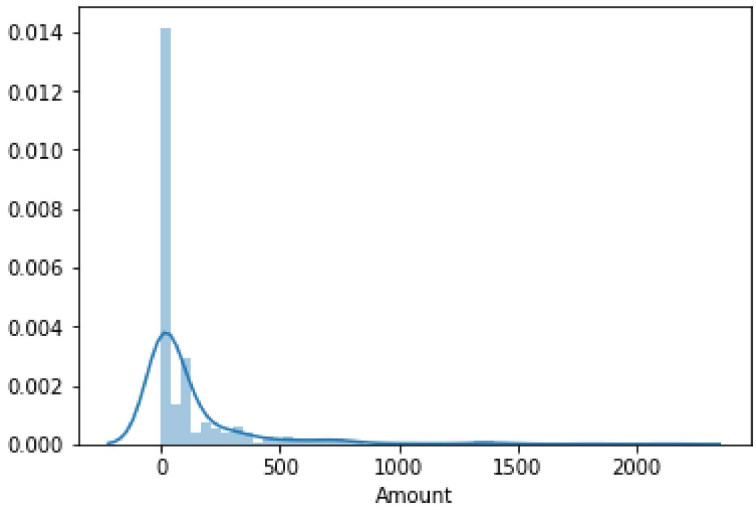
```
#Plotting according to Amount
(sns.FacetGrid(df,hue="Class",size=7)
 .map(sns.distplot,"Amount")
 .add_legend())
plt.show()
```



Most of the transactions are below 2000 dollars for both fraud and normal transactions

In [10]:

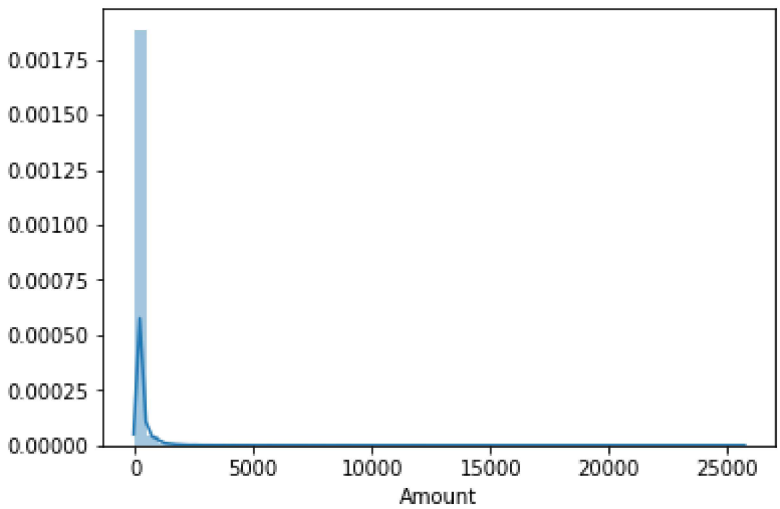
```
fraud=df.loc[df["Class"]==1]
normal=df.loc[df["Class"]==0]
sns.distplot(fraud.Amount)
plt.show()
```



All of the fraud transactions are below 2400 dollars and Most of them are below 500$

In [11]:

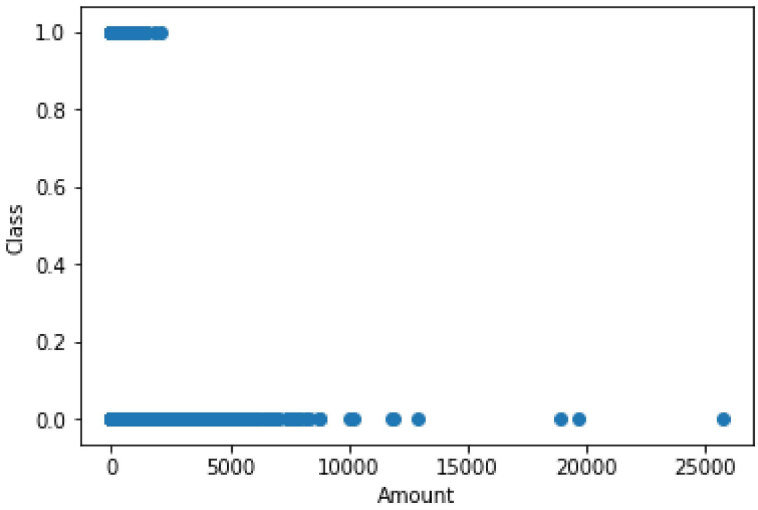
```
sns.distplot(normal.Amount)
plt.show()
```



We can observe that normal transactions are bit more spread than the fraud transactions in terms of amount and the non-fraud transactions mostly below 1000$

In [12]:

```
plt.scatter(df['Amount'],df['Class'])
plt.xlabel("Amount")
plt.ylabel("Class")
plt.show()
```

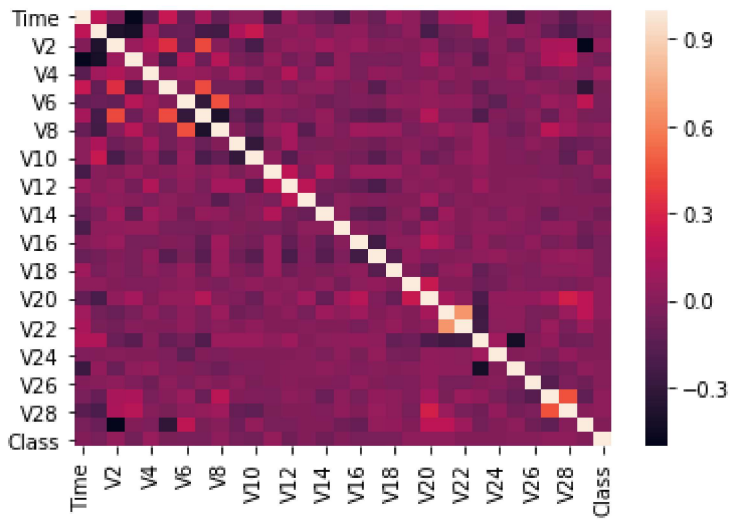


From the scatter plot we can obeserve that only few points in normal transactions are above 10,000$

Correlation between features

In [13]:

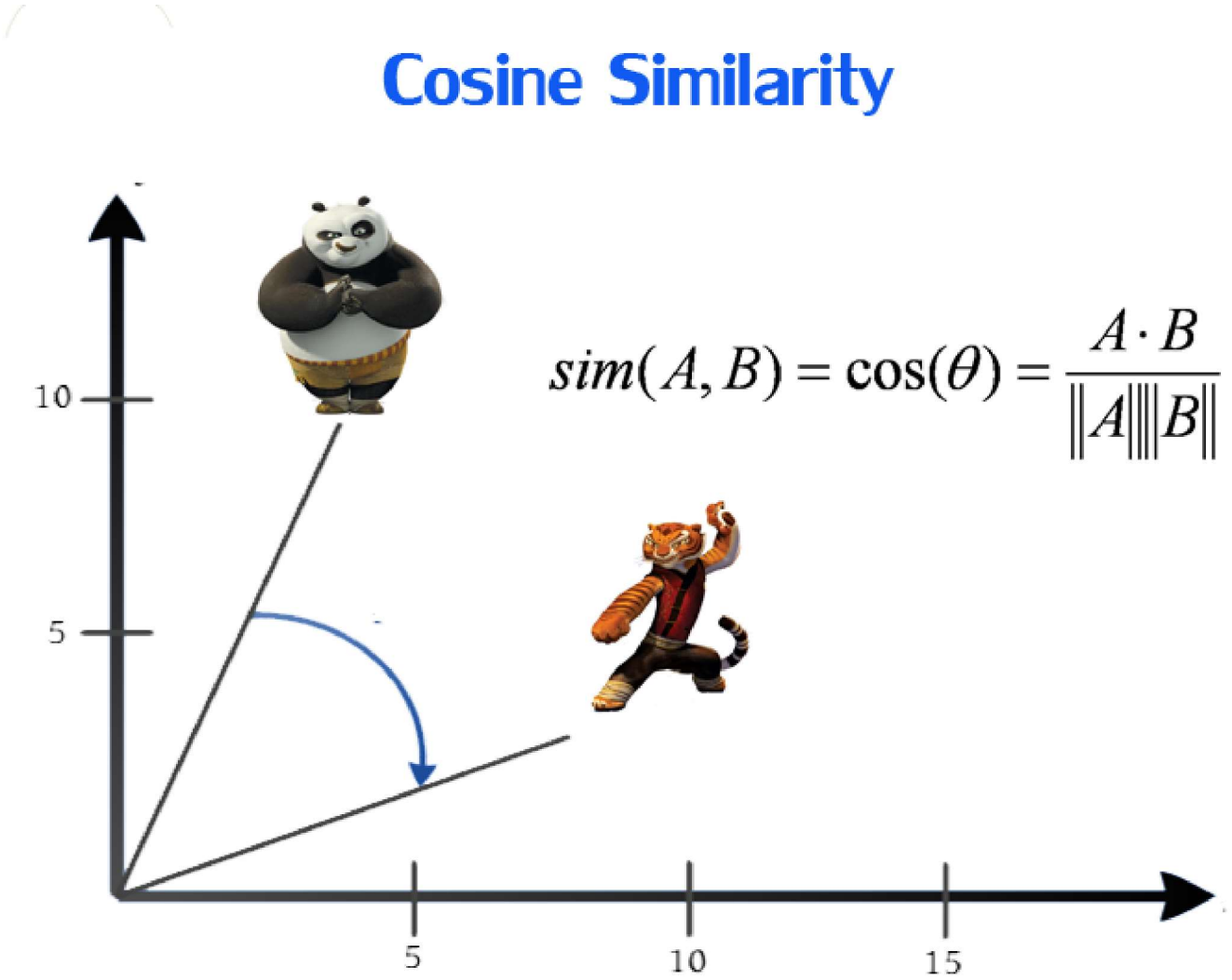
```
#Using Spearman Correlation Coefficient to see how much the features are co-related to each
ax = sns.heatmap(df.corr(method='spearman'))
plt.show()
```



It seems the features are quite not related to each other as no points are dark orange and very few are light orange which means few slightly related

Task 2 (Similarity between transactions)

As the defined metric is cosine similarity
similarity(i,j) = cosine^-1(dot product (vi, vj) / (length(vi) * length(vj)))



In [37]:

```
#Sampling 100 random datapoints/rows from the table
sample100 = df.sample(100)
sample100.shape
```

Out[37]:

(100, 31)

In [38]:

```
sample100.head()
```

Out[38]:

	Time	V1	V2	V3	V4	V5	V6	V7	
27318	34497.0	1.036074	-1.010511	0.657139	-1.298448	-1.397930	-0.473776	-0.709625	0.
134093	80644.0	1.177076	0.578651	-0.376805	1.066586	0.454467	-0.208963	0.125942	0.
51948	45204.0	-0.742751	0.475642	1.266253	0.514362	-0.310715	1.112718	1.079678	0.
69529	53466.0	0.893483	-0.561434	-0.353750	0.198708	-0.574481	-1.185392	0.481672	-0.
31057	36192.0	-1.209178	0.846116	1.968886	-0.207012	-0.170817	0.592541	-0.032847	0

5 rows × 31 columns

In [39]:

```
#Indexes of the 100 sampled datavalues
sample100.index
```

Out[39]:

```
Int64Index([ 27318, 134093,  51948,  69529,  31057, 158472, 109986, 121403,
            230955, 208947,  63480,  26663, 272567, 223494, 283225, 274475,
            259774, 282306, 247580, 155729, 198896, 138592, 277810, 220537,
              5655, 275474, 124930,  94814, 217284, 154813,   7008, 210570,
            110415, 195673, 241653, 255369, 117565, 114814, 198441, 142232,
             40548, 156958, 190047,  25005, 221028, 262655, 197710, 145542,
             81941, 130238,  96112, 154059,  85865, 260128,  53704,  96740,
            277398, 241984, 257688,  71482, 202244, 245132, 144935, 106556,
            136020, 160001,   3184,   4645,  69415, 222984,  79306, 135917,
             87235,  57524, 248662, 183472, 169656, 186667, 145869, 119259,
             73815, 262966,  33942, 261553, 273838, 203968,  46666,  27417,
              5904,  88477, 274079, 128640, 178974,  67663, 249376, 199384,
             15256, 136605, 220884, 168041],
           dtype='int64')
```

In [40]:

```
#importing cosine similarity to calculate cosine similarity of each data point with the eve
from sklearn.metrics.pairwise import cosine_similarity
similar = cosine_similarity(sample100)
# similar = cosine_similarity(df.iloc[188683:188694])
print(similar.shape)
```

```
(100, 100)
```

In [41]:

```
#Cosine Similarity Matrix
print(similar)
```

```
[[ 1.          0.99999552  0.99999899 ...,  0.99999555  0.99996431
   0.99999578]
 [ 0.99999552  1.          0.99999031 ...,  1.          0.99993458
   0.99999999]
 [ 0.99999899  0.99999031  1.         ...,  0.99999037  0.99997523
   0.99999071]
 ...,
 [ 0.99999555  1.          0.99999037 ...,  1.          0.99993471
   0.99999999]
 [ 0.99996431  0.99993458  0.99997523 ...,  0.99993471  1.          0.999935
61]
 [ 0.99999578  0.99999999  0.99999071 ...,  0.99999999  0.99993561  1.
  ]]
```

In [42]:

```
similar.shape
```

Out[42]:

```
(100, 100)
```

In [43]:

```
similar.min()
```

Out[43]:

```
0.99993443667853621
```

In [46]:

```
similarity_arr = []
#zip to iterate through to arrays parallelly
for trans_array,head in zip(similar,sample100.index):
    l = []
    #trans_array is the list with similarity values of all points w.r.t. head
    #Making a tuple as (transaction_id,similarity with head)
    for trans_elt,index in zip(trans_array,sample100.index):
        l.append((index,trans_elt))

    #Sort the transactions according to the similarity(ascending order)
    new_list = sorted(l,key=lambda x:x[1],reverse=True)
    #Creating a tuple (head,list of tuple of (transaction_id,similarity with head))
    similarity_arr.append((head,new_list))

similarity_arr[:2]
```

Out[46]:

```
[(27318,
  [(27318, 1.0000000000000004),
   (25005, 0.99999998467392459),
   (248662, 0.9999999432867771),
   (222984, 0.99999985448014339),
   (7008, 0.99999976616694319),
   (69529, 0.99999973360959293),
   (124930, 0.999999672456517),
   (67663, 0.99999936655299659),
   (110415, 0.99999913631860271),
   (255369, 0.99999907864641457),
   (220537, 0.99999901363107224),
   (51948, 0.99999899456223529),
   (53704, 0.99999880409545416),
   (198441, 0.99999873561849151),
   (183472, 0.99999866597597276),
   (46666, 0.9999983991072513),
   (119259, 0.99999836819905874),
```

In [47]:

```
#Printing the top 10 least similar transactions with respect to each transaction.
for trans in similarity_arr:
    print("--------------------------------------")
    print("Given transactions id=",trans[0],"Class:",int(sample100.loc[trans[0]].Class))
    print("--------------------------------------")
    print("Similar Transations:")
    print("--------------------")
    for elt in trans[1][:10]:
        print("Transaction id:=",elt[0],"Similarity:",elt[1],"Class:",int(sample100.loc[elt
    print("\n")
```

```
--------------------------------------
Given transactions id= 27318 Class: 0
--------------------------------------
Similar Transations:
--------------------
Transaction id:= 27318 Similarity: 1.0 Class: 0
Transaction id:= 25005 Similarity: 0.999999984674 Class: 0
Transaction id:= 248662 Similarity: 0.999999943287 Class: 0
Transaction id:= 222984 Similarity: 0.99999985448 Class: 0
Transaction id:= 7008 Similarity: 0.999999766167 Class: 0
Transaction id:= 69529 Similarity: 0.99999973361 Class: 0
Transaction id:= 124930 Similarity: 0.999999672457 Class: 0
Transaction id:= 67663 Similarity: 0.999999366553 Class: 0
Transaction id:= 110415 Similarity: 0.999999136319 Class: 0
Transaction id:= 255369 Similarity: 0.999999078646 Class: 0


--------------------------------------
Given transactions id= 134093 Class: 0
```