Credit Card Fraud Detection

Task 1:

Exploratory Data Analysis of the Dataset

Dataset & Attribute Information:

The dataset contains transactions made by credit cards in September 2013 by european cardholders. It contains only numerical input variables and due to confidentiality issues, it 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. Finally there are two classes that each transaction falls into. A fradulent one represented 1 or non-fradulent one represented by 0.

Importing the data and relevant libraries:

In [1]:

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

import warnings                         # In order to supress any warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
cc = pd.read_csv('creditcard.csv') # Downloaded the dataset and saved it in the local directory.
cc['Class'] = cc['Class'].map({0:'Non_Fraudulent',1:'Fraudulent'}) # Changing the class value to a more easier notation.
```

In [3]:

```
cc.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.

5 rows × 31 columns

Basic Analysis:

In [4]:

```
# Shape of the data
cc.shape
```

Out[4]:

```
(284807, 31)
```

In [5]:

```
# The columns of the data
cc.columns
```

Out[5]:

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Class'],
      dtype='object')
```

In [6]:

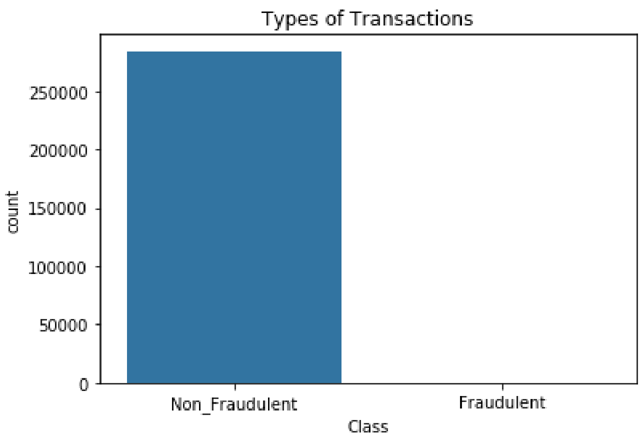
```
# The number of datapoints belonging to each class
cc.Class.value_counts()
```

Out[6]:

```
Non_Fraudulent    284315
Fraudulent           492
Name: Class, dtype: int64
```

In [7]:

```
sns.countplot('Class',data = cc).set_title('Types of Transactions')
plt.show()
```



Clearly, this is a highly imbalanced dataset.

Here, the feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. This doesn't say anything about the time taken by each transaction individually. However, it can be calculated from the data.

In [8]:

```
times = [0] # Since the first transaction took place right away.

for i in range(1,len(cc)):
        times.append(cc.Time[i]-cc.Time[i-1])

times = np.array(times) # Since numpy arrays are more efficient than lists
print(times[:5])
```

```
[0. 0. 1. 0. 1.]
```

```
[0. 0. 1. 0. 1.]
```

In [9]:

```
# The reason we are modifying the dataframe is to simplify the analysis process.
cc.insert(30,'Transaction_Time',times)
```

In [10]:

```
cc.columns # Updated Dataframe
```

Out[10]:

```
Index(['Time', 'V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
       'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
       'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount',
       'Transaction_Time', 'Class'],
      dtype='object')
```

In [11]:

```
# Basic information regarding the labeled features.
print(cc.Amount.describe())
print('\n')
print(cc.Time.describe())
print('\n')
print(cc.Transaction_Time.describe())
```

```
count    284807.000000
mean         88.349619
std         250.120109
min           0.000000
25%           5.600000
50%          22.000000
75%          77.165000
max       25691.160000
Name: Amount, dtype: float64


count    284807.000000
mean      94813.859575
std       47488.145955
min           0.000000
25%       54201.500000
50%       84692.000000
75%      139320.500000
max      172792.000000
Name: Time, dtype: float64


count    284807.000000
mean          0.606699
std           1.053380
min           0.000000
25%           0.000000
50%           0.000000
75%           1.000000
max          32.000000
Name: Transaction_Time, dtype: float64
```

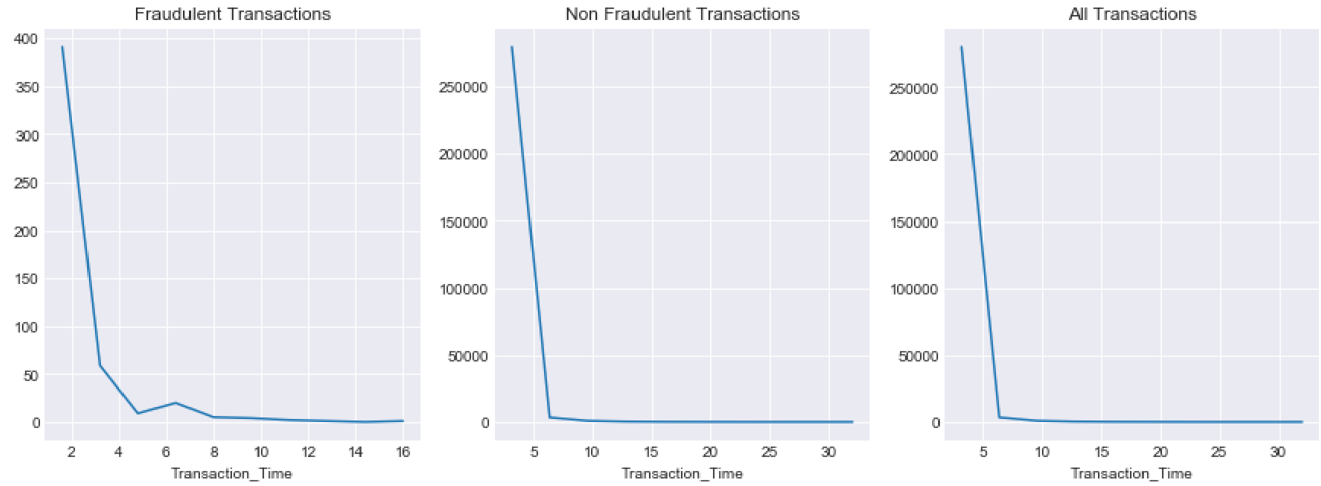
In [12]:

```
# Now, let's create subsets of the DataFrame based on their class.
FT = cc[cc['Class'] == 'Fraudulent'] # FT = Fraudulent Transactions
NFT = cc[cc['Class'] == 'Non_Fraudulent'] # NFT = Non Fraudulent Transactions
```

In [13]:

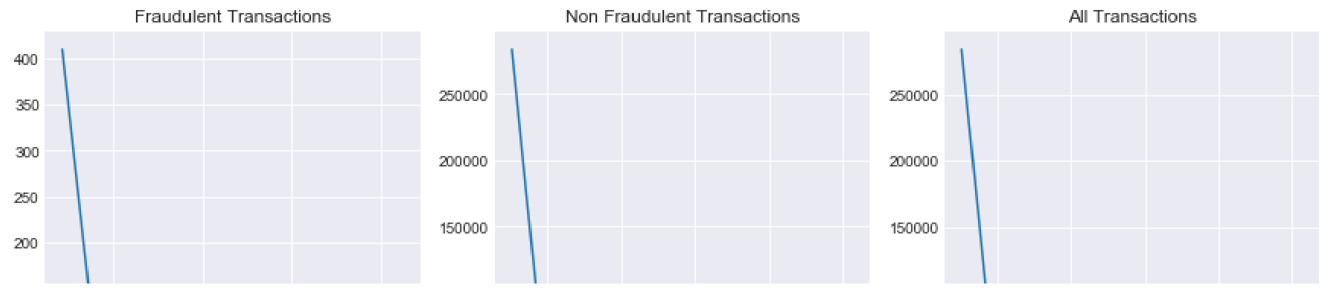
```
# Let us plot the PDFs for each labeled feature and compare them.
# Transaction Time
sns.set_style('darkgrid')
plt.figure(figsize=(15,5))
```

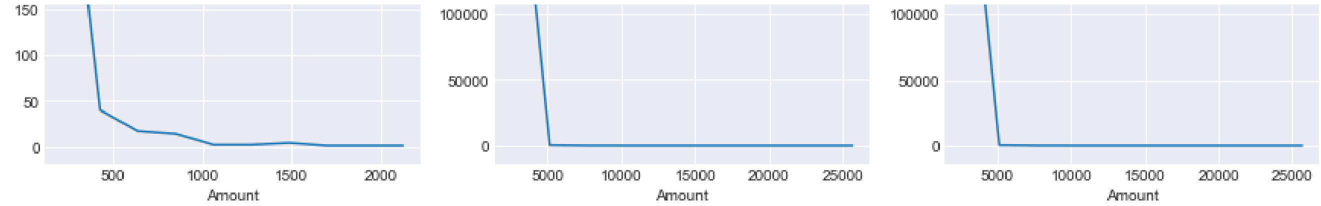
```
plt.subplot(131)
counts, bin_edges = np.histogram(FT.Transaction_Time)
plt.plot(bin_edges[1:],counts)
plt.xlabel('Transaction_Time')
plt.title('Fraudulent Transactions')

plt.subplot(132)
counts, bin_edges = np.histogram(NFT.Transaction_Time)
plt.plot(bin_edges[1:],counts)
plt.xlabel('Transaction_Time')
plt.title('Non Fraudulent Transactions')

plt.subplot(133)
counts, bin_edges = np.histogram(cc.Transaction_Time)
plt.plot(bin_edges[1:],counts)
plt.xlabel('Transaction_Time')
plt.title('All Transactions')

plt.show()
```



In [14]:

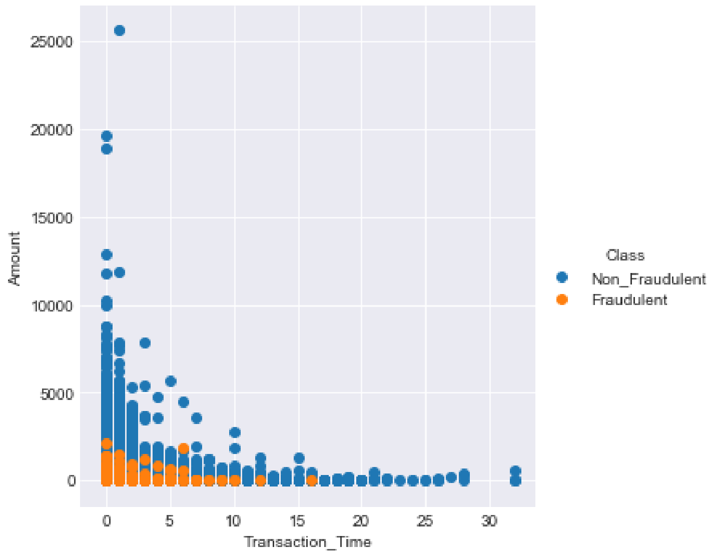
```
# Amount
plt.figure(figsize=(15,5))

plt.subplot(131)
counts, bin_edges = np.histogram(FT.Amount)
plt.plot(bin_edges[1:],counts)
plt.xlabel('Amount')
plt.title('Fraudulent Transactions')

plt.subplot(132)
counts, bin_edges = np.histogram(NFT.Amount)
plt.plot(bin_edges[1:],counts)
plt.xlabel('Amount')
plt.title('Non Fraudulent Transactions')

plt.subplot(133)
counts, bin_edges = np.histogram(cc.Amount)
plt.plot(bin_edges[1:],counts)
plt.xlabel('Amount')
plt.title('All Transactions')

plt.show()
```





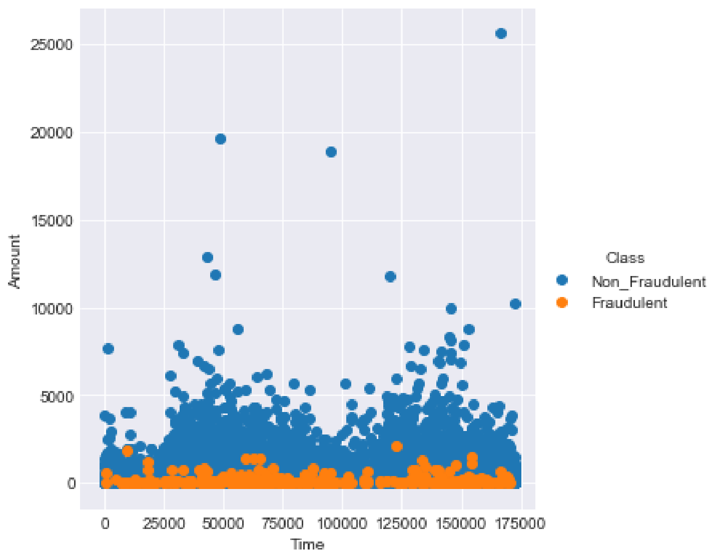
In [15]:

```
# Let's plot a scatter plot and see the relation between the Transaction Time, Time and Amount fea
tures.

p1 = sns.FacetGrid(cc,hue='Class',size = 5)
p1.map(plt.scatter, 'Transaction_Time', 'Amount')
p1.add_legend()
plt.show()
```



In [16]:

```
p2 = sns.FacetGrid(cc,hue='Class',size = 5)
p2.map(plt.scatter, 'Time', 'Amount')
p2.add_legend()
plt.show()
```



Conclusions:

1. Out of all the transactions, 99.82% of them are Non Fraudulent Transactions.
2. Although the mean of the transaction time is 0.6, the median is 0, i.e. 50% of the transactions happen instantly.
3. The median amount transferred is 22, but the mean is roughly 88. This is because of quite a few outliers.

4. When looking at the Transaction Time histograms, we might think that very few Non Fraudulent Transactions are over 5-6 seconds, whereas quite a few Fraudulent ones are.
5. This however, is not true because the y axis in both those cases vary drastically, which might be misleading. 6. In reality, there are much more non fraudulent transactions that take over 6 seconds or so to execute when compared to the fraudulent ones.
7. Majority of the fraudulent transactions transfer amounts between 0-500, while most of the non-fraudulent transactions transfer amounts between 0-5000.
8. Looking at the Time vs Amount plot, we can say that in general, fraudulent transactions transfer low amounts of money.
9. But if we look at the Transaction Time vs Amount plot, we can notice that fraudulent transactions in general, not only transfer low amounts, but have low transaction times as well.
10. So the Transaction Time vs Amount plot does a better job at classifying the transactions into the two classes.

Task 2

Finding Similarities

In [17]:

```
# Firstly, we need to read the data again since we added an additional column
# and the metric we are using here doesn't consider the additional column.

ccf =  pd.read_csv('creditcard.csv')
ccf.head()
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.

5 rows × 31 columns

In [18]:

```
# Now let us define our metric
import math

def length(x):
    '''
    This function takes in a vector and returns the length of that vector.
    '''
    sum = 0
    for i in x:
        sum += i**2
    return math.sqrt(sum)

def similarity(x,y):
    '''
    This function computes the angle between the two vectors x and y in radians.
    '''
    dot_product = np.dot(x,y)
    product_of_lengths = length(x) * length(y)
    return math.acos((dot_product/product_of_lengths))
```

In [19]:

```
# We need a sample of the main dataset, which represents the
# distribution of classes in the main dataset. Since roughly 99%
```

```
# distribution of classes in the main dataset. Since rougly 99%
# of the transactions are non fraudulent, we need 99 non fraudulent
# and 1 fraudulent transaction in our sample of 100 transactions.

nft = ccf[ccf['Class'] == 0].sample(99)
ft = ccf[ccf['Class'] == 1].sample(1)
```

In [20]:

```
sample = pd.concat([nft,ft])

sample
```

Out[20]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
244338	152310.0	2.117349	-0.147794	-1.619695	-0.019813	0.565325	-0.110829	0.038876	-0.089620	0.403485	...	-0.330166	-0.8
206410	136199.0	-0.351773	0.845695	-0.089441	0.678521	0.857399	-0.985360	1.507309	-0.369851	-1.423575	...	0.388089	1.
203439	134817.0	1.611193	-1.364765	-0.421570	-0.488959	-1.045838	-0.015210	-0.582332	0.050158	1.704151	...	-0.101573	-0.3
220018	141983.0	1.711077	-0.967045	-0.667354	-0.272593	-0.676820	-0.312633	-0.341915	-0.050622	0.743007	...	-0.155954	-0.
39053	39672.0	-0.439930	-0.295157	1.685874	-1.987982	-0.360950	0.395780	-0.438423	0.363248	-1.016557	...	0.551577	1.3
74770	55747.0	-0.524995	-0.962507	0.509121	-2.825122	-0.717511	-0.817658	0.742923	-0.273124	-2.121742	...	0.017055	0.
105038	69338.0	-4.274920	-1.297880	0.166557	3.594189	0.013444	0.342348	-1.319782	1.638366	-1.692479	...	0.085583	-0.4
25058	33501.0	-0.308106	-0.072117	2.588229	-1.173736	-1.006808	-0.019729	-0.310793	-0.066923	-0.794617	...	0.234112	0.9
66710	52181.0	1.389088	-1.730484	1.409175	-0.928413	-2.352872	0.435487	-2.004104	0.286962	-0.151339	...	-0.246615	-0.0
211826	138586.0	-2.986939	2.111087	-1.672516	-2.101711	-1.381548	-1.213100	-0.588836	1.827454	0.214092	...	0.341046	0.5
130230	79312.0	-1.149270	1.039504	1.572330	-1.497455	-0.163628	-0.443614	0.394016	0.262601	0.134880	...	-0.055892	0.0
84460	60316.0	-0.765909	1.286549	1.103637	0.187078	-0.443037	-0.879688	0.256126	0.499630	-0.491548	...	-0.164799	-0.5
102269	68111.0	-0.685235	-0.791945	3.101388	1.098874	-0.803735	0.315929	-0.968778	0.239347	-0.256780	...	-0.185997	0.2
242190	151383.0	-0.467349	0.434023	-0.430155	-0.571319	0.842228	-1.020521	0.377140	0.257199	-0.187045	...	-0.154292	-0.6
192155	129568.0	-0.038191	-0.397755	0.963928	-2.518069	-0.718317	-0.960968	-0.069869	-0.193775	-2.077591	...	-0.089278	0.
24873	33419.0	1.152678	0.078977	-0.423224	0.579669	1.176834	1.858877	-0.178407	0.483062	0.134402	...	0.050488	0.3
84155	60187.0	-0.873993	1.787541	-0.734499	0.313870	0.182358	-0.718071	-0.630044	-2.315037	-0.808993	...	-1.312671	0.0
279720	169053.0	2.312026	-1.430082	-1.208911	-1.619857	-1.075136	-0.531039	-1.106151	-0.138716	-1.059177	...	-0.201595	-0.
7548	10361.0	-3.536531	-2.371165	1.161073	-0.276129	1.280908	-1.177305	0.169124	-0.532746	2.051900	...	-0.849081	0.4
281091	169933.0	-0.753424	1.166956	0.561580	-0.151216	0.001585	-0.349857	1.759428	-0.341610	-0.348086	...	-0.217843	-0.4

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
93271	64327.0	1.334090	0.6756[illegible]	0.486493	0.8185[illegible]	1.0694[illegible]	0.4677[illegible]	0.7822[illegible]	0.024688	0.8251[illegible]	...	0.143537	0.2
264122	161276.0	1.850512	-0.879905	-2.471575	-0.356205	0.096056	-1.390529	0.795908	-0.651153	-1.610024	...	0.080822	0.4
11275	19598.0	-0.730821	0.486595	2.159508	-1.921198	0.061057	0.269466	0.155053	0.152670	2.433759	...	-0.131423	0.0
61695	49942.0	-0.456166	0.321902	1.490023	-1.034844	-0.507239	-0.635986	0.180561	-0.165235	-1.100344	...	0.280056	0.8
70806	54027.0	-1.548044	-0.652997	1.288688	2.483189	0.845845	0.640723	0.695014	0.397628	-1.616411	...	0.279627	-0.0
92422	63925.0	1.139142	-0.574897	0.176115	-0.812239	-0.809421	-0.568300	-0.259045	0.085214	1.617206	...	-0.212520	-0.5
49489	44085.0	1.356517	-0.699455	0.619041	-0.873008	-0.827660	0.255769	-1.017008	0.056607	-0.524691	...	0.378341	1.1
25519	33646.0	-1.300761	1.379387	0.759240	-2.060415	0.688569	0.220077	0.674803	0.010790	0.684206	...	-0.308256	-0.4
40176	40126.0	-0.299175	0.034401	2.651542	2.090642	-0.308757	0.641502	-2.675274	-1.123456	1.433776	...	1.533241	-0.2
8832	12069.0	-1.395817	0.007771	2.043464	-0.748526	0.428032	-0.106904	0.141829	0.150656	1.325704	...	-0.137556	-0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
22273	32142.0	-0.323397	0.287511	1.601672	-0.791938	-0.148444	0.286998	0.137619	0.265149	-1.570090	...	-0.403799	-0.8
32581	36862.0	0.526384	-1.295599	0.359875	0.283537	-0.808748	0.610357	-0.223047	0.281681	0.418605	...	0.216576	0.0
207828	136849.0	-0.675347	0.491543	-0.299686	-0.369913	1.520477	-0.806435	0.497727	-0.953603	-0.313002	...	1.159825	1.1
260242	159465.0	-2.700256	-0.725367	0.943277	0.019468	0.374078	-1.282380	-0.605848	0.368488	0.555367	...	-0.255606	-0.6
172344	121063.0	2.207690	-0.785063	-1.467966	-1.067656	-0.434705	-0.909134	-0.453021	-0.229775	-0.674515	...	0.452658	1.2
282808	171168.0	-2.019066	2.363896	-1.077266	-0.832488	-0.470869	-1.240417	0.250472	0.724461	0.742573	...	-0.352254	-0.6
74791	55756.0	1.222118	-0.054932	0.645684	0.761856	-0.760569	-0.691848	-0.194021	-0.038620	0.710121	...	-0.313973	-0.7
112607	72724.0	-0.540918	1.039774	1.994325	0.548435	0.073726	-0.425563	0.655152	-0.044140	-0.954150	...	-0.083654	-0.1
241918	151247.0	1.675060	-2.657511	-0.407602	-1.630894	-1.988255	0.603920	-1.778640	0.228123	-0.849414	...	0.280013	0.5
30211	35826.0	1.228871	-0.203502	-1.270980	-0.594310	2.029934	3.225213	-0.406828	0.762261	-0.079954	...	-0.231912	-0.9
97476	66236.0	-1.597441	0.009935	-0.547304	-0.800612	3.072643	3.443685	-1.042975	-1.954562	-0.577354	...	-1.640258	-0.4
240901	150800.0	2.019982	0.552217	-2.478975	0.573797	0.815750	-1.204808	0.280813	-0.241948	0.204631	...	0.138368	0.5
262553	160530.0	-0.563153	0.671650	0.996302	-1.059933	-0.209804	0.092689	-0.183322	0.739987	0.144684	...	-0.075637	-0.4
139635	83264.0	-2.864978	1.497778	0.833091	1.000310	-1.314949	0.681588	-0.801724	1.361502	0.528773	...	-0.218982	-0.5
207772	136830.0	-7.794335	-6.587339	-3.031769	2.634665	2.759608	-2.895588	0.511571	-1.462214	1.685848	...	-0.910071	-0.2
65041	51408.0	-2.612167	1.384607	-0.099379	-2.542641	1.255802	-1.481799	1.360444	-1.148469	2.007792	...	-0.137828	0.2
230976	146562.0	-	0.642260	-	0.636368	1.998335	-	1.037026	0.122777	-		-	-

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
83841	60051.0	1.119885	0.016731	0.474662	1.343661	-0.395155	-0.038226	-0.198140	0.252141	0.373006	...	-0.068005	-0.0
13824	24510.0	1.111137	0.058045	0.017382	1.249269	0.478713	0.871117	-0.107408	0.130760	1.436603	...	-0.250881	-0.3
116801	74432.0	1.236026	0.015663	0.608775	0.888823	-0.642314	-0.604799	-0.146688	-0.093858	0.637451	...	-0.093438	-0.0
263367	160922.0	-6.705277	5.167674	-4.259196	-1.190870	-2.690993	0.919557	-3.962091	5.423173	0.074429	...	0.219869	-0.4
83594	59930.0	-1.351824	0.258557	1.941050	-0.301880	0.969254	-1.321467	0.274363	0.092087	-0.866401	...	-0.104771	-0.6
35735	38239.0	-1.753297	-0.115577	1.333006	0.544401	-0.481036	-0.527319	1.523210	-0.411381	-0.601413	...	-0.342050	-0.3
73935	55352.0	1.535012	-0.707702	0.074725	-1.217884	-1.336797	-1.654706	-0.403627	-0.488241	-2.316939	...	-0.134744	0.0
100019	67387.0	-0.036242	-0.414161	0.855020	-1.956822	-0.928008	-0.807745	0.219687	-0.241048	-2.535038	...	-0.122401	-0.1
258596	158726.0	1.914905	-0.145660	-1.043478	1.411911	-0.057574	-0.490398	0.029659	-0.057260	0.558307	...	0.231246	0.8
31336	36324.0	-0.227573	0.370244	0.834773	-1.059895	-0.065585	-0.234575	0.408178	0.012472	-1.940238	...	-0.275827	-1.1
145910	87273.0	-0.775666	0.625065	-0.181978	-1.926704	0.717958	-0.182528	2.172828	-0.552211	-0.268436	...	0.029269	0.3
142816	84955.0	1.388851	-0.325344	-0.085581	-0.723360	-0.668147	-1.196336	-0.114580	-0.285659	-1.446453	...	0.257890	0.6
226877	144839.0	-6.423306	1.658515	-5.866440	2.052064	-0.615817	-3.372266	-5.036556	2.643106	-2.274630	...	0.641211	-0.2

100 rows × 31 columns

In [21]:

```
# For every transaction in the sample we are finding out the top 10
# transactions in the dataset which have the lowest similarity(i,j).
count = 0
for indexS,rowS in sample.iterrows():
    similarities = []
    d = {}
    for indexM,rowM in ccf.iterrows():
        if indexS == indexM:
            continue
        else:
            similarities.append(similarity(rowS.values,rowM.values))
            d[similarities[-1]] = indexM
    similarities.sort()
    print('Given Transaction ID is: ' + str(indexS) + '\n')
    print('Similar Transactions \n')
    for i in range(10):
        print('Class = ' + str(ccf.iloc[d[similarities[i]]].Class) + \
              ' Similarity = ' + str(similarities[i]) + ' Index = ' + \
              str(ccf.iloc[d[similarities[i]]].name))
    print('\n')

    # The computation required here is massive. Which is why we are limiting the
    # number of transactions in the sample set that were compared, to 5 transactions.
    # Simply removing the if condition below will run the entire sample dataset against
    # the entire main dataset.

    count += 1
    if count == 5:
        break
```

```
Given Transaction ID is: 244338
```

```
Similar Transactions

Class = 0.0 Similarity = 1.0415901015580942e-06 Index = 255485
Class = 0.0 Similarity = 1.9235200159777223e-06 Index = 261889
Class = 0.0 Similarity = 2.7121750813619954e-06 Index = 282324
Class = 0.0 Similarity = 2.9795988721175364e-06 Index = 239495
Class = 0.0 Similarity = 3.1480620716596715e-06 Index = 273205
Class = 0.0 Similarity = 3.403406949004242e-06 Index = 242092
Class = 0.0 Similarity = 3.5486983202769385e-06 Index = 242711
Class = 0.0 Similarity = 3.6035247384518964e-06 Index = 255674
Class = 0.0 Similarity = 3.62432265922467e-06 Index = 267958
Class = 0.0 Similarity = 3.7440483375002686e-06 Index = 245972


Given Transaction ID is: 206410

Similar Transactions

Class = 0.0 Similarity = 1.7319649847862104e-05 Index = 208670
Class = 0.0 Similarity = 2.5964767762797268e-05 Index = 253390
Class = 0.0 Similarity = 3.0759270599186364e-05 Index = 171091
Class = 0.0 Similarity = 3.0805187512166155e-05 Index = 212412
Class = 0.0 Similarity = 3.082431888155934e-05 Index = 262603
Class = 0.0 Similarity = 3.08619237005744e-05 Index = 263166
Class = 0.0 Similarity = 3.179536295580699e-05 Index = 246695
Class = 0.0 Similarity = 3.231735862275158e-05 Index = 196682
Class = 0.0 Similarity = 3.245972311781009e-05 Index = 274510
Class = 0.0 Similarity = 3.325193369325333e-05 Index = 175654


Given Transaction ID is: 203439

Similar Transactions

Class = 0.0 Similarity = 1.6842076579355933e-05 Index = 193917
Class = 0.0 Similarity = 1.933633379980684e-05 Index = 205533
Class = 0.0 Similarity = 2.152288460774707e-05 Index = 249567
Class = 0.0 Similarity = 2.1708230315996383e-05 Index = 190211
Class = 0.0 Similarity = 2.1802038606066137e-05 Index = 171652
Class = 0.0 Similarity = 2.2047943611267507e-05 Index = 279512
Class = 0.0 Similarity = 2.2225050700054423e-05 Index = 283612
Class = 0.0 Similarity = 2.2532344385234286e-05 Index = 201305
Class = 0.0 Similarity = 2.2562769269501186e-05 Index = 200389
Class = 0.0 Similarity = 2.4087679827303137e-05 Index = 211416


Given Transaction ID is: 220018

Similar Transactions

Class = 0.0 Similarity = 1.3221319259519743e-05 Index = 246490
Class = 0.0 Similarity = 1.4264217791075797e-05 Index = 167162
Class = 0.0 Similarity = 1.6503633067129407e-05 Index = 214484
Class = 0.0 Similarity = 1.7362928681012266e-05 Index = 266686
Class = 0.0 Similarity = 1.8052281702047996e-05 Index = 189191
Class = 0.0 Similarity = 1.8595150635670672e-05 Index = 274023
Class = 0.0 Similarity = 2.022323650874534e-05 Index = 219069
Class = 0.0 Similarity = 2.0416619250892344e-05 Index = 247786
Class = 0.0 Similarity = 2.1599663630313538e-05 Index = 246694
Class = 0.0 Similarity = 2.1649341157183316e-05 Index = 226604


Given Transaction ID is: 39053

Similar Transactions

Class = 0.0 Similarity = 5.37654896749427e-05 Index = 114370
Class = 0.0 Similarity = 5.657192632814452e-05 Index = 53913
Class = 0.0 Similarity = 6.321944915183924e-05 Index = 100092
Class = 0.0 Similarity = 6.346431668751963e-05 Index = 27456
Class = 0.0 Similarity = 6.574919513473686e-05 Index = 47830
Class = 0.0 Similarity = 6.59125153705029e-05 Index = 106362
Class = 0.0 Similarity = 6.601883919267656e-05 Index = 64851
Class = 0.0 Similarity = 6.641871965774443e-05 Index = 59241
Class = 0.0 Similarity = 6.688316278331114e-05 Index = 73140
Class = 0.0 Similarity = 6.835663551100146e-05 Index = 131370
```