1. Importing all required libraries.
2. Creating variable->data and assigning it a dataframe of the 'creditcard.csv' file by using pandas read_csv function.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from math import sqrt
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler
from scipy import stats
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error

data = pd.read_csv('creditcard.csv' , sep=',')
```

```
/home/aayush/.local/lib/python2.7/site-packages/IPython/core/interactivesh
ell.py:2714: DtypeWarning: Columns (1,2,3,4,5,6,24,25) have mixed types. S
pecify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

- Getting the information of the dataframe data.

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284909 entries, 0 to 284908
Data columns (total 31 columns):
Time      284909 non-null int64
V1        284909 non-null object
V2        284909 non-null object
V3        284909 non-null object
V4        284909 non-null object
V5        284909 non-null object
V6        284909 non-null object
V7        284909 non-null float64
V8        284909 non-null float64
V9        284909 non-null float64
V10       284909 non-null float64
V11       284909 non-null float64
V12       284909 non-null float64
V13       284909 non-null float64
V14       284909 non-null float64
V15       284909 non-null float64
V16       284909 non-null float64
V17       284909 non-null float64
V18       284909 non-null float64
V19       284909 non-null float64
V20       284909 non-null float64
V21       284909 non-null float64
V22       284909 non-null float64
V23       284909 non-null float64
V24       284909 non-null object
V25       284909 non-null object
V26       284909 non-null float64
V27       284909 non-null float64
V28       284909 non-null float64
Amount    284909 non-null float64
Class     284909 non-null int64
dtypes: float64(21), int64(2), object(8)
memory usage: 67.4+ MB
```

- Droping NaN values from the dataframe data and checking its information.

```
In [4]: data.dropna(inplace=True)
        data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 284909 entries, 0 to 284908
Data columns (total 31 columns):
Time      284909 non-null int64
V1        284909 non-null object
V2        284909 non-null object
V3        284909 non-null object
V4        284909 non-null object
V5        284909 non-null object
V6        284909 non-null object
V7        284909 non-null float64
V8        284909 non-null float64
V9        284909 non-null float64
V10       284909 non-null float64
V11       284909 non-null float64
V12       284909 non-null float64
V13       284909 non-null float64
V14       284909 non-null float64
V15       284909 non-null float64
V16       284909 non-null float64
V17       284909 non-null float64
V18       284909 non-null float64
V19       284909 non-null float64
V20       284909 non-null float64
V21       284909 non-null float64
V22       284909 non-null float64
V23       284909 non-null float64
V24       284909 non-null object
V25       284909 non-null object
V26       284909 non-null float64
V27       284909 non-null float64
V28       284909 non-null float64
Amount    284909 non-null float64
Class     284909 non-null int64
dtypes: float64(21), int64(2), object(8)
memory usage: 69.6+ MB
```

- Checking the description of the dataframe data.

```
In [5]: data.describe()
```

Out[5]:

	Time	V7	V8	V9	V10	V
count	284909.000000	284909.000000	2.849090e+05	284909.000000	284909.000000	284909.00000
mean	94826.986259	0.000171	-9.434918e-07	-0.000010	0.000002	-0.000179
std	47485.356111	1.238456	1.194284e+00	1.098634	1.088858	1.020704
min	0.000000	-43.557242	-7.321672e+01	-13.434066	-24.588262	-4.797473
25%	54215.000000	-0.554068	-2.086343e-01	-0.643099	-0.535465	-0.762624
50%	84728.000000	0.040103	2.235024e-02	-0.051416	-0.092926	-0.032868
75%	139310.000000	0.570497	3.273893e-01	0.597165	0.453998	0.739334
max	172792.000000	120.589494	2.000721e+01	15.594995	23.745136	12.018913

8 rows × 23 columns

- Removing the duplicates from the dataframe data.

In [6]:
```
data=data.drop_duplicates()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 283827 entries, 0 to 284908
Data columns (total 31 columns):
Time      283827 non-null int64
V1        283827 non-null object
V2        283827 non-null object
V3        283827 non-null object
V4        283827 non-null object
V5        283827 non-null object
V6        283827 non-null object
V7        283827 non-null float64
V8        283827 non-null float64
V9        283827 non-null float64
V10       283827 non-null float64
V11       283827 non-null float64
V12       283827 non-null float64
V13       283827 non-null float64
V14       283827 non-null float64
V15       283827 non-null float64
V16       283827 non-null float64
V17       283827 non-null float64
V18       283827 non-null float64
V19       283827 non-null float64
V20       283827 non-null float64
V21       283827 non-null float64
V22       283827 non-null float64
V23       283827 non-null float64
V24       283827 non-null object
V25       283827 non-null object
V26       283827 non-null float64
V27       283827 non-null float64
V28       283827 non-null float64
Amount    283827 non-null float64
Class     283827 non-null int64
dtypes: float64(21), int64(2), object(8)
memory usage: 69.3+ MB
```

- Describing the dataframe data.

In [7]:
```
data.describe()
```

Out[7]:

	Time	V7	V8	V9	V10	V
count	283827.000000	283827.000000	283827.000000	283827.000000	283827.000000	283827.0000
mean	94824.121493	0.001973	-0.000859	-0.001602	-0.001440	0.000029
std	47478.280460	1.229047	1.178990	1.095495	1.076422	1.018706
min	0.000000	-43.557242	-73.216718	-13.434066	-24.588262	-4.797473
25%	54219.000000	-0.552503	-0.208839	-0.644220	-0.535610	-0.761795
50%	84728.000000	0.040862	0.021889	-0.052596	-0.093239	-0.032549
75%	139287.500000	0.570559	0.325733	0.595985	0.453642	0.739325
max	172792.000000	120.589494	20.007208	15.594995	23.745136	12.018913

8 rows × 23 columns

- Changing the values from the dataframe data which are not numeric (float64) to NaN.

In [8]:
```
cols= data.columns[data.dtypes.eq(object)]
for c in cols:
    data[c] = pd.to_numeric(data[c], errors='coerce')
data
```

Out[8]:

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0986
1	0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705
5	2	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.2603
6	4	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.0812
7	7	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.8078
8	7	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.8510
9	9	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.0695
10	10	1.449044	-1.176339	0.913860	-1.375667	-1.971383	-0.629152	-1.423236	0.0484
11	10	0.384978	0.616109	-0.874300	-0.094019	2.924584	3.317027	0.470455	0.5382
12	10	0.384978	0.616109	-0.874300	NaN	2.924584	3.317027	0.470455	0.5382
13	10	1.249999	-1.221637	0.383930	-1.234899	-1.485419	-0.753230	-0.689405	-0.2274
14	11	1.069374	0.287722	0.828613	2.712520	-0.178398	0.337544	-0.096717	0.1159
15	12	-2.791855	-0.327771	1.641750	1.767473	-0.136588	0.807596	-0.422911	-1.9071
16	12	-0.752417	0.345485	2.057323	-1.468643	-1.158394	-0.077850	-0.608581	0.0036
17	12	1.103215	-0.040296	1.267332	1.289091	-0.735997	0.288069	-0.586057	0.1893
18	13	-0.436905	0.918966	0.924591	-0.727219	0.915679	-0.127867	0.707642	0.0879
19	14	-5.401258	-5.450148	1.186305	1.736239	3.049106	-1.763406	-1.559738	0.1608
20	15	1.492936	-1.029346	0.454795	-1.438026	-1.555434	-0.720961	-1.080664	-0.0531
21	16	0.694885	-1.361819	1.029221	0.834159	-1.191209	1.309109	-0.878586	0.4452
22	17	0.962496	0.328461	-0.171479	2.109204	1.129566	1.696038	0.107712	0.5215
23	18	1.166616	0.502120	-0.067300	2.261569	0.428804	0.089474	0.241147	0.1380
24	18	0.247491	0.277666	1.185471	-0.092603	-1.314394	-0.150116	-0.946365	-1.6179
25	22	-1.946525	-0.044901	-0.405570	-1.013057	2.941968	2.955053	-0.063063	0.8555
26	22	-2.074295	-0.121482	1.322021	0.410008	0.295198	-0.959537	0.543985	-0.1046
27	23	1.173285	0.353498	0.283905	1.133563	-0.172577	-0.916054	0.369025	-0.3272
28	23	1.322707	-0.174041	0.434555	0.576038	-0.836758	-0.831083	-0.264905	-0.2209
29	23	-0.414289	0.905437	1.727453	1.473471	0.007443	-0.200331	0.740228	-0.0292
...	...	...	...	...	...	...	...	...	...
284878	132792	-0.777053	1.452754	-0.577852	-0.785506	1.234916	-0.672722	1.374274	-0.2156
284879	132793	-1.200649	0.859778	-1.414095	-1.148850	0.680327	-0.177575	0.525039	0.6149
284880	132793	1.853913	-1.333570	-2.009582	-0.572370	-0.061570	0.205897	-0.275633	-0.0523
284881	132793	-1.024355	-0.274919	-0.088891	-1.773025	0.935078	1.789531	0.750018	0.2441

- Checking the information of the datarame data.

In [9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 283827 entries, 0 to 284908
Data columns (total 31 columns):
Time      283827 non-null int64
V1        283825 non-null float64
V2        283826 non-null float64
V3        283826 non-null float64
V4        283826 non-null float64
V5        283825 non-null float64
V6        283826 non-null float64
V7        283827 non-null float64
V8        283827 non-null float64
V9        283827 non-null float64
V10       283827 non-null float64
V11       283827 non-null float64
V12       283827 non-null float64
V13       283827 non-null float64
V14       283827 non-null float64
V15       283827 non-null float64
V16       283827 non-null float64
V17       283827 non-null float64
V18       283827 non-null float64
V19       283827 non-null float64
V20       283827 non-null float64
V21       283827 non-null float64
V22       283827 non-null float64
V23       283827 non-null float64
V24       283812 non-null float64
V25       283821 non-null float64
V26       283827 non-null float64
V27       283827 non-null float64
V28       283827 non-null float64
Amount    283827 non-null float64
Class     283827 non-null int64
dtypes: float64(29), int64(2)
memory usage: 69.3 MB
```

- Checking the information of the datarame data.

In [10]:
```
data.dropna(inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 283798 entries, 0 to 284907
Data columns (total 31 columns):
Time      283798 non-null int64
V1        283798 non-null float64
V2        283798 non-null float64
V3        283798 non-null float64
V4        283798 non-null float64
V5        283798 non-null float64
V6        283798 non-null float64
V7        283798 non-null float64
V8        283798 non-null float64
V9        283798 non-null float64
V10       283798 non-null float64
V11       283798 non-null float64
V12       283798 non-null float64
V13       283798 non-null float64
V14       283798 non-null float64
V15       283798 non-null float64
V16       283798 non-null float64
V17       283798 non-null float64
V18       283798 non-null float64
V19       283798 non-null float64
V20       283798 non-null float64
V21       283798 non-null float64
V22       283798 non-null float64
V23       283798 non-null float64
V24       283798 non-null float64
V25       283798 non-null float64
V26       283798 non-null float64
V27       283798 non-null float64
V28       283798 non-null float64
Amount    283798 non-null float64
Class     283798 non-null int64
dtypes: float64(29), int64(2)
memory usage: 69.3 MB
```

- Removing duplicates from dataframe data.

In [11]:
```
data=data.drop_duplicates()
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 283726 entries, 0 to 284807
Data columns (total 31 columns):
Time      283726 non-null int64
V1        283726 non-null float64
V2        283726 non-null float64
V3        283726 non-null float64
V4        283726 non-null float64
V5        283726 non-null float64
V6        283726 non-null float64
V7        283726 non-null float64
V8        283726 non-null float64
V9        283726 non-null float64
V10       283726 non-null float64
V11       283726 non-null float64
V12       283726 non-null float64
V13       283726 non-null float64
V14       283726 non-null float64
V15       283726 non-null float64
V16       283726 non-null float64
V17       283726 non-null float64
V18       283726 non-null float64
V19       283726 non-null float64
V20       283726 non-null float64
V21       283726 non-null float64
V22       283726 non-null float64
V23       283726 non-null float64
V24       283726 non-null float64
V25       283726 non-null float64
V26       283726 non-null float64
V27       283726 non-null float64
V28       283726 non-null float64
Amount    283726 non-null float64
Class     283726 non-null int64
dtypes: float64(29), int64(2)
memory usage: 69.3 MB
```

- Creating variable y to store 'Class' column from dataframe in numpy array format.

In [12]:
```
y=data['Class']
y=np.asarray(y)
```

- Describing dataframe data.

```
In [13]: data.describe()
```

```
Out[13]:
```

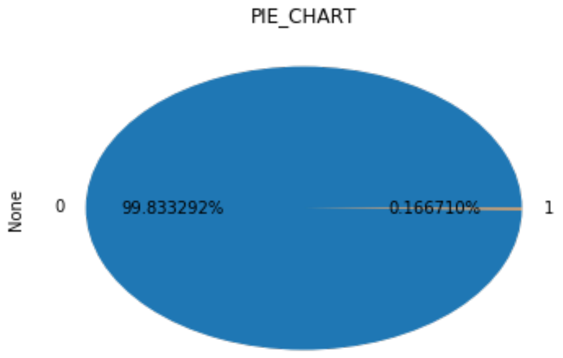
	Time	V1	V2	V3	V4	
count	283726.000000	283726.000000	283726.000000	283726.000000	283726.000000	283726.0000
mean	94811.077600	0.005917	-0.004135	0.001613	-0.002966	0.001828
std	47481.047891	1.948026	1.646703	1.508682	1.414184	1.377008
min	0.000000	-56.407510	-72.715728	-48.325589	-5.683171	-113.743307
25%	54204.750000	-0.915951	-0.600321	-0.889682	-0.850134	-0.689830
50%	84692.500000	0.020384	0.063949	0.179963	-0.022248	-0.053468
75%	139298.000000	1.316068	0.800283	1.026960	0.739647	0.612218
max	172792.000000	2.454930	22.057729	9.382558	16.875344	34.801666

8 rows × 31 columns

- Mapping the pie chart for 'Class' column.

```
In [14]: s = data.groupby("Class").size()
         s.plot(kind='pie', autopct='%1.6f%%', title="PIE_CHART")
```

```
Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5e00026390>
```



- Removing 'Class' column from dataframe data.

In [15]:
```
del data['Class']
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 283726 entries, 0 to 284807
Data columns (total 30 columns):
Time      283726 non-null int64
V1        283726 non-null float64
V2        283726 non-null float64
V3        283726 non-null float64
V4        283726 non-null float64
V5        283726 non-null float64
V6        283726 non-null float64
V7        283726 non-null float64
V8        283726 non-null float64
V9        283726 non-null float64
V10       283726 non-null float64
V11       283726 non-null float64
V12       283726 non-null float64
V13       283726 non-null float64
V14       283726 non-null float64
V15       283726 non-null float64
V16       283726 non-null float64
V17       283726 non-null float64
V18       283726 non-null float64
V19       283726 non-null float64
V20       283726 non-null float64
V21       283726 non-null float64
V22       283726 non-null float64
V23       283726 non-null float64
V24       283726 non-null float64
V25       283726 non-null float64
V26       283726 non-null float64
V27       283726 non-null float64
V28       283726 non-null float64
Amount    283726 non-null float64
dtypes: float64(29), int64(1)
memory usage: 67.1 MB
```

- Removing 'Time' column from dataframe data.

In [16]:
```
del data['Time']
```

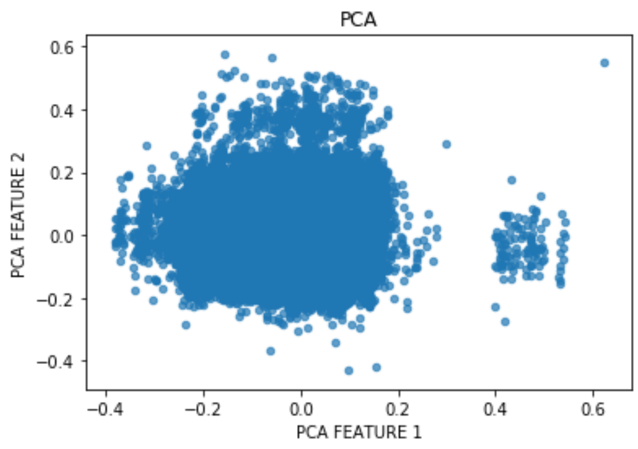
- Creating variable x to store dataframe in numpy array format.
- Creating variable x1 and assigning it same value as x.
- Creating variable y1 and assigning it same value as y.

In [17]:
```
data.describe()
x=np.asarray(data)
x1=x
y1=y
```

- Creating variable x2 and assigning it value x.
- Performing Min-Max Normalization to reduce range of every column.
- Applying PCA (making argument n_components=2) and plotting the dataframe.

```
In [18]: x2 = x
         scaler = MinMaxScaler()
         x2=scaler.fit_transform(x2)
         pca = PCA(n_components=2)
         pca.fit(x2)
         T = pca.transform(x2)
         T = pd.DataFrame(T)
         T.columns = ['PCA FEATURE 1', 'PCA FEATURE 2']
         T.plot.scatter(x='PCA FEATURE 1', y='PCA FEATURE 2', marker='o',alpha=0.7,
```

```
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5e01ad1b50>
```



- We can see from above graph that we need only 2 clusters.
- Hence we use K-mean Clustering.

- Splitting x and y in x_train, x_text and y_train, y_test (90% train, 10% test).
- Applying Min Max Normalization and then applying K mean Clustering.
- Printing accuracy_score and root mean squared error.

```
In [19]: scaler = MinMaxScaler()
         x_train,x_test,y_train,y_test = train_test_split(x,y, test_size=0.1, random_
         x_train = scaler.fit_transform(x_train)
         x_test = scaler.transform(x_test)
         kmeans = KMeans(n_clusters=2, max_iter=600, algorithm = 'auto')
         kmeans.fit(x_train)
         result = kmeans.predict(x_test)
         print(accuracy_score(result, y_test))
         print(sqrt(mean_squared_error(result, y_test)))

         0.6779684911711839
         0.567478201193
```

- Splitting x1 and y1 in x1_train, x1_text and y1_train, y1_test (90% train, 10% test).
- Applying Z-Score Normalization and then applying K mean Clustering.
- Printing accuracy_score and root mean squared error.

```
In [20]: x1 = stats.zscore(x1)
         x1_train,x1_test,y1_train,y1_test = train_test_split(x1,y1, test_size=0.1,
         kmeans = KMeans(n_clusters=2, max_iter=600, algorithm = 'auto')
         kmeans.fit(x1_train)
         result1 = kmeans.predict(x1_test)
         print(accuracy_score(result1, y1_test.round()))
         print(sqrt(mean_squared_error(result1, y1_test.round())))

         0.1494026010643922
         0.922278373885
```

- Splitting x and y in x_train, x_text and y_train,y_test (90% train, 10% test).
- Applying K mean Clustering without Noramlization.
- Printing accuracy_score and root mean squared error.

In [21]:
```
x_train,x_test,y_train,y_test = train_test_split(x,y, test_size=0.1, random_
kmeans = KMeans(n_clusters=2, max_iter=1000, algorithm = 'auto')
kmeans.fit(x_train)
result = kmeans.predict(x_test)
print (accuracy_score(result, y_test))
print (sqrt(mean_squared_error(result, y_test)))
```

```
0.9788531350227329
0.145419616893
```

- Printing the Correlation between class(y_test) and predicted_column(result).

In [22]:
```
np.corrcoef(y_test, result)
```

Out[22]:
```
array([[1.        , 0.01509506],
       [0.01509506, 1.        ]])
```

- Root mean squared error is minimum when we apply k mean on x and y(or the dataframe) without normalization.