DATA MINING PROJECT

CREDIT CARD FARUDELTY DETECTION

Catherine Ashraf Emiel | 20201378779
Mazen Mohammed El-Sayed | 20201312620
Andrew Botros | 20201322493

1. Our Data:

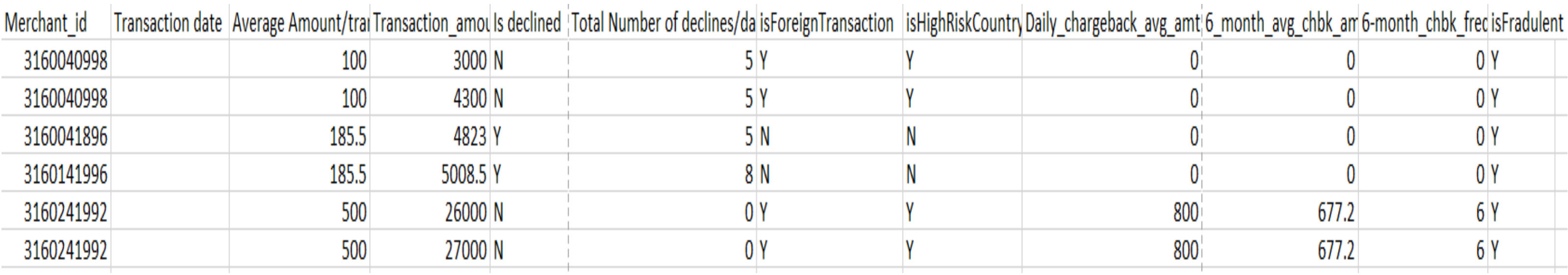
Credit card Fraud Detection:

Target:

We aim to know if the credit card is fraud or not based on some features…

Our data contains 12 rows in our model, the first 11 row are the inputs, and the last row is the desired output

data =

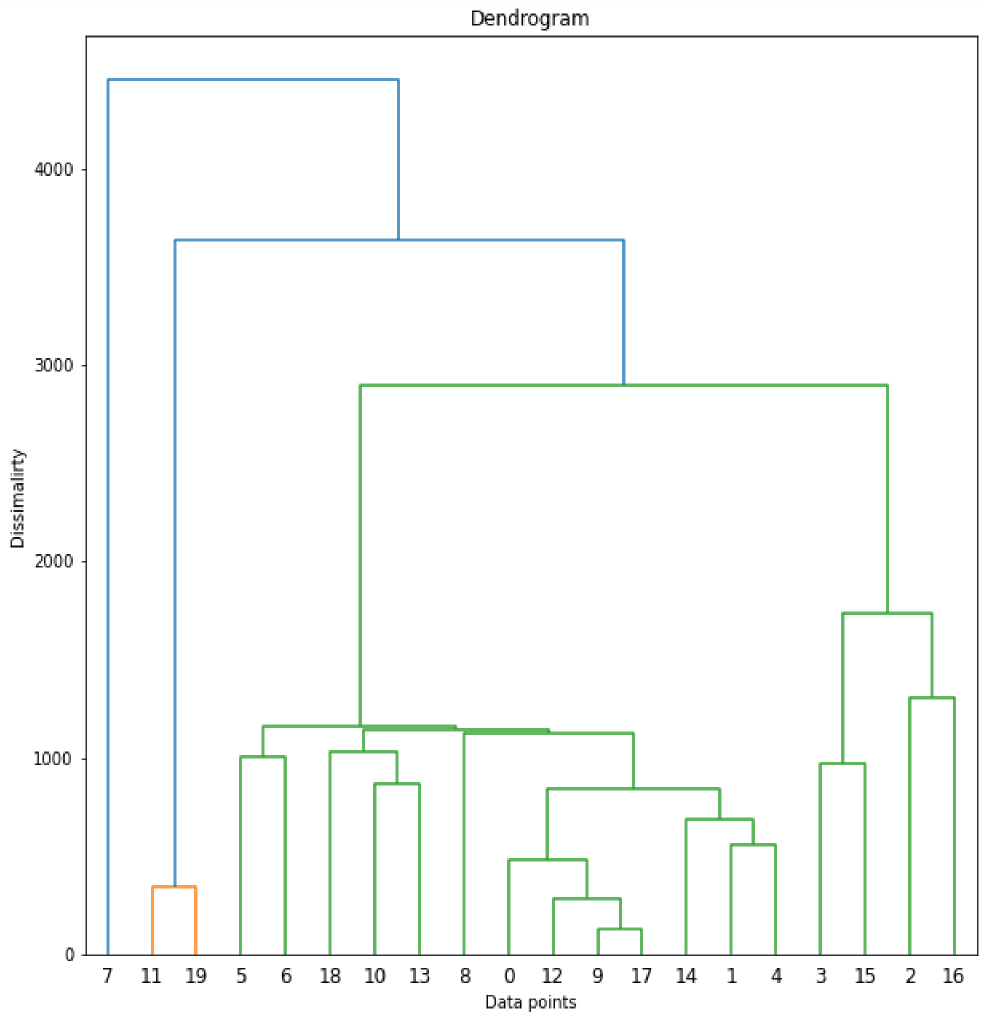


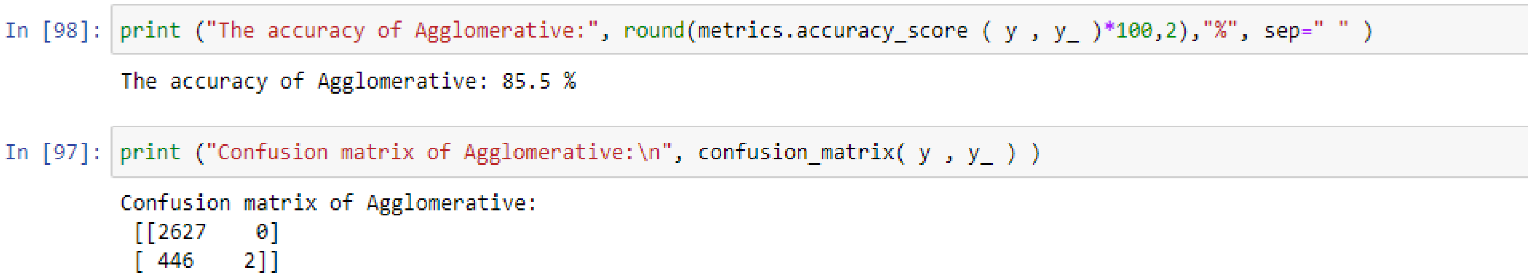
Merchant_id	Transaction date	Average Amount/trai	Transaction_amou	Is declined	Total Number of declines/da	isForeignTransaction	isHighRiskCountry	Daily_chargeback_avg_amt	6_month_avg_chbk_am	6-month_chbk_frec	isFradulent
3160040998		100	3000	N	5	Y	Y	0	0	0	Y
3160040998		100	4300	N	5	Y	Y	0	0	0	Y
3160041896		185.5	4823	Y	5	N	N	0	0	0	Y
3160141996		185.5	5008.5	Y	8	N	N	0	0	0	Y
3160241992		500	26000	N	0	Y	Y	800	677.2	6	Y
3160241992		500	27000	N	0	Y	Y	800	677.2	6	Y

Dataset description:

1. Merchant_id – The id of the merchant
2. Transaction_date – The date of the transaction, this column is mostly null, thus neglected (dropped)
3. Average_amount/Transaction/day – Average amount of the transaction in local currency per day.
4. Transaction_amount – Amount of the transaction in local currency
5. Total_Number_of_Declines /day – Number of times the card was declined before in a day
6. IsForeignTransaction – If the transaction done by a foreign ATM or country
7. IsHighRiskCountry - If the local country of the merchant is considered a high-risk country
8. Daily_chargeback_avg_amt – Daily average amount of chargeback of each merchant
9. 6_month_avg_chbk_amt - average amount of chargeback of each merchant in past 6 months
10. 6-month_chbk_freq - frequency of chargeback of each merchant in past 6 months
11. isFradulent – The final result, if this card is Fraud or not.

2. Agglomerative:



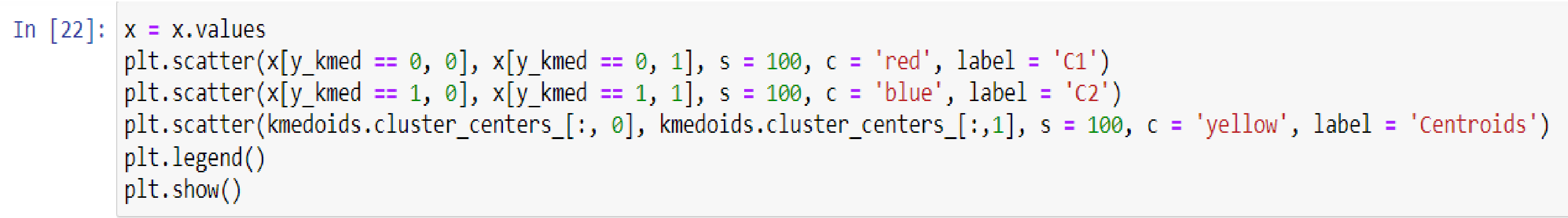


```
In [98]: print ("The accuracy of Agglomerative:", round(metrics.accuracy_score ( y , y_ )*100,2),"%", sep=" " )

         The accuracy of Agglomerative: 85.5 %

In [97]: print ("Confusion matrix of Agglomerative:\n", confusion_matrix( y , y_ ) )

         Confusion matrix of Agglomerative:
          [[2627    0]
          [ 446    2]]
```

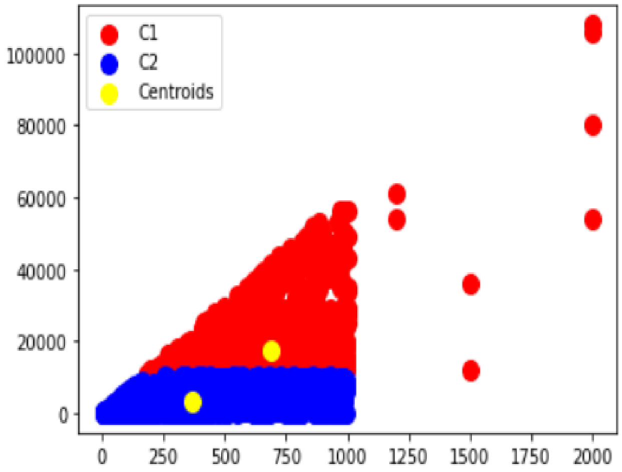
The graph here shows, how much clusters can be grouped together after certain iterations and how dissimilar they are. Knowing we wanted 2 clusters, the algorithm grouped all data points together till there was 2 clusters, one containing the data point (7) and the other contains the rest. The accuracy of this algorithm is only 85.5% which is relatively high and good. Meaning, most predicted outputs are equal to the desired output. The confusion matrix shows that sum of true predictions is way higher than the false ones.

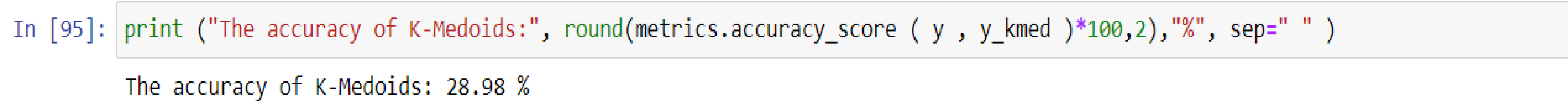
So, this algorithm is desirable for such dataset.

3. K-Medoids:

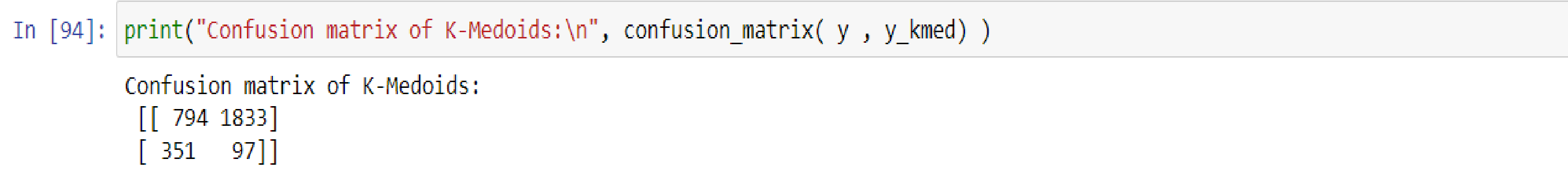


```
In [22]: x = x.values
         plt.scatter(x[y_kmed == 0, 0], x[y_kmed == 0, 1], s = 100, c = 'red', label = 'C1')
         plt.scatter(x[y_kmed == 1, 0], x[y_kmed == 1, 1], s = 100, c = 'blue', label = 'C2')
         plt.scatter(kmedoids.cluster_centers_[:, 0], kmedoids.cluster_centers_[:,1], s = 100, c = 'yellow', label = 'Centroids')
         plt.legend()
         plt.show()
```





```
In [95]: print ("The accuracy of K-Medoids:", round(metrics.accuracy_score ( y , y_kmed )*100,2),"%", sep=" " )

         The accuracy of K-Medoids: 28.98 %
```

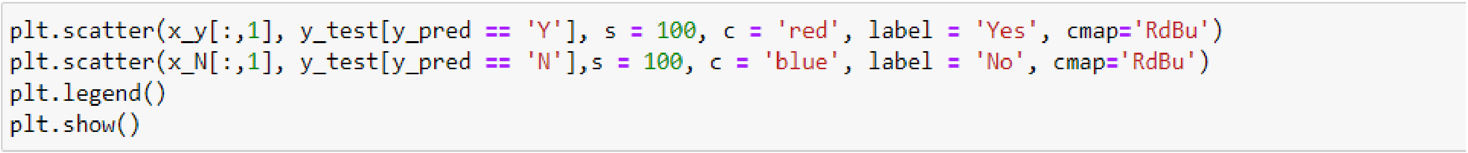


```
In [94]: print("Confusion matrix of K-Medoids:\n", confusion_matrix( y , y_kmed) )

         Confusion matrix of K-Medoids:
          [[ 794 1833]
          [ 351   97]]
```

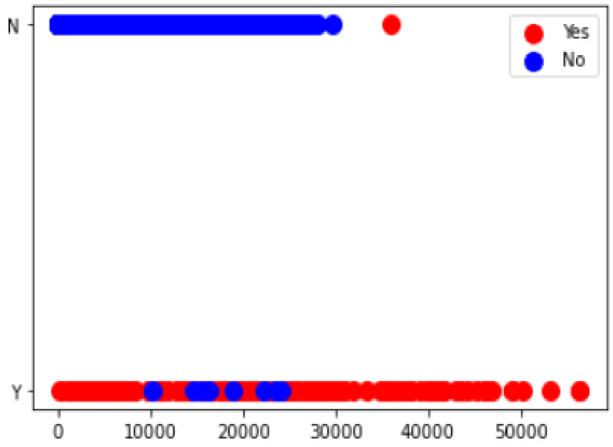
The graph here shows, the 2 centroids and the points belonging to each, we see most points belongs to cluster 1, that's 'Not Fraud'. The accuracy of this algorithm is only 28.98% which is relatively low and bad. Meaning, most predicted outputs are false. Explained by the confusion matrix, the sum of True positives and negatives is way lower than those of false positives and negatives.

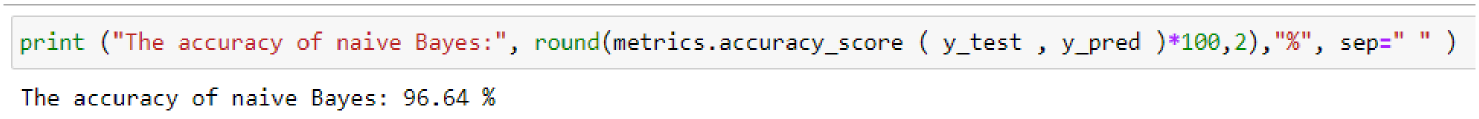
So, this algorithm isn't desirable for such dataset.

4. Naïve Bayes



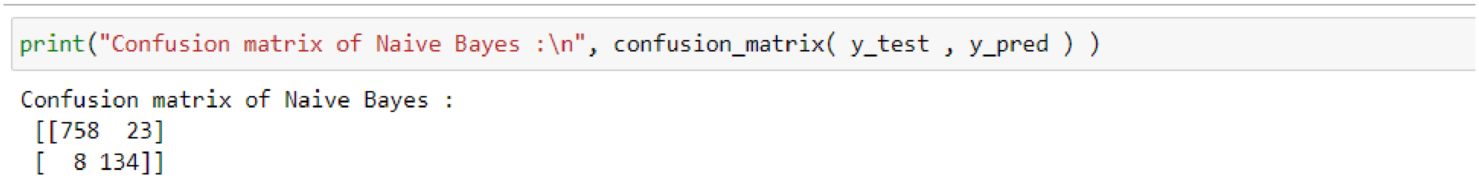
```
plt.scatter(x_y[:,1], y_test[y_pred == 'Y'], s = 100, c = 'red', label = 'Yes', cmap='RdBu')
plt.scatter(x_N[:,1], y_test[y_pred == 'N'],s = 100, c = 'blue', label = 'No', cmap='RdBu')
plt.legend()
plt.show()
```





```
print ("The accuracy of naive Bayes:", round(metrics.accuracy_score ( y_test , y_pred )*100,2),"%", sep=" " )
```

```
The accuracy of naive Bayes: 96.64 %
```



```
print("Confusion matrix of Naive Bayes :\n", confusion_matrix( y_test , y_pred ) )
```

```
Confusion matrix of Naive Bayes :
 [[758  23]
 [  8 134]]
```

The graph here shows 2 lines of plotting in y axis, The 'Y' and 'N' that indicates the actual output and the colors represent how they were classified. We can say that the graph is more confusion matrix like. The accuracy of this algorithm is 96.94% which is relatively high and great. Meaning, most predicted outputs are matched with the desired one. Explained by the confusion matrix, the sum of True positives and negatives is way greater than those of false positives and negatives.

So, this algorithm is commonly desirable for such dataset.