Agglomarative final

February 15, 2022

1 Implementation of agglomarative clustering with scikit-learn

2 importing the neccessary library

[1]:
```
#importing the neccessary library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

3 Read the dataset

[2]:
```
data =pd.read_csv('credit_card.csv')# Reading or importing the dataset
```

[3]:
```
data.head(5)# Shows the first five rows with all columns of our dataset
```

[3]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	\
0	C10001	40.900749	0.818182	95.40	0.00	
1	C10002	3202.467416	0.909091	0.00	0.00	
2	C10003	2495.148862	1.000000	773.17	773.17	
3	C10004	1666.670542	0.636364	1499.00	1499.00	
4	C10005	817.714335	1.000000	16.00	16.00	

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	\
0	95.4	0.000000	0.166667	
1	0.0	6442.945483	0.000000	
2	0.0	0.000000	1.000000	
3	0.0	205.788017	0.083333	
4	0.0	0.000000	0.083333	

	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	\
0	0.000000	0.083333	
1	0.000000	0.000000	
2	1.000000	0.000000	
3	0.083333	0.000000	
4	0.083333	0.000000	

	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	\

```
0                0.000000                    0              2        1000.0
1                0.250000                    4              0        7000.0
2                0.000000                    0             12        7500.0
3                0.083333                    1              1        7500.0
4                0.000000                    0              1        1200.0

      PAYMENTS  MINIMUM_PAYMENTS  PRC_FULL_PAYMENT  TENURE
0   201.802084        139.509787          0.000000      12
1  4103.032597       1072.340217          0.222222      12
2   622.066742        627.284787          0.000000      12
3     0.000000               NaN          0.000000      12
4   678.334763        244.791237          0.000000      12
```

[25]:
```
data.shape
```

[25]:
```
(8950, 17)
```

4 removing unwanted feature from the dataset

[4]:
```
data = data.drop('CUST_ID', axis = 1) # removing "CUST_ID" feature from the␣
↪columns
```

[5]:
```
data.head(5)
```

[5]:
```
        BALANCE  BALANCE_FREQUENCY  PURCHASES  ONEOFF_PURCHASES  \
0    40.900749           0.818182      95.40              0.00
1  3202.467416           0.909091       0.00              0.00
2  2495.148862           1.000000     773.17            773.17
3  1666.670542           0.636364    1499.00           1499.00
4   817.714335           1.000000      16.00             16.00

   INSTALLMENTS_PURCHASES  CASH_ADVANCE  PURCHASES_FREQUENCY  \
0                    95.4      0.000000             0.166667
1                     0.0   6442.945483             0.000000
2                     0.0      0.000000             1.000000
3                     0.0    205.788017             0.083333
4                     0.0      0.000000             0.083333

   ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
0                    0.000000                          0.083333
1                    0.000000                          0.000000
2                    1.000000                          0.000000
3                    0.083333                          0.000000
4                    0.083333                          0.000000

   CASH_ADVANCE_FREQUENCY  CASH_ADVANCE_TRX  PURCHASES_TRX  CREDIT_LIMIT  \
```

```
0                0.000000                    0              2        1000.0
1                0.250000                    4              0        7000.0
2                0.000000                    0             12        7500.0
3                0.083333                    1              1        7500.0
4                0.000000                    0              1        1200.0

      PAYMENTS  MINIMUM_PAYMENTS  PRC_FULL_PAYMENT  TENURE
0   201.802084        139.509787          0.000000      12
1  4103.032597       1072.340217          0.222222      12
2   622.066742        627.284787          0.000000      12
3     0.000000               NaN          0.000000      12
4   678.334763        244.791237          0.000000      12
```

5 Dealing with missing values

[6]:
```
data.isnull().values.any()# cheacking whether there is a missing value or not␣
↪in our dataset as a whole
```

[6]:
```
True
```

[7]:
```
data.isnull().sum()#To cheack in which of the feature has a missing value
```

[7]:
```
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
CASH_ADVANCE_TRX                    0
PURCHASES_TRX                       0
CREDIT_LIMIT                        1
PAYMENTS                            0
MINIMUM_PAYMENTS                  313
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

[8]:
```
data.fillna(method ='ffill',axis=0, inplace = True) #Forward-fill
```

[26]:
```
data.isnull().values.any()# cheacking whether there is a missing value or not␣
↪in our dataset as a whole
```

[26]:
```
False
```

6 Normalizing the dataset

[10]:
```
from sklearn.preprocessing import normalize #To make the scale of each variable␣
↪is the same.
data_scaled = normalize(data)
data_scaled = pd.DataFrame(data_scaled, columns=data.columns)# Changing to␣
↪dataframe type
data_scaled.head(5)
```

[10]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES
0	0.039353	0.000787	0.091790	0.000000
1	0.293876	0.000083	0.000000	0.000000
2	0.310798	0.000125	0.096307	0.096307
3	0.208403	0.000080	0.187437	0.187437
4	0.504284	0.000617	0.009867	0.009867

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY
0	0.09179	0.000000	0.000160
1	0.00000	0.591240	0.000000
2	0.00000	0.000000	0.000125
3	0.00000	0.025732	0.000010
4	0.00000	0.000000	0.000051

	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY
0	0.000000	0.00008
1	0.000000	0.00000
2	0.000125	0.00000
3	0.000010	0.00000
4	0.000051	0.00000

	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT
0	0.000000	0.000000	0.001924	0.962156
1	0.000023	0.000367	0.000000	0.642358
2	0.000000	0.000000	0.001495	0.934206
3	0.000010	0.000125	0.000125	0.937809
4	0.000000	0.000000	0.000617	0.740040

	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
0	0.194165	0.134230	0.00000	0.011546
1	0.376516	0.098404	0.00002	0.001101
2	0.077485	0.078135	0.00000	0.001495
3	0.000000	0.078436	0.00000	0.001500
4	0.418329	0.150963	0.00000	0.007400

7 Applying PCA

[27]:
```
from sklearn.decomposition import PCA # To reduce the dimension of the data
pca = PCA(n_components = 2)# To reduce the data dimension into two components
data_reduced = pca.fit_transform(data_scaled)
data_reduced = pd.DataFrame(data_reduced)# changing the data type to DataFrame
data_reduced.columns = ['P1', 'P2']# assigning tha name of the columns as␣
 ↪"P1",and "P2"
```

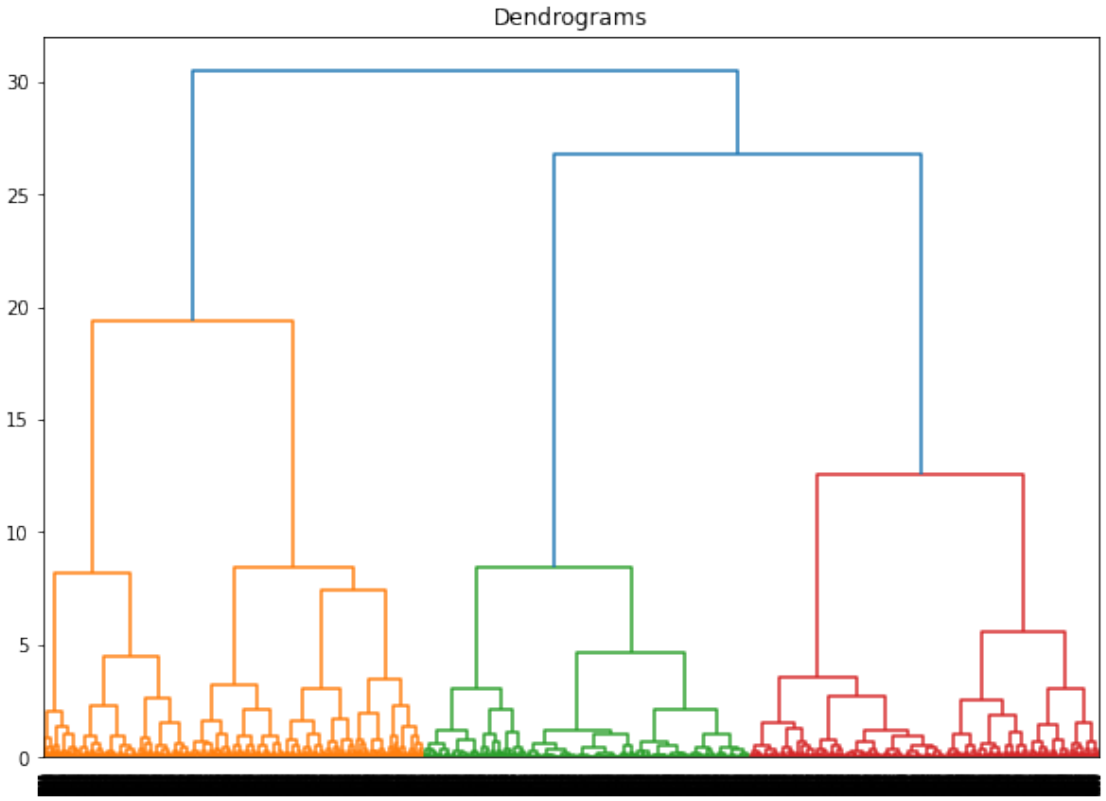
[28]:
```
data_reduced.head(5)
```

[28]:

	P1	P2
0	-0.315576	-0.044823
1	0.317328	-0.156866
2	-0.206978	-0.183004
3	-0.279409	-0.118059
4	0.140988	-0.090323

8 Ploting the dendogram

[13]:
```
import scipy.cluster.hierarchy as hc # used us to Plot the hierarchical␣
 ↪clustering as a dendrogram
plt.figure(figsize=(10, 7))# Width, height in inches.
plt.title("Dendrograms")  # the title of the diagram
dendo=hc.dendrogram(hc.linkage(data_reduced, method='ward'))
```



[ ]:

9 Applying Agglomarative clustering

[29]:
```
from sklearn.cluster import AgglomerativeClustering # importing␣
↪AgglomerativeClustering algorithm functions
Ag_cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean',␣
↪linkage='ward')# number of cluster= 3
                                                    #Distance measure␣
↪method='euclidean' and linkage method = 'ward'
y_pred= Ag_cluster.fit_predict(data_reduced) #estimates the best representative␣
↪function for the the data points
```

[30]:
```
y_pred #looking the clusters
```

[30]:
```
array([2, 0, 1, …, 2, 2, 0], dtype=int64)
```

[31]:
```
y=pd.DataFrame(y_pred) #changing the array 'y_pred' cluster into DataFrame and␣
↪assigned into 'y'
```

[32]:
```
y
```

[32]:
```
      0
0     2
1     0
2     1
3     1
4     0
…   ..
8945  2
8946  2
8947  2
8948  2
8949  0

[8950 rows x 1 columns]
```

10 counting the total number of instances in each clusters

[33]:
```
print("the total number of each of the assigned points in each cluster is: ")
print("For cluster two" , y[y==2].value_counts())
print("For cluster one" , y[y==1].value_counts())
print("For cluster zeros" , y[y==0].value_counts())
```

```
the total number of each of the assigned points in each cluster is:
For cluster two 2.0    2769
dtype: int64
For cluster one 1.0    2950
dtype: int64
For cluster zeros 0.0    3231
dtype: int64
```

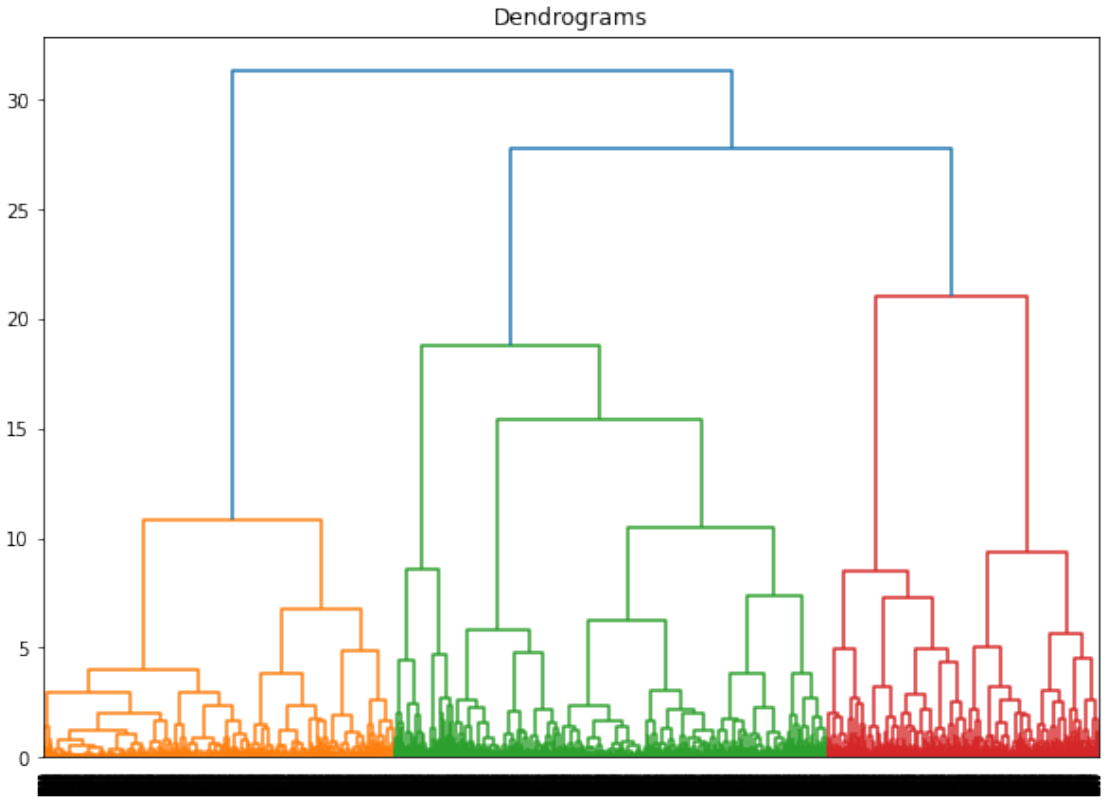
11 Scatter ploting the clusters

[34]:
```
plt.figure(figsize=(10, 7))
plt.scatter(data_reduced['P1'], data_reduced['P2'], c=Ag_cluster.labels_) # c=␣
 ↪list of colors
plt.title('Clusters of  reduced data')
```

[34]:
```
Text(0.5, 1.0, 'Clusters of  reduced data')
```

12 With out Applying PCA

[21]:
```
import scipy.cluster.hierarchy as hc # used us to Plot the hierarchical␣
 ↪clustering as a dendrogram
plt.figure(figsize=(10, 7))# Width, height in inches.
plt.title("Dendrograms")  # the title of the diagram
dend = hc.dendrogram(hc.linkage(data_scaled, method='ward'))
```



[22]:
```
from sklearn.cluster import AgglomerativeClustering # importing␣
↪AgglomerativeClustering algorithm functions
cluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean',␣
↪linkage='ward')# number of cluster= 3
#Distance measure method='euclidean' and linkage method = 'ward'
cluster.fit_predict(data_scaled)
```

[22]:
```
array([1, 0, 2, …, 1, 1, 0], dtype=int64)
```

[23]:
```
data_scaled.head()
```

[23]:

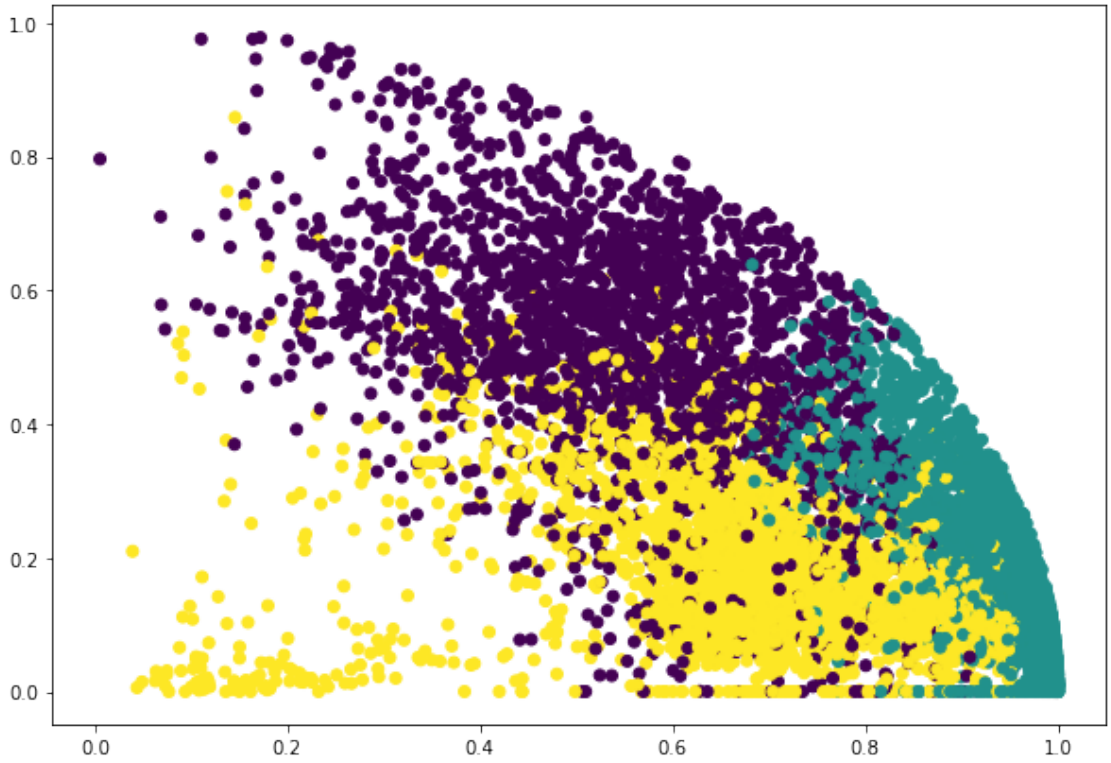
	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	\
0	0.039353	0.000787	0.091790	0.000000	
1	0.293876	0.000083	0.000000	0.000000	
2	0.310798	0.000125	0.096307	0.096307	
3	0.208403	0.000080	0.187437	0.187437	
4	0.504284	0.000617	0.009867	0.009867	

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	\
0	0.09179	0.000000	0.000160	
1	0.00000	0.591240	0.000000	

```
2                 0.00000      0.000000             0.000125
3                 0.00000      0.025732             0.000010
4                 0.00000      0.000000             0.000051

   ONEOFF_PURCHASES_FREQUENCY  PURCHASES_INSTALLMENTS_FREQUENCY  \
0                    0.000000                           0.00008
1                    0.000000                           0.00000
2                    0.000125                           0.00000
3                    0.000010                           0.00000
4                    0.000051                           0.00000

   CASH_ADVANCE_FREQUENCY  CASH_ADVANCE_TRX  PURCHASES_TRX  CREDIT_LIMIT  \
0                0.000000          0.000000       0.001924      0.962156
1                0.000023          0.000367       0.000000      0.642358
2                0.000000          0.000000       0.001495      0.934206
3                0.000010          0.000125       0.000125      0.937809
4                0.000000          0.000000       0.000617      0.740040

   PAYMENTS  MINIMUM_PAYMENTS  PRC_FULL_PAYMENT    TENURE
0  0.194165          0.134230           0.00000  0.011546
1  0.376516          0.098404           0.00002  0.001101
2  0.077485          0.078135           0.00000  0.001495
3  0.000000          0.078436           0.00000  0.001500
4  0.418329          0.150963           0.00000  0.007400
```

[24]:
```
plt.figure(figsize=(10, 7))
plt.scatter(data_scaled['CREDIT_LIMIT'], data_scaled['PAYMENTS'], c=cluster.
 ↪labels_)
```

[24]:
```
<matplotlib.collections.PathCollection at 0x2130e411c40>
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

[ ]: