CREDIT CARD CUSTOMER SEGMENTATION USING CLUSTERING AND DIMENSIONALITY REDUCTION

Unsupervised Machine Learning: Clustering and Dimensionality Reduction

JUNE 21, 2022

RATEEB YEHYA

Table of Contents

1. About the Data Set

The dataset summarizes the usage behavior of about 9000 active credit card holders within a period of six months. The file consists of 18 behavioral variables, discussed below:

a. **CUST_ID** : Identification of Credit Card holder (Categorical)
b. **BALANCE** : Balance amount left in their account to make purchases
c. **BALANCE_FREQUENCY** : How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated)
d. **PURCHASES** : Amount of purchases made from account
e. **ONEOFF_PURCHASES** : Maximum purchase amount done in one-go
f. **INSTALLMENTS_PURCHASES** : Amount of purchase done in installment
g. **CASH_ADVANCE** : Cash in advance given by the user
h. **PURCHASES_FREQUENCY** : How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)
i. **ONEOFF_PURCHASES_FREQUENCY** : How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)
j. **PURCHASES_INSTALLMENTS_FREQUENCY** : How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)
k. **CASH_ADVANCE_FREQUENCY** : How frequently the cash in advance being paid
l. **CASH_ADVANCE_TRX** : Number of Transactions made with "Cash in Advanced"
m. **PURCHASES_TRX** : Number of purchase transactions made
n. **CREDIT_LIMIT** : Limit of Credit Card for user
o. **PAYMENTS** : Amount of Payment done by user
p. **MINIMUM_PAYMENTS** : Minimum amount of payments made by user
q. **PRC_FULL_PAYMENT** : Percent of full payment paid by user
r. **TENURE** : Tenure of credit card service for user

The dataset is extracted from the following link on *Kaggle*: https://www.kaggle.com/datasets/arjunbhasin2013/ccdata

2. Objective

The objective of this report is to develop a customer segmentation of active credit card users for a certain bank in order to define an efficient marketing strategy. This project is an example of unsupervised machine learning since we do not have a target variable. Therefore, we aim to segment our dataset into different clusters from which we can develop effective marketing campaigns based on the various features that reflect customer behavior.

3. Exploratory Data Analysis

First, we import all the necessary libraries.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, AgglomerativeClustering
```

We now import the data into python.

```
data_original = pd.read_csv('D:/Uni/Masters/Python/IBMMachineLearning/Course 4/FinalProj/CC GENERAL.csv')
✓ 0.2s
```

We can see from the code below that we have 8950 rows with 18 columns.

```
data_original.shape
[271] ✓ 0.9s
... (8950, 18)
```

We then check for null values.

```
data_original.isnull().sum()
✓ 0.1s
CUST_ID                             0
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
```

We can see that there exist 313 null values in $MINIMUM\_PAYMENTS$ and 1 null value in $CREDIT\_LIMIT$. Since the number of observations is high, we choose to drop all rows with null values.

```
data_original = data_original.dropna(axis=0)
data_original.shape
```
✓ 0.8s

```
(8636, 18)
```

We can see that the number of observations decreased to 8636 after dropping the rows with nulls. Since $CUST\_ID$ is not used in analysis, we drop it.

```
data_original = data_original.drop(['CUST_ID'], axis = 1)
data_original.shape
```
✓ 0.6s

```
(8636, 17)
```

We can see that the number of columns is now 17.

We save our original dataset and create a copy of it which we will use for analysis.

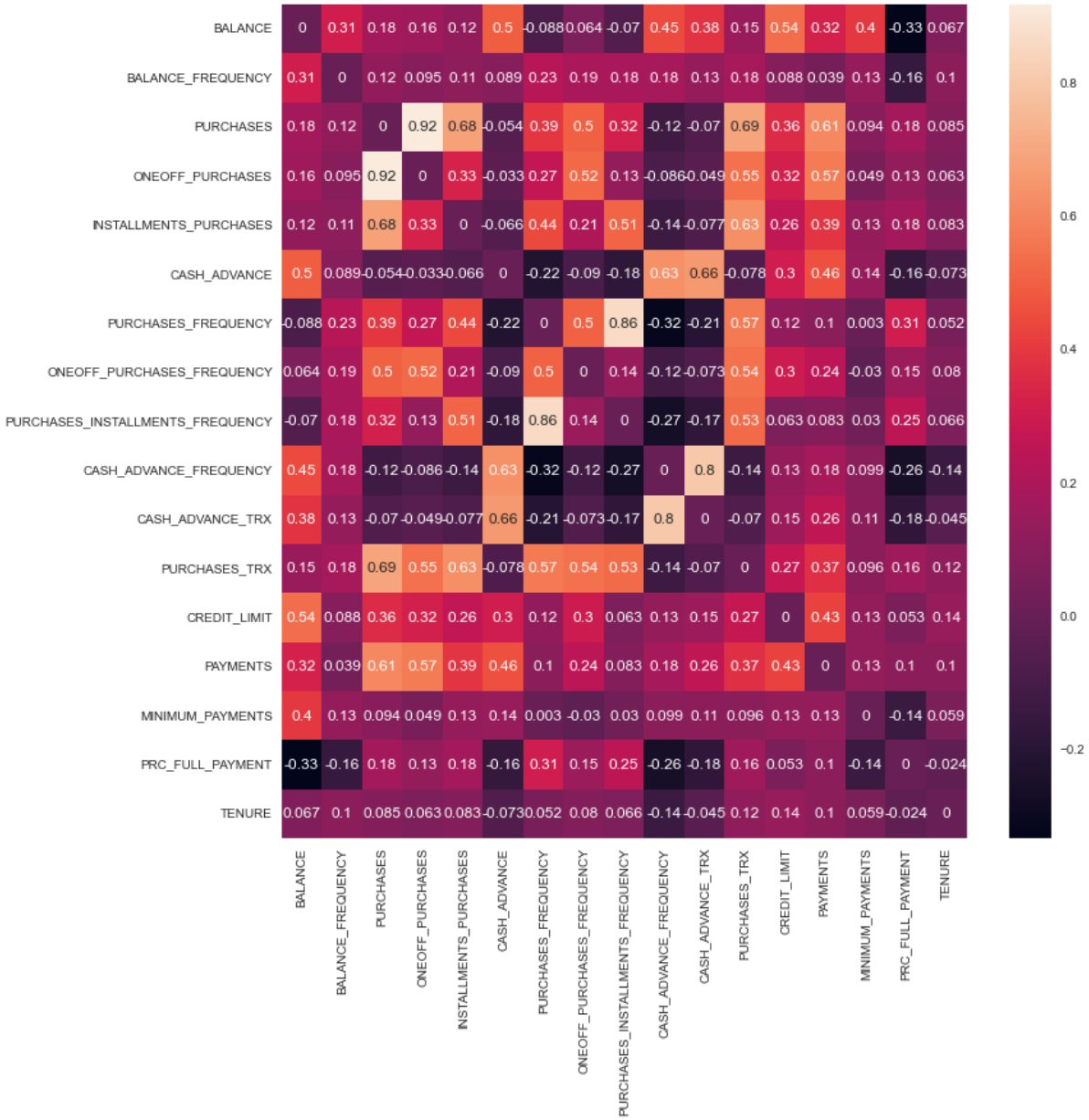
```
data = data_original.copy()
data.head().T
```
✓ 0.7s

	0	1	2	4	5
BALANCE	40.900749	3202.467416	2495.148862	817.714335	1809.828751
BALANCE_FREQUENCY	0.818182	0.909091	1.000000	1.000000	1.000000
PURCHASES	95.400000	0.000000	773.170000	16.000000	1333.280000
ONEOFF_PURCHASES	0.000000	0.000000	773.170000	16.000000	0.000000
INSTALLMENTS_PURCHASES	95.400000	0.000000	0.000000	0.000000	1333.280000
CASH_ADVANCE	0.000000	6442.945483	0.000000	0.000000	0.000000
PURCHASES_FREQUENCY	0.166667	0.000000	1.000000	0.083333	0.666667
ONEOFF_PURCHASES_FREQUENCY	0.000000	0.000000	1.000000	0.083333	0.000000
PURCHASES_INSTALLMENTS_FREQUENCY	0.083333	0.000000	0.000000	0.000000	0.583333
CASH_ADVANCE_FREQUENCY	0.000000	0.250000	0.000000	0.000000	0.000000
CASH_ADVANCE_TRX	0.000000	4.000000	0.000000	0.000000	0.000000
PURCHASES_TRX	2.000000	0.000000	12.000000	1.000000	8.000000
CREDIT_LIMIT	1000.000000	7000.000000	7500.000000	1200.000000	1800.000000
PAYMENTS	201.802084	4103.032597	622.066742	678.334763	1400.057770
MINIMUM_PAYMENTS	139.509787	1072.340217	627.284787	244.791237	2407.246035
PRC_FULL_PAYMENT	0.000000	0.222222	0.000000	0.000000	0.000000
TENURE	12.000000	12.000000	12.000000	12.000000	12.000000

We now plot the correlation matrix after changing the diagonal values to 0. The reason why we change them to 0 is because we later need to find the maximum correlation between features and we do not want it to output the number 1 on the diagonal.

```
corr_mat = data.corr()
for x in range(len(corr_mat)):
    corr_mat.iloc[x,x] = 0
```
✓ 0.1s

```
corr_mat
plt.figure(figsize=(12,12))
sns.heatmap(corr_mat, annot = True)
```
✓ 1.7s



We can then check between maximum correlation between features.

```
corr_mat.abs().max().sort_values(ascending=False)
```
✓ 0.1s

```
PURCHASES                          0.916780
ONEOFF_PURCHASES                   0.916780
PURCHASES_INSTALLMENTS_FREQUENCY   0.862338
PURCHASES_FREQUENCY                0.862338
CASH_ADVANCE_FREQUENCY             0.799593
CASH_ADVANCE_TRX                   0.799593
PURCHASES_TRX                      0.688732
INSTALLMENTS_PURCHASES             0.679259
CASH_ADVANCE                       0.656911
PAYMENTS                           0.606782
ONEOFF_PURCHASES_FREQUENCY         0.544364
CREDIT_LIMIT                       0.535518
BALANCE                            0.535518
MINIMUM_PAYMENTS                   0.398669
PRC_FULL_PAYMENT                   0.333594
BALANCE_FREQUENCY                  0.310140
TENURE                             0.140038
dtype: float64
```

We notice that there exists high correlation among features (most of the values above are greater than 0.5). This suggests the use of dimensionality reduction before performing any clustering on the dataset.

Now let us check for the skew of the dataset.

```
#Skew
skew_cols = data.skew().sort_values(ascending = False)
skew_cols = skew_cols.loc[skew_cols > 0.75]
print(skew_cols)
```
✓ 0.2s

```
MINIMUM_PAYMENTS              13.622193
ONEOFF_PURCHASES               9.935776
PURCHASES                      8.055789
INSTALLMENTS_PURCHASES         7.216133
PAYMENTS                       5.873049
CASH_ADVANCE_TRX               5.673327
CASH_ADVANCE                   5.139629
PURCHASES_TRX                  4.578418
BALANCE                        2.374254
PRC_FULL_PAYMENT               1.886027
CASH_ADVANCE_FREQUENCY         1.795915
CREDIT_LIMIT                   1.507019
ONEOFF_PURCHASES_FREQUENCY     1.504234
dtype: float64
```

We can see here all the features that have a skew greater than 0.75. To obtain better results, we will transform those features using a log transformation for normality by using the code below.

```
for col in skew_cols.index.tolist():
    data[col] = np.log1p(data[col])
```
✓ 0.1s

Now that we have our skewed data normalized, we will standardize all columns using a standard scaler. This is of utmost importance as clustering and dimensionality reduction work on distances. Therefore, failing to scale our data will result in inaccurate clustering.

```
s = StandardScaler()
data = s.fit_transform(data)
```
✓ 0.1s

```
data = pd.DataFrame(data, columns=corr_mat.columns)
data
```
✓ 0.2s

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF
0	-1.334707	-0.370047	-0.123312	-0.996024	0.381976	-0.937894	-0.820769	
1	0.952729	0.067679	-1.686451	-0.996024	-1.096108	1.518315	-1.236139	
2	0.821135	0.505405	0.589495	1.049151	-1.096108	-0.937894	1.256077	
3	0.233137	0.505405	-0.717052	-0.124918	-1.096108	-0.937894	-1.028455	
4	0.651839	0.505405	0.775749	-0.996024	1.232117	-0.937894	0.425339	
...	...	...	...	...	...	...	...	
8631	-2.288300	-1.902089	-0.630393	-0.047046	-1.096108	-0.937894	-0.820769	
8632	-1.519916	0.505405	0.256021	-0.996024	0.740669	-0.937894	1.256077	
8633	-1.619945	-0.297095	0.017308	-0.996024	0.514944	-0.937894	0.840707	
8634	-1.895971	-0.297095	-1.686451	-0.996024	-1.096108	0.077508	-1.236139	
8635	-0.180530	-1.099590	0.707892	1.155543	-1.096108	0.420960	0.425339	

We can see how the data is now on the same scale.

4. Dimensionality Reduction: Principal Component Analysis (PCA)

It appears to be a high correlation among features from the figure regarding $corr\_mat.abs().\max()$. Therefore, we suggest performing dimensionality reduction to reduce the number of features (or columns). In that way, we avoid the curse of dimensionality when we perform clustering later on.

At first, we will check the explained variance for reducing the number of columns to all possible cases using PCA. Then, we will choose a number of components to reduce to with an acceptable explained variance. In that way, we decrease the number of dimensions without affecting our overall accuracy.

```
pca_list = list()
feature_weight_list = list()

for n in range(1,18):
    PCAmod = PCA(n_components = n)
    PCAmod.fit(data)

    pca_list.append(pd.Series({'n': n, 'model': PCAmod, 'var': PCAmod.explained_variance_ratio_.sum()}))
    weights = PCAmod.explained_variance_ratio_.reshape(-1,1)/PCAmod.explained_variance_ratio_.sum()
    overall_contributions = np.abs(PCAmod.components_)*weights
    abs_feature_values = overall_contributions.sum(axis =0)
    feature_weight_list.append(pd.DataFrame({'n':n, 'features': data.columns, 'values': abs_feature_values/(abs_feature_values.sum())}))
```

✓ 0.8s

```
pca_df = pd.concat(pca_list, axis = 1).T.set_index('n')
print(pca_df)
```

n	model	var
1	PCA(n_components=1)	0.344562
2	PCA(n_components=2)	0.569428
3	PCA(n_components=3)	0.665851
4	PCA(n_components=4)	0.743034
5	PCA(n_components=5)	0.81132
6	PCA(n_components=6)	0.854222
7	PCA(n_components=7)	0.896171
8	PCA(n_components=8)	0.926657
9	PCA(n_components=9)	0.947552
10	PCA(n_components=10)	0.963044
11	PCA(n_components=11)	0.973192
12	PCA(n_components=12)	0.981365
13	PCA(n_components=13)	0.988121
14	PCA(n_components=14)	0.993252
15	PCA(n_components=15)	0.996003
16	PCA(n_components=16)	0.998378
17	PCA(n_components=17)	1.0

We can see here that the explained variance starts to plateau starting with dimensions $= 11$. Therefore, we will reduce the number of columns from 18 to 11. In that way, when we perform clustering, we attenuate the curse of dimensionality.

As a result, we will now fit our PCA model with $n_{components} = 11$ and print down the transformed values.

```
PCAmoduse = PCA(n_components=11)
data_new = PCAmoduse.fit_transform(data)
data_new = pd.DataFrame(data_new, columns = ['col' + str(x) for x in range(1,12)])
data_new
```

[192] ✓ 0.2s

	col1	col2	col3	col4	col5	col6	col7	col8	col9	col10	col11
0	-0.264610	2.974655	-0.380826	-1.112927	0.092551	0.636947	-0.414451	-0.089470	-0.063974	-0.417113	0.246660
1	-3.469330	-0.997126	0.219202	0.447120	-1.719069	-0.979248	0.252447	-0.352971	-0.189181	0.404088	-0.347133
2	1.453137	-1.244969	2.320869	-1.583904	0.797392	-0.454235	0.199237	0.902914	0.209600	1.201579	0.553510
3	-1.142145	1.465458	0.695047	-1.925179	0.135694	-0.534331	-0.519716	-0.519485	-0.603960	-0.288553	0.267792
4	1.051982	-0.357756	-1.756010	-1.749370	-0.568199	-0.064581	0.552457	-1.270317	0.365577	-0.215965	0.456537
...	...	...	...	...	...	...	...	...	...	...	...
8631	-0.864778	5.072465	0.773861	0.835767	4.062753	-0.057519	1.251541	-0.668039	-0.115818	0.338735	0.302352
8632	1.782385	3.064747	-2.169873	1.723910	2.825652	-2.159671	0.976739	-0.353917	-0.819183	-0.236473	0.189359
8633	1.051888	3.628606	-1.900932	1.146574	3.274783	-1.225701	1.492995	-0.044131	-0.049579	0.093318	0.175810
8634	-2.680774	4.393601	-0.652961	1.458861	3.674892	-1.454746	0.277197	0.032546	-0.060706	0.081341	0.052500
8635	-0.145522	1.031985	1.181525	1.698800	5.357434	0.265261	0.824243	0.277917	0.564001	0.434584	0.166083

8636 rows × 11 columns

Notice that our dataset now contains 11 columns instead of 18. Now we are ready to perform clustering.

5. Clustering with $KMeans$ with $n = 3$

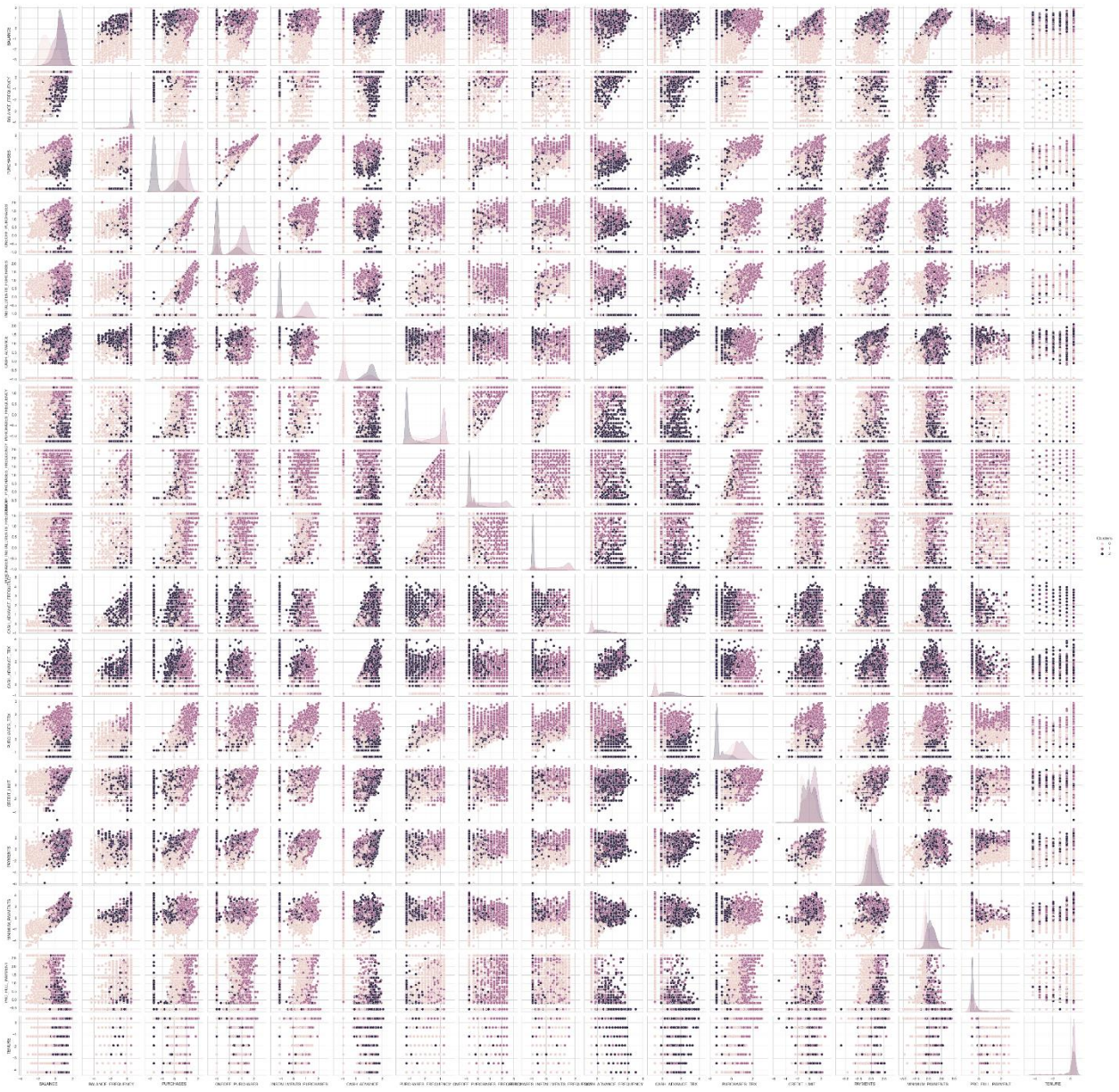
In this section, we use $KMeans$ with $n = 3$ as a first clustering method to segment active credit card customers into three segments according to the features produced by the PCA model above.

```
km = KMeans(n_clusters=3)
y_pred_km = km.fit_predict(data_new)
data_new['Clusters']=y_pred_km
#Add it also to the original data frame
data['Clusters']=y_pred_km
```

✓ 0.2s

Notice that we add an extra column called clusters which shows the cluster class, in this case 0,1 *or* 2, for each observation.

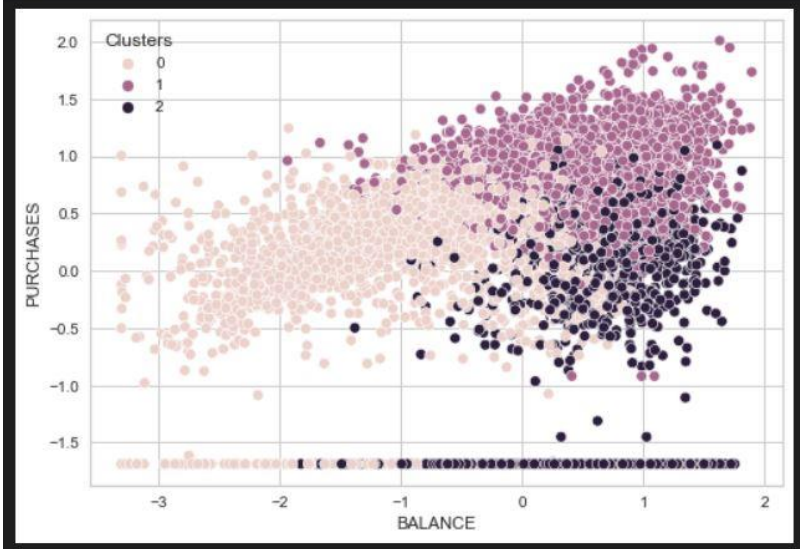
Now that we have the cluster classes of all our observations, we can produce a pair plot, hued by cluster.



The pair plot above shows the scatter plots of all our features, segmented by the clusters we produced by K-Means. Notice that we have three colors: Red, Purple, and Yellow, each representing a different cluster.

To make this clearer, let us plot the variation of Purchases versus Balance, segmented by the K-Means model.

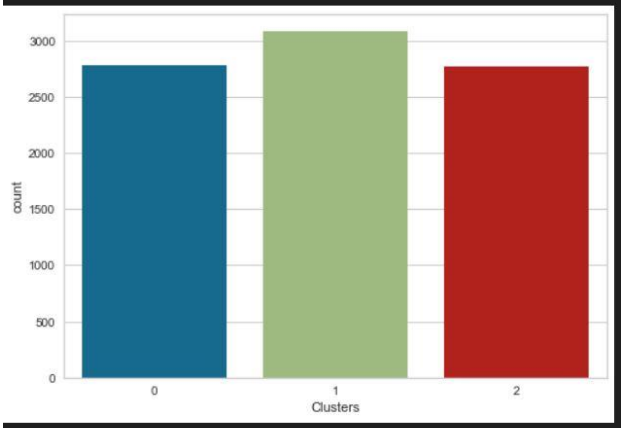
```
ax = sns.scatterplot(data = data, x= data['BALANCE'], y=data['PURCHASES'], hue = data['Clusters'])
✓ 0.9s
```



Note the distribution of the clusters in the graph above. Group 0 represents the class of customers with law balance and law-medium purchases. Group 1 represents the class of customers with high balance and low purchases. As for group 2, it represents the class of customers with high balance and high purchases.

We can also count the number of customers in each cluster.

```
sns.countplot(x=data['Clusters'])
✓ 0.4s
<AxesSubplot:xlabel='Clusters', ylabel='count'>
```



We can see that the number of customers in each cluster is approximately the same.

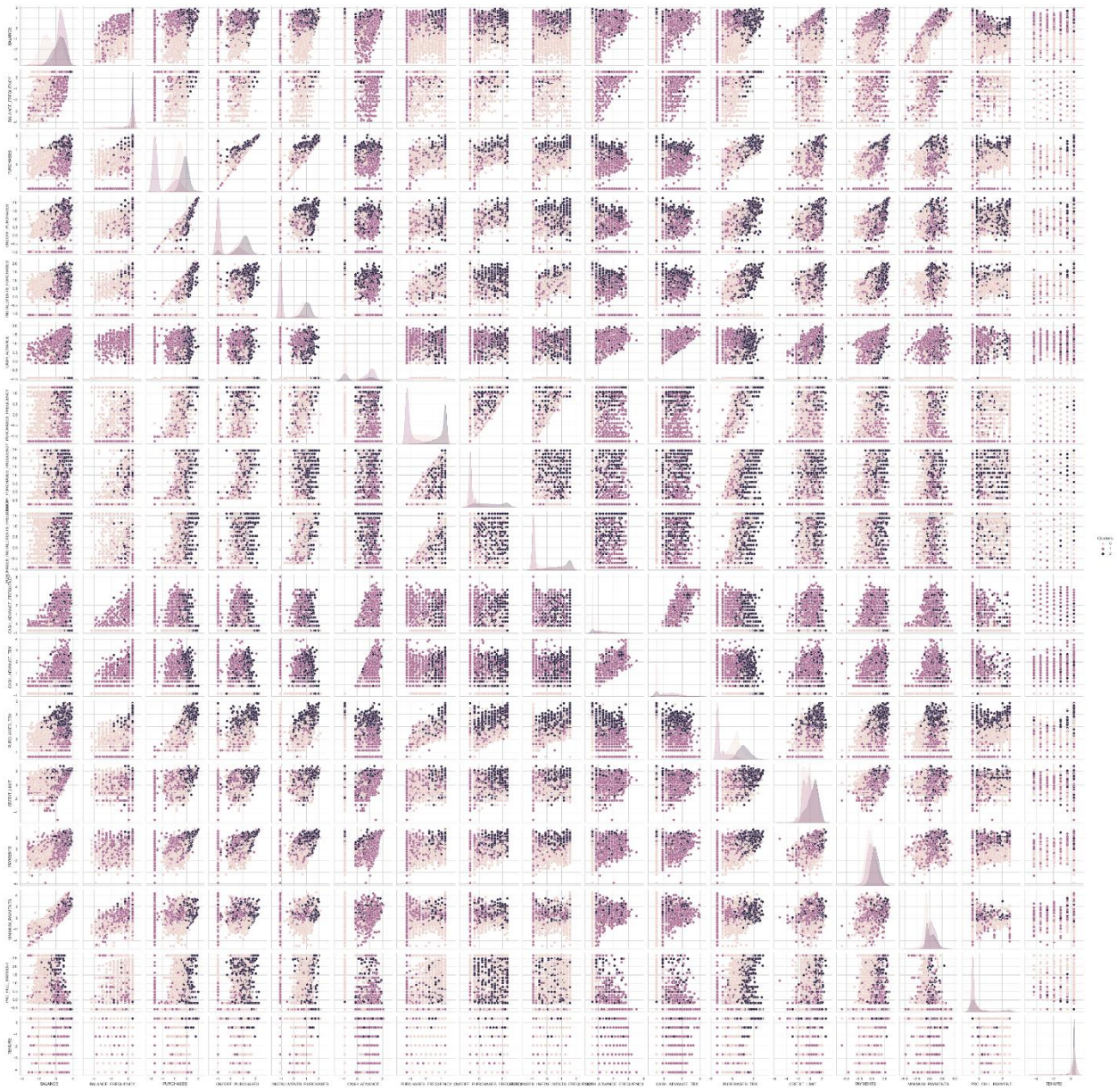
6. Clustering with Agglomerative Clustering with $n = 3$

Let us try to segment the data into three clusters, but using an Agglomerative clustering model instead of K-Means. The difference between the two models is that K-Means groups the data according to the distance between the data points and the cluster centroids. Agglomerative Clustering, however, groups the data according to the distance between the data points and other data points/ clusters.

```
AC = AgglomerativeClustering(n_clusters=3)
y_pred_ac = AC.fit_predict(data_new)
data_new['Clusters']=y_pred_ac
#Add it also to the original data frame
data['Clusters']=y_pred_ac
```

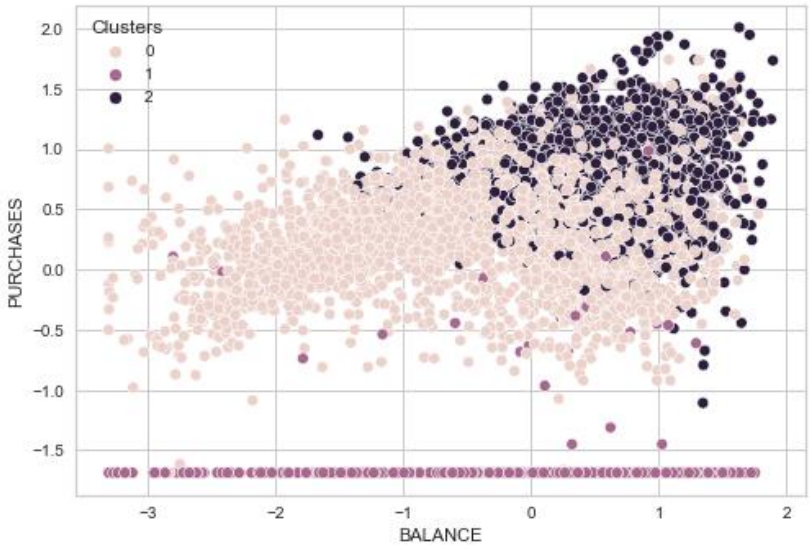
✓ 3.4s

After fitting the agglomerative clustering model with $n = 3$, we can produce a pair plot of the data, hued by clusters, as we did before with K-Means. The results are shown below.



We can see how the clustering differs between the two models, the reason being that each one of the models has its own algorithm for clustering.

To make it clearer, we will produce the exact same scatter plot of Purchases versus Balance, segmented by clusters according to the results of the Agglomerative Clustering Model. Results are shown below.



Notice how Agglomerative Clustering segments cluster 1 separately from the others.

7. Summary and Key Insights

In this report, we used two clustering algorithms, namely K-Means and Agglomerative Clustering, to segment active credit card customers of a certain bank into three different clusters. Principal Component Analysis (PCA) was used to bring down the dimensionality of the dataset so that we obtain more efficient and predictive models.

Considering Balance and Purchases, K-Means segments the customers into the following three clusters. Cluster 0 represents the class of customers with law balance and law-medium purchases. Cluster 2 represents the class of customers with high balance and low purchases. Finally, cluster 1 represents the class of customers with high balance and high purchases. As a result, different marketing campaigns can be set to target each of these clusters differently. For example, the bank might opt to ensure that cluster 1 is satisfied with the bank services, as both the balance and the purchases for this class are high. For cluster 2, however, the bank might opt to increase marketing campaigns so that customers from this class purchase more products. The reason is that customers from cluster 2 already have a high balance (i.e. they have the cash), but they are not purchasing many products. As for cluster 0, the bank might opt to encourage the low-medium purchase rate of this class by giving them loans to purchase, as this class seems to have relatively high purchases but low balance.

As for Agglomerative clustering, the data is segmented differently. Therefore, management might choose a different approach in dealing with each cluster.