In

[1]:



In

[2]:



Out[2]:

**CUST\_ID**

**BALANCE**

**BALANCE\_FREQUENCY**

**PURCHASES**

**ONEOFF\_PURCHASES**

**INSTALLMENTS\_PURCHASES**

**CASH\_ADVANCE**

**PURCHASES\_FREQ**

**import**

numpy

**as**

np

**import**

pandas

**as**

pd

**import**

matplotlib

.

pyplot

**as**

plt

**import**

seaborn

**as**

sns

**from**

scipy

**import**

stats

**from**

sklearn

.

preprocessing

**import**

StandardScaler

,

normalize

**from**

sklearn

.

cluster

**import**

KMeans

**from**

sklearn

.

cluster

**import**

MiniBatchKMeans

**from**

sklearn

.

cluster

**import**

AgglomerativeClustering

**import**

scipy

.

cluster

.

hierarchy

**as**

shc

**from**

sklearn

.

cluster

**import**

DBSCAN

**from**

sklearn

.

mixture

**import**

GaussianMixture

**from**

sklearn

.

cluster

**import**

MeanShift

**from**

sklearn

.

cluster

**import**

estimate\_bandwidth

**from**

sklearn

**import**

metrics

**from**

sklearn

.

decomposition

**import**

PCA

**import**

warnings

warnings

.

filterwarnings

(

'ignore'

)

df

**=**

pd

.

read\_csv

(

"CC GENERAL.csv"

)

df

.

head

()

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | C10001 | 40.900749 | 0.818182 | 95.40 | 0.00 | 95.4 | 0.000000 | 0 |
| **1** | C10002 | 3202.467416 | 0.909091 | 0.00 | 0.00 | 0.0 | 6442.945483 | 0 |
| **2** | C10003 | 2495.148862 | 1.000000 | 773.17 | 773.17 | 0.0 | 0.000000 | 1 |
| **3** | C10004 | 1666.670542 | 0.636364 | 1499.00 | 1499.00 | 0.0 | 205.788017 | 0 |

In [3]: 

**4**

C10005

817.714335

1.000000

16.00

16.00

0.0

0.000000

0

There are 18 columns and 8950 rows in this data collection.

print

(

'This data set has {} rows and {} columns.\n'

.

format

(

df

.

shape

[

0

,

]

df

.

shape

[

1

]))

df

.

info

()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 8950 entries, 0 to 8949 Data columns (total 18 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

1. CUST\_ID 8950 non-null object
2. BALANCE 8950 non-null float64
3. BALANCE\_FREQUENCY 8950 non-null float64
4. PURCHASES 8950 non-null float64
5. ONEOFF\_PURCHASES 8950 non-null float64
6. INSTALLMENTS\_PURCHASES 8950 non-null float64
7. CASH\_ADVANCE 8950 non-null float64
8. PURCHASES\_FREQUENCY 8950 non-null float64
9. ONEOFF\_PURCHASES\_FREQUENCY 8950 non-null float64
10. PURCHASES\_INSTALLMENTS\_FREQUENCY 8950 non-null float64
11. CASH\_ADVANCE\_FREQUENCY 8950 non-null float64
12. CASH\_ADVANCE\_TRX 8950 non-null int64
13. PURCHASES\_TRX 8950 non-null int64
14. CREDIT\_LIMIT 8949 non-null float64
15. PAYMENTS 8950 non-null float64
16. MINIMUM\_PAYMENTS 8637 non-null float64
17. PRC\_FULL\_PAYMENT 8950 non-null float64
18. TENURE 8950 non-null int64 dtypes: float64(14), int64(3), object(1) memory usage: 1.2+ MB

[4]:

Out[4]:

**BALANCE**

**BALANCE\_FREQUENCY**

**PURCHASES**

**ONEOFF\_PURCHASES**

**INSTALLMENTS\_PURCHASES**

**CASH\_ADVANCE**

**PURCHASES\_FREQUEN**

**count**

8950.000000

8950.000000

8950.000000

8950.000000

8950.000000

8950.000000

8950.0000

df

.

describe

()

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **mean** | 1564.474828 | 0.877271 | 1003.204834 | 592.437371 | 411.067645 | 978.871112 | 0.4903 |
| **std** | 2081.531879 | 0.236904 | 2136.634782 | 1659.887917 | 904.338115 | 2097.163877 | 0.4013 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0000 |
| **25%** | 128.281915 | 0.888889 | 39.635000 | 0.000000 | 0.000000 | 0.000000 | 0.0833 |
| **50%** | 873.385231 | 1.000000 | 361.280000 | 38.000000 | 89.000000 | 0.000000 | 0.5000 |
| **75%** | 2054.140036 | 1.000000 | 1110.130000 | 577.405000 | 468.637500 | 1113.821139 | 0.9166 |

Out[5]:

In

[5]:



**max**

19043.138560

1.000000

49039.570000

40761.250000

22500.000000

47137.211760

1.0000

**Missing\_Records**

**MINIMUM\_PAYMENTS**

313

*# Now we shall check the values that are missing and fill them by using proper procedure.*

**def**

null\_values

(

df

):

nv

**=**

pd

.

DataFrame

(

df

.

isnull

().

sum

()).

rename

(

columns

**=**

{

0

:

'Missing\_Records'

})

**return**

nv

[

nv

.

Missing\_Records

**>**

0

].

sort\_values

(

'Missing\_Records'

,

ascending

**=**

**False**

)

null\_values

(

df

)

**CREDIT\_LIMIT** 1

In

[6]:



Out[6]:

Missing\_Records 0.0

*# The null values can be filled with mean.*

df

[

'MINIMUM\_PAYMENTS'

]

**=**

df

[

'MINIMUM\_PAYMENTS'

].

fillna

(

df

.

MINIMUM\_PAYMENTS

.

mean

())

df

[

'CREDIT\_LIMIT'

]

**=**

df

[

'CREDIT\_LIMIT'

].

fillna

(

df

.

CREDIT\_LIMIT

.

mean

())

null\_values

(

df

).

sum

()

In

[7]:



In

[8]:



dtype: float64

Out[8]:

BALANCE 695

BALANCE\_FREQUENCY 1493

*# the cust\_id coloumn is not used so we can drop it, it is completely unrelevant in the data.*

df

**=**

df

.

drop

(

'CUST\_ID'

,

axis

**=**

1

)

#there are several outliers in the columns, but we won't use winsorize or any other techniques on them. Since we might have informed.

*#any other clusters can actually be represented.*

Q1

**=**

df

.

quantile

(

0.25

)

Q3

**=**

df

.

quantile

(

0.75

)

IQR

**=**

Q3

**-**

Q1

((

df

[

df

.

columns

]

**<**

(

Q1

**-**

1.5

**\***

IQR

))

**|**

(

df

[

df

.

columns

]

**>**

(

Q3

**+**

1.5

**\***

IQR

))).

sum

()

PURCHASES 808

ONEOFF\_PURCHASES 1013

INSTALLMENTS\_PURCHASES 867

CASH\_ADVANCE 1030

PURCHASES\_FREQUENCY 0

ONEOFF\_PURCHASES\_FREQUENCY 782

PURCHASES\_INSTALLMENTS\_FREQUENCY 0

CASH\_ADVANCE\_FREQUENCY 525

CASH\_ADVANCE\_TRX 804

PURCHASES\_TRX 766

CREDIT\_LIMIT 248

PAYMENTS 808

MINIMUM\_PAYMENTS 774

PRC\_FULL\_PAYMENT 1474 TENURE 1366 dtype: int64

In

[9]:



*# The task of standardization is carried out by StandardScaler. A dataset often contains variables with various scales.* *The task of standardization is carried out by StandardScaler. A dataset often contains variables with various scales.*

scaler

**=**

StandardScaler

()

df\_scl

**=**

scaler

.

fit\_transform

(

df

)

[10]:

In

[11]:



*# Rescaling real-valued numerical attributes into the range between 0 and 1 is known as normalization.*

*# For a model that depends on the magnitude of values, such as distance measurements, it is helpful to scale the input attributes.*

norm

**=**

normalize

(

df\_scl

)

*# Before clustering, we can apply both StandardScaler and Normalize to our data.*

df\_norm

**=**

pd

.

DataFrame

(

norm

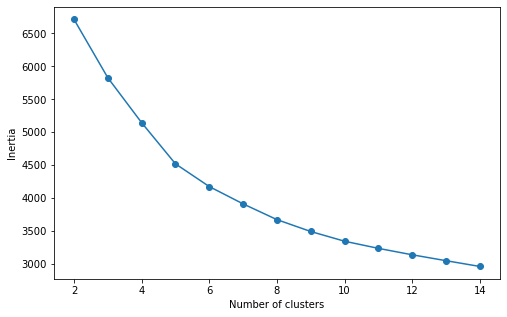
)

K MEANS CLUSTERING

In

[12]:





scores

**=**

[]

**for**

k

**in**

range

(

2

,

15

):

km

**=**

KMeans

(

n\_clusters

**=**

k

,

random\_state

**=**

123

)

km

**=**

km

.

fit

(

df\_norm

)

scores

.

append

(

km

.

inertia\_

)

dfk

**=**

pd

.

DataFrame

({

'Cluster'

:

range

(

2

,

15

)

,

'Score'

:

scores

})

plt

.

figure

(

figsize

**=**

(

8

,

5

))

plt

.

plot

(

dfk

[

'Cluster'

,

]

dfk

[

'Score'

]

,

marker

**=**

'o'

)

plt

.

xlabel

(

'Number of clusters'

)

plt

.

ylabel

(

'Inertia'

)

plt

.

show

()

The silhouette value gauges a point's cohesion with its own cluster in relation to neighboring clusters (separation).

In

[13]:



Silhouette score for 5 clusters k-means : 0.229

Silhouette score for 6 clusters k-means : 0.245

eans

(

n\_clusters

**=**

i

,

random\_state

**=**

123

).

fit\_predict

(

df\_norm

)

e score for {} clusters k-means : {} "

.

format

(

i

,

metrics

.

silhouette\_score

(

df\_norm

,

kmeans\_labels

,

metric

**=**

'euclidean'

).

round

(

3

)))

Silhouette score for 7 clusters k-means : 0.238

Silhouette score for 8 clusters k-means : 0.239

Silhouette score for 9 clusters k-means : 0.218

Silhouette score for 10 clusters k-means : 0.217

In the range of 6 to 8, the silhouette score values are reasonably near to one another. Let's take a look at another metric in light of the situation. The Davies Bouldin metric, where similarity is the ratio of within-cluster to between-cluster distances, is defined as the average similarity measure of each cluster with its most similar cluster. With a minimum score of 0, better clustering is indicated by lower numbers.

In

[14]:



Davies Bouldin Score:1.404

Davies Bouldin Score:1.354

**for**

i

**in**

[

6

,

7

,

8

]:

kmeans\_labels

**=**

KMeans

(

n\_clusters

**=**

i

,

random\_state

**=**

123

).

fit\_predict

(

df\_norm

)

print

(

'Davies Bouldin Score:'

**+**

str

(

metrics

.

davies\_bouldin\_score

(

df\_norm

,

kmeans\_labels

).

round

(

3

)))

Davies Bouldin Score:1.412

We wish to have a high Silhouette score, unlike Davies Bouldin. Therefore, according to the K-Means Algorithm, the ideal cluster numbers when evaluating the Elbow method and Silhouette score are 7. Therefore, I've decided that the K-means model's k values should be 7.

In

[15]:



kmeans\_labels

**=**

KMeans

(

n\_clusters

**=**

7

,

random\_state

**=**

123

).

fit\_predict

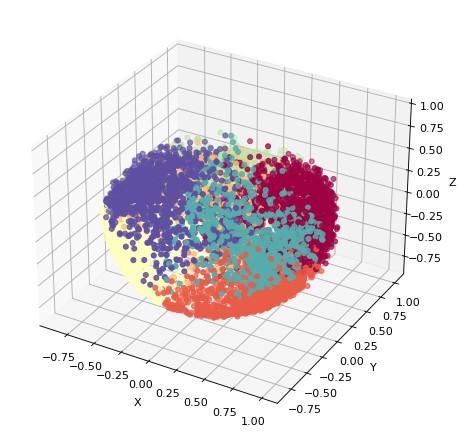
(

df\_norm

)

Let's now explore the "CC GENERAL" dataset in three dimensions. Therefore, PCA should be used first.

[16]:



pca

**=**

PCA

(

n\_components

**=**

3

).

fit\_transform

(

df\_norm

)

fig

**=**

plt

.

figure

(

figsize

**=**

(

12

,

7

,

)

dpi

**=**

80

,

facecolor

**=**

'w'

,

edgecolor

**=**

'k'

)

ax

**=**

plt

.

axes

(

projection

**=**

"3d"

)

ax

.

scatter3D

(

pca

.

T

[

0

]

,

pca

.

T

[

1

]

,

pca

.

T

[

2

]

,

c

**=**

kmeans\_labels

,

cmap

**=**

'Spectral'

)

xLabel

**=**

ax

.

set\_xlabel

(

'X'

)

yLabel

**=**

ax

.

set\_ylabel

(

'Y'

)

zLabel

**=**

ax

.

set\_zlabel

(

'Z'

)

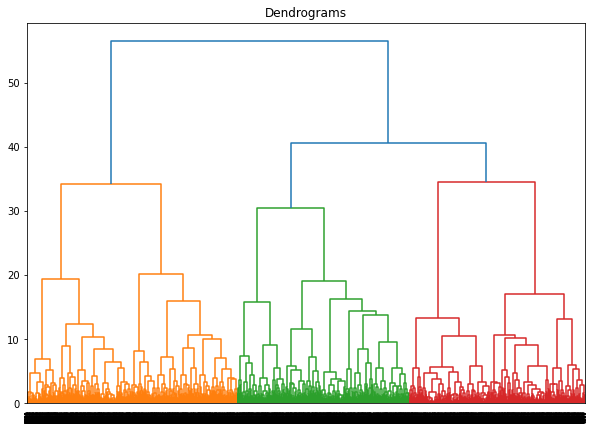
5.3 Hierarchical Clustering

A clustering method called hierarchical clustering seeks to establish a hierarchy of clusters inside the data that resembles a tree. We may apply a dendogram on this model to find the number of clusters, or n clusters.

In

[21]:





plt

.

figure

(

figsize

**=**

(

10

,

7

))

plt

.

title

(

"Dendrograms"

)

dend

**=**

shc

.

dendrogram

(

shc

.

linkage

(

df\_norm

,

method

**=**

'ward'

))

The samples are on the x-axis, and the separation between them is shown on the y-axis. The dendrogram can be clipped at a threshold of 38 because the blue line, which is the vertical line with the greatest distance, is visible.

The dendogram indicates that there are three clusters.

[22]: hcluster **=** AgglomerativeClustering(n\_clusters**=**3, affinity**=**'euclidean', linkage**=**'ward') hcp**=**hcluster.fit\_predict(df\_norm)

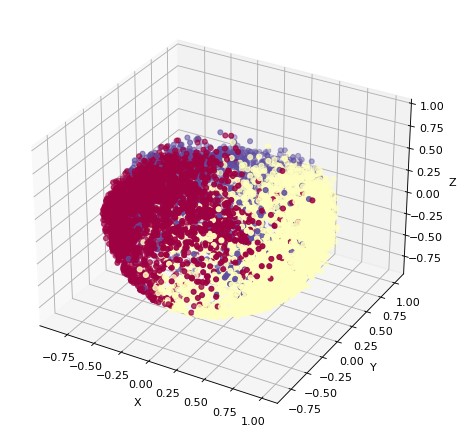
print('Silhouette Score for Hieararchial Clustering:'**+**str(metrics.silhouette\_score(df\_norm,hcp,metric**=**'euclidean'))) print('Davies Bouldin Score:'**+**str(metrics.davies\_bouldin\_score(df\_norm,hcp)))

Silhouette Score for Hieararchial Clustering:0.16269232126810304

Davies Bouldin Score:2.0178566980982713

In [23]:





fig

**=**

plt

.

figure

(

figsize

**=**

(

12

,

7

)

,

dpi

**=**

80

,

facecolor

**=**

'w'

,

edgecolor

**=**

'k'

)

ax

**=**

plt

.

axes

(

projection

**=**

"3d"

)

ax

.

scatter3D

(

pca

.

T

[

0

]

,

pca

.

T

[

1

]

,

pca

.

T

[

2

]

,

c

**=**

hcp

,

cmap

**=**

'Spectral'

)

xLabel

**=**

ax

.

set\_xlabel

(

'X'

)

yLabel

**=**

ax

.

set\_ylabel

(

'Y'

)

zLabel

**=**

ax

.

set\_zlabel

(

'Z'

)

DBSCAN Algorithm

DBSCAN is a density-based clustering technique, as its name suggests. Data with clusters of a comparable density are desirable, and density refers to the closeness of data points within a cluster.

First, we need select two parameters: epsilon, a positive number, and minPoints, a natural number. Then the model was made.

In

[24]:



In

[25]:



Out[25]:

**Eps**

**Min\_Samples**

**Number of Cluster**

**Silhouette Score**

**Davies Bouldin Score**

**16**

0.4

6.0

3.0

-0.03353

4.460939

results

**=**

pd

.

DataFrame

(

columns

**=**

[

'Eps'

,

'Min\_Samples'

,

'Number of Cluster'

,

'Silhouette Score'

])

**for**

i

**in**

range

(

1

,

12

):

**for**

j

**in**

range

(

1

,

12

):

dbscan\_cluster

**=**

DBSCAN

(

eps

**=**

i

**\***

0.2

,

min\_samples

**=**

j

)

clusters

**=**

dbscan\_cluster

.

fit\_predict

(

df\_norm

)

**if**

len

(

np

.

unique

(

clusters

))

**>**

2

:

results

**=**

results

.

append

({

'Eps'

:

i

**\***

0.2

,

'Min\_Samples'

:

j

,

'Number of Cluster'

:

len

(

np

.

unique

(

clusters

,

))

'Silhouette Score'

:

metrics

.

silhouette\_score

(

df\_norm

,

clusters

,

)

'Davies Bouldin Score'

:

metrics

.

davies\_bouldin\_score

(

df\_norm

,

clusters

,

)}

ignore\_index

**=**

**True**

)

results

.

sort\_values

(

'Silhouette Score'

,

ascending

**=**

**False**

)[:

5

]

**18** 0.6 2.0 4.0 -0.04625 3.857114

1. 0.4 4.0 5.0 -0.12267 3.361897
2. 0.4 5.0 5.0 -0.124499 3.304983

**10** 0.2 11.0 18.0 -0.238821 1.344078

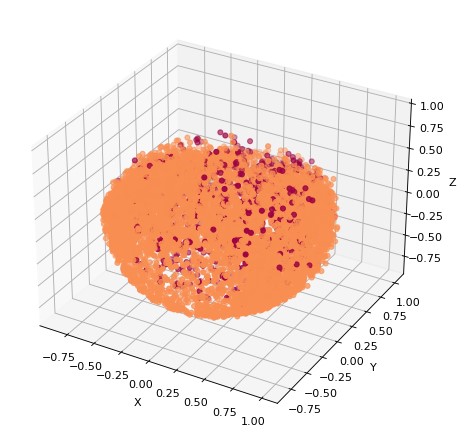
DBSCAN appears to be the wrong technique for this dataset. The values that we have chosen for eps and min samples are 0.4 and 4, respectively.

[27]:

In

[28]:





dbscan\_cluster

**=**

DBSCAN

(

eps

**=**

0.4

,

min\_samples

**=**

4

)

db\_clusters

**=**

dbscan\_cluster

.

fit\_predict

(

df\_norm

)

fig

**=**

plt

.

figure

(

figsize

**=**

(

12

,

7

)

,

dpi

**=**

80

,

facecolor

**=**

'w'

,

edgecolor

**=**

'k'

)

ax

**=**

plt

.

axes

(

projection

**=**

"3d"

)

ax

.

scatter3D

(

pca

.

T

[

0

]

,

pca

.

T

[

1

]

,

pca

.

T

[

2

,

]

c

**=**

db\_clusters

,

cmap

**=**

'Spectral'

)

xLabel

**=**

ax

.

set\_xlabel

(

'X'

)

yLabel

**=**

ax

.

set\_ylabel

(

'Y'

)

zLabel

**=**

ax

.

set\_zlabel

(

'Z'

)

There don't seem to be any discernible distributions in the data for clustering.

Comparison of Results

In

[32]:



In

[33]:



**Algorithms**

**Davies Bouldin**

**Silhouette Score**

**0**

K-Means

1.354323

0.237578

algorithms

**=**

[

"K-Means"

,

"Hierarchical Clustering"

,

"DBSCAN"

]

*# Silhouette Score*

ss

**=**

[

metrics

.

silhouette\_score

(

df\_norm

,

kmeans\_labels

)

,

metrics

.

silhouette\_score

(

df\_norm

,

hcp

)

,

metrics

.

silhouette\_score

(

df\_norm

,

db\_clusters

)]

*# Davies Bouldin Score*

db

**=**

[

metrics

.

davies\_bouldin\_score

(

df\_norm

,

kmeans\_labels

)

,

metrics

.

davies\_bouldin\_score

(

df\_norm

,

hcp

,

)

metrics

.

davies\_bouldin\_score

(

df\_norm

,

db\_clusters

)]

comprsn

**=**

{

"Algorithms"

:

algorithms

,

"Davies Bouldin"

:

db

,

"Silhouette Score"

:

ss

}

compdf

**=**

pd

.

DataFrame

(

comprsn

)

display

(

compdf

.

sort\_values

(

by

**=**

[

"Silhouette Score"

]

,

ascending

**=**

**False**

))

|  |  |  |  |
| --- | --- | --- | --- |
| **1** | Hierarchical Clustering | 2.017857 | 0.162692 |
| **2** | DBSCAN | 3.361897 | -0.122670 |

Finally, We have tried 3 algorithm. K-Means has the best Silhouette and Davies Bouldin score. For this reason, K-Means Algorithm is more suitable for customer segmentation. Thus we have 7 customer types. Let’s try to understand behaviours or labels of customers.

Out[34]:

In

[34]:



**3**

**0**

**4**

**1**

**6**

**5**

**2**

**Number of Customers**

1865

1653

1551

1305

1150

772

654

df

[

'Clusters'

]

**=**

list

(

kmeans\_labels

)

customers

**=**

pd

.

DataFrame

(

df

[

'Clusters'

].

value\_counts

()).

rename

(

columns

**=**

{

'Clusters'

:

'Number of Customers'

})

customers

.

T

Out[35]:

[35]:

In

[36]:



**BALANCE**

**BALANCE\_FREQUENCY**

**PURCHASES\_FREQUENCY**

**PURCHASES\_INSTALLMENTS\_FREQUENCY**

**CASH\_ADVANCE\_FREQUENCY**

**PURCHAS**

**mean**

1564.5

0.9

0.5

0.4

0.1

Out[36]:

means

**=**

pd

.

DataFrame

(

df

.

describe

().

loc

[

'mean'

])

means

.

T

.

iloc

,

[

[:

0

,

1

,

6

,

8

,

9

,

11

,

12

,

16

]].

round

(

1

)

df

.

set\_index

(

'Clusters'

)

grouped

**=**

df

.

groupby

(

by

**=**

'Clusters'

).

mean

().

round

(

1

)

grouped

.

iloc

[:

,

[

0

,

1

,

6

,

8

,

9

,

11

,

12

,

16

]]

**BALANCE BALANCE\_FREQUENCY PURCHASES\_FREQUENCY PURCHASES\_INSTALLMENTS\_FREQUENCY CASH\_ADVANCE\_FREQUENCY PURCH**

**Clusters**

1. 2020.6 1.0 0.9 0.6 0.1
2. 947.8 1.0 0.9 0.8 0.0
3. 871.9 0.8 0.4 0.3 0.2
4. 1259.5 1.0 0.1 0.0 0.1
5. 4047.4 1.0 0.2 0.2 0.4
6. 99.6 0.9 0.8 0.7 0.0

In

[37]:



**6**

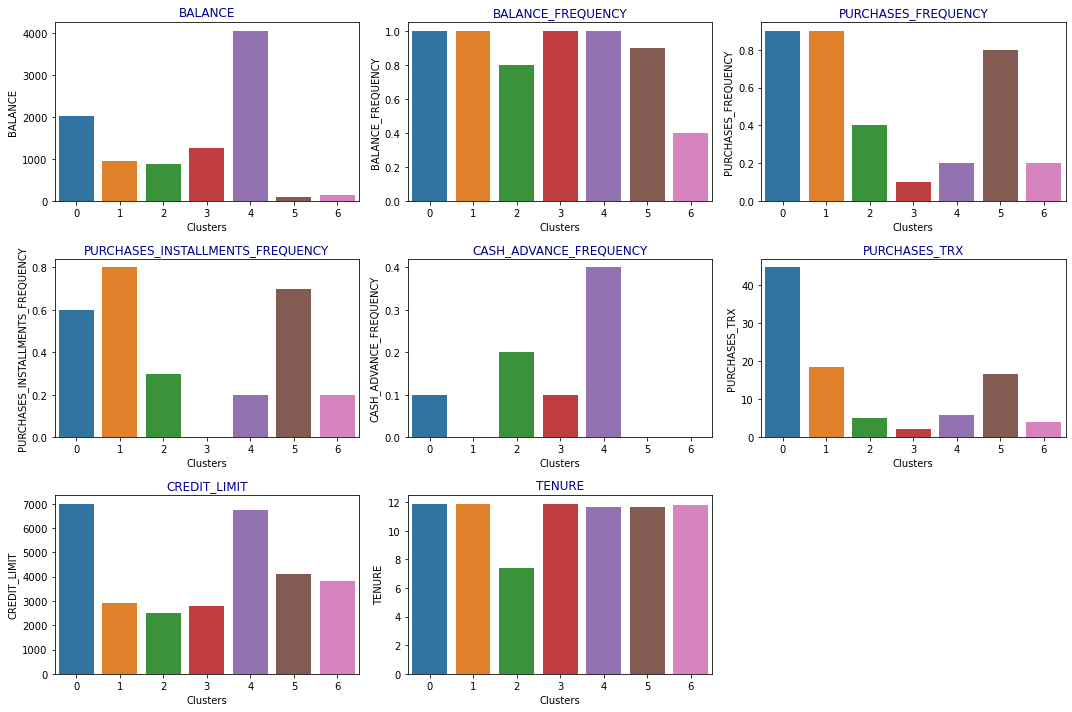
131.8

0.4

0.2

0.2

0.0



features

**=**

[

"BALANCE"

,

"BALANCE\_FREQUENCY"

,

"PURCHASES\_FREQUENCY"

,

"PURCHASES\_INSTALLMENTS\_FREQUENCY"

,

"CASH\_ADVANCE\_FREQUENCY"

,

"PU

plt

.

figure

(

figsize

**=**

(

15

,

10

))

**for**

i

,

j

**in**

enumerate

(

features

):

plt

.

subplot

(

3

,

3

,

i

**+**

1

)

sns

.

barplot

(

grouped

.

index

,

grouped

[

j

])

plt

.

title

(

j

,

fontdict

**=**

{

'color'

:

'darkblue'

})

plt

.

tight\_layout

()

plt

.

show

()

We have chosen some columns that are significant to identify the clusters.

Cluster 0 : Customers with greater credit limits and longer tenure who purchase more frequently and tend to pay in installments.

Cluster 1 : Pretty low frequency of purchases and balance. They have a lower credit limit and use credit cards infrequently.

Cluster 2 : The most clients and the least quantity of card usage belong to this group. Long-term as well as inactive customers.

Cluster 3 : High likelihood of installment payments, higher frequency of purchases, and above-average tenure times.

Cluster 4 : The greatest balance amount, but not the best in terms of purchasing frequency. Have a bigger credit limit than others and tend to pay in cash. They dislike making purchases.

Cluster 5 : Second-highest purchasing frequency and larger propensity for installment payments. They have been consumers for a while.

Cluster 6 : In this group, there are the fewest number of customers who make purchases less frequently than normal and for shorter periods of time.

To summarize, First, we started with the preparation of the data. Then, we used methods for clustering. We ultimately chose to employ K-Means as the model after analyzing different clustering models. The data was then separated into seven clusters since it is simple to predict customer behavior using these clusters. The clusters do, however, each have unique traits.In [ ]: 