In [1]:
```
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')
```

In [2]:
```
df=pd.read_csv("CC GENERAL.csv")
df.head()
```

Out[2]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQ
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
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0

```
print('This data set has {} rows and {} columns.\n'.format(df.shape[0],df.shape[1]))
df.info()
```

In [3]: There are 18 columns and 8950 rows in this data collection.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 18 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
  0   CUST_ID                           8950 non-null   object
  1   BALANCE                           8950 non-null   float64
  2   BALANCE_FREQUENCY                 8950 non-null   float64
  3   PURCHASES                         8950 non-null   float64
  4   ONEOFF_PURCHASES                  8950 non-null   float64
  5   INSTALLMENTS_PURCHASES            8950 non-null   float64
  6   CASH_ADVANCE                      8950 non-null   float64
  7   PURCHASES_FREQUENCY               8950 non-null   float64
  8   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
  9   PURCHASES_INSTALLMENTS_FREQUENCY  8950 non-null   float64
  10  CASH_ADVANCE_FREQUENCY            8950 non-null   float64
  11  CASH_ADVANCE_TRX                  8950 non-null   int64
  12  PURCHASES_TRX                     8950 non-null   int64
  13  CREDIT_LIMIT                      8949 non-null   float64
  14  PAYMENTS                          8950 non-null   float64
  15  MINIMUM_PAYMENTS                  8637 non-null   float64
  16  PRC_FULL_PAYMENT                  8950 non-null   float64
  17  TENURE                            8950 non-null   int64  dtypes: float64(14), int64(3), object(1) memory usage:
 1.2+ MB
```

[4]:
```
df.describe()
```

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
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

In

75%	2054.140036	1.000000	1110.130000	577.405000	468.637500	1113.821139	0.9166
max	19043.138560	1.000000	49039.570000	40761.250000	22500.000000	47137.211760	1.0000

In [5]:
```
# 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)
```

Out[5]:

	Missing_Records
MINIMUM_PAYMENTS	313
CREDIT_LIMIT	1

In [6]:
```
# 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()
```

Out[6]:
```
Missing_Records    0.0
dtype: float64
```

In [7]:
```
# 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)
```

In [8]:
```
#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()
```

Out[8]:
```
BALANCE                             695
BALANCE_FREQUENCY                  1493
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 st
scaler=StandardScaler()
df_scl=scaler.fit_transform(df)
```

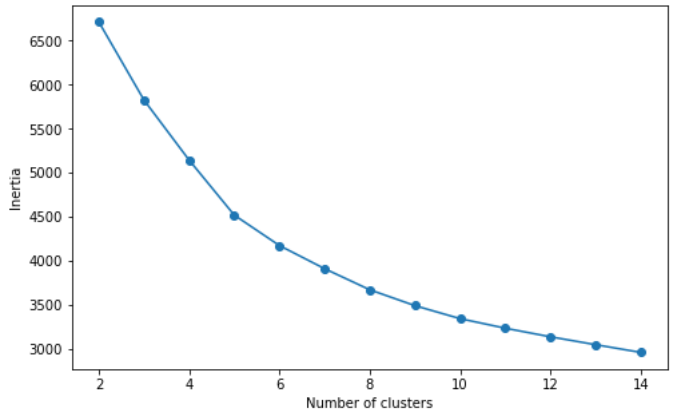
[10]:
```
# Rescaling real-valued numerical attributes into the range between 0 and 1 is known as normalization.
# For a model that depends on the magnitude of values, such as distance measurements, it is helpful to scale the input attributes.
norm=normalize(df_scl)
```

In [11]:
```
# Before clustering, we can apply both StandardScaler and Normalize to our data.
df_norm=pd.DataFrame(norm)
```

K MEANS CLUSTERING

In

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]:
```
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 5 clusters k-means : 0.229
Silhouette score for 6 clusters k-means : 0.245
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]:
```
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:1.404
Davies Bouldin Score:1.354
Davies Bouldin Score:1.412
```

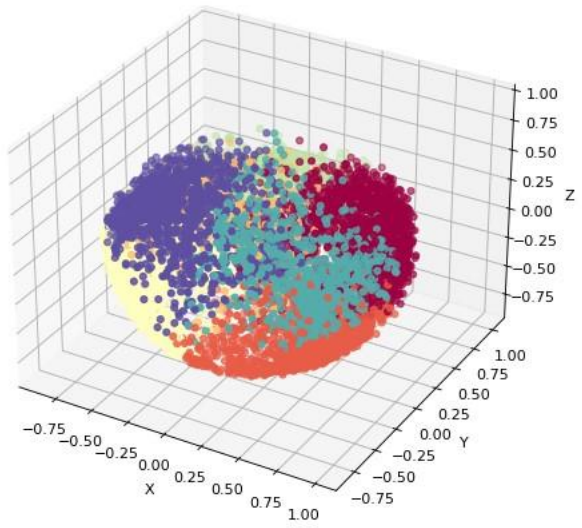
We wish to have a high Silhouette score, unlike Davies Bouldin. Therefore, according to the K-Means Algorithm, the ideal cluster numbers when evaluating the Elbow method and Silhouette score are 7. Therefore, I've decided that the K-means model's k values should be 7.

In [15]:
```
kmeans_labels=KMeans(n_clusters=7,random_state=123).fit_predict(df_norm)
```

In

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

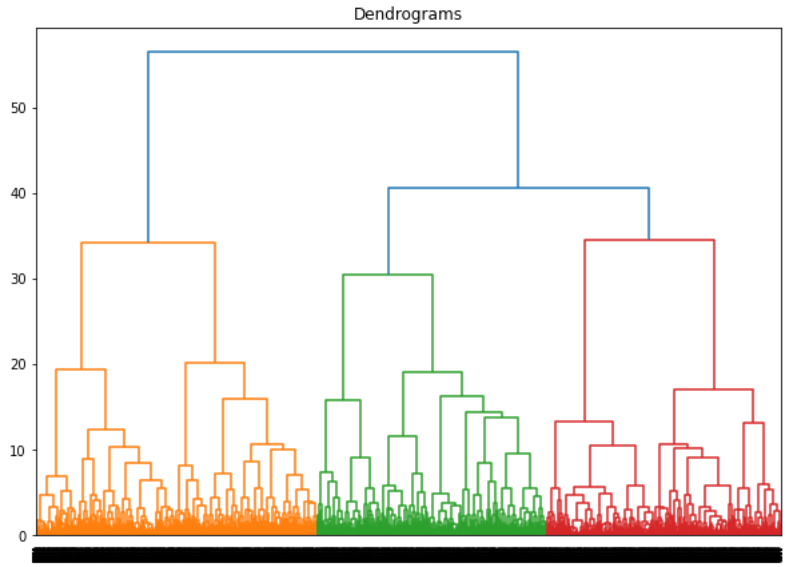
[16]:
```
pca = PCA(n_components=3).fit_transform(df_norm)
fig = plt.figure(figsize=(12, 7), dpi=80, facecolor='w', edgecolor='k')
ax = plt.axes(projection="3d")
ax.scatter3D(pca.T[0],pca.T[1],pca.T[2],c=kmeans_labels,cmap='Spectral')

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.

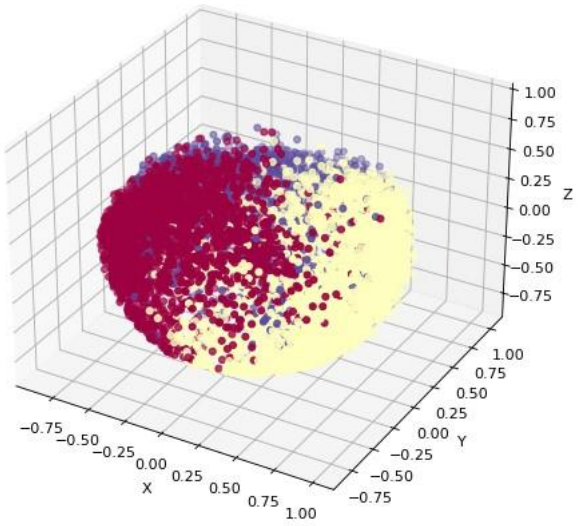
[22]:
```
hcluster = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')  hcp=hcluster.fit_predict(df_norm)
print('Silhouette Score for Hiearchial Clustering:'+str(metrics.silhouette_score(df_norm,hcp,metric='euclidean')))
print('Davies Bouldin Score:'+str(metrics.davies_bouldin_score(df_norm,hcp)))
```

```
Silhouette Score for Hiearchial Clustering:0.16269232126810304
```

In

```
Davies Bouldin Score:2.0178566980982713
```

```
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')
```



In [23]:

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]:

```
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)
```

In [25]:

```
results.sort_values('Silhouette Score',ascending=False)[:5]
```

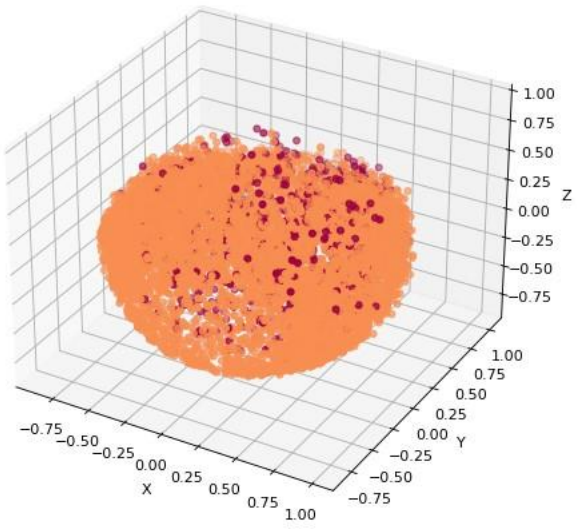
Out[25]:

	Eps	Min_Samples	Number of Cluster	Silhouette Score	Davies Bouldin Score
16	0.4	6.0	3.0	-0.03353	4.460939
18	0.6	2.0	4.0	-0.04625	3.857114
14	0.4	4.0	5.0	-0.12267	3.361897
15	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.

In [27]:
```
dbscan_cluster = DBSCAN(eps=0.4, min_samples=4)
db_clusters=dbscan_cluster.fit_predict(df_norm)
```

In [28]:
```
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]:
```
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)]
```

In [33]:
```
comprsn={"Algorithms":algorithms,"Davies Bouldin":db,"Silhouette Score":ss}
compdf=pd.DataFrame(comprsn)
display(compdf.sort_values(by=["Silhouette Score"], ascending=False))
```

	Algorithms	Davies Bouldin	Silhouette Score
0	K-Means	1.354323	0.237578
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.

In [34]:
```
df['Clusters']=list(kmeans_labels)
customers=pd.DataFrame(df['Clusters'].value_counts()).rename(columns={'Clusters':'Number of Customers'})
customers.T
```

Out[34]:

	3	0	4	1	6	5	2
Number of Customers	1865	1653	1551	1305	1150	772	654

In [35]:
```
means=pd.DataFrame(df.describe().loc['mean'])
means.T.iloc[:,[0,1,6,8,9,11,12,16]].round(1)
```

	BALANCE	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	PURCHAS
mean	1564.5	0.9	0.5	0.4	0.1	

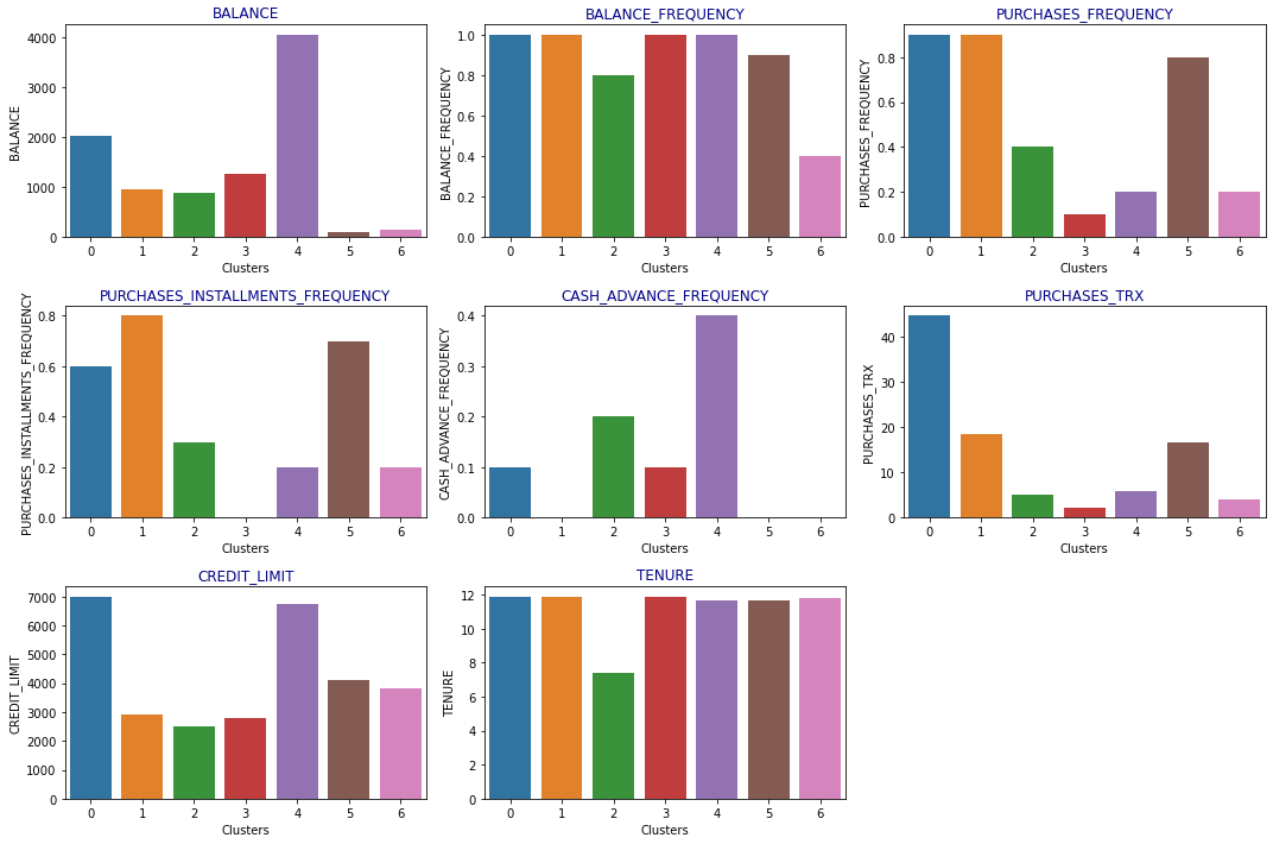
In [36]:
```
df.set_index('Clusters')
grouped=df.groupby(by='Clusters').mean().round(1)
grouped.iloc[:,[0,1,6,8,9,11,12,16]]
```

Out[36]:
Out[35]:

Clusters	BALANCE	BALANCE_FREQUENCY	PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	PURCH
0	2020.6	1.0	0.9	0.6	0.1	
1	947.8	1.0	0.9	0.8	0.0	
2	871.9	0.8	0.4	0.3	0.2	
3	1259.5	1.0	0.1	0.0	0.1	
4	4047.4	1.0	0.2	0.2	0.4	
5	99.6	0.9	0.8	0.7	0.0	
6	131.8		0.4		0.2	

	0.2	0.2	0.0

In [37]:
```
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()
```



In

To find the clusters, we have selected a few important columns.

Cluster 0: Customers with a large credit limit and a lengthy tenure who make purchases most frequently and tend to pay in installments.

Cluster 1: The balance and frequency of purchases are quite low. 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 : This group has the most patrons yet the least number of card usage. Customers who have been inactive for a long time.

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 : Both the second-highest purchasing frequency and the tendency to pay in installments are higher. They are devoted customers.

Cluster 6 : The smallest number of consumers in this category are those who make purchases less frequently than the average and do so for a brief period 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 [ ]: