otala8scf

May 11, 2023

0.1 Segmentación de clientes_Clustering

Este caso requiere desarrollar una segmentación de clientes para definir una estrategia de marketing. El conjunto de datos de muestra resume el comportamiento de uso de alrededor de 9000 titulares de tarjetas de crédito activos durante los últimos 6 meses. El archivo está a nivel de cliente con 18 variables de comportamiento.

A continuación se presenta el Diccionario de datos para el conjunto de datos de tarjeta de crédito:

CUST_ID: Identificación del titular de la tarjeta de crédito (Categórico).

BALANCE: Importe del saldo que queda en su cuenta para realizar compras.

BALANCE_FREQUENCY: Frecuencia con la que se actualiza el saldo, puntuación entre 0 y 1 (1 = se actualiza con frecuencia, 0 = no se actualiza con frecuencia)

PURCHASES: Importe de las compras realizadas desde la cuenta.

ONEOFF_PURCHASES: Importe máximo de compras realizadas de una sola vez.

INSTALLMENTS_PURCHASES: Importe de las compras realizadas a plazos.

CASH_ADVANCE: Efectivo adelantado por el usuario.

PURCHASES_FREQUENCY: Frecuencia con la que se realizan las compras, puntuación entre 0 y 1 (1 = compras frecuentes, 0 = compras poco frecuentes)

ONEOFF_PURCHASES_FREQUENCY: Frecuencia con la que se realizan las compras de una sola vez (1 = compras frecuentes, 0 = compras poco frecuentes)

PURCHASES_INSTALLMENTS_FREQUENCY: Frecuencia con la que se realizan compras a plazos (1 = se realizan con frecuencia, 0 = no se realizan con frecuencia)

CASH_ADVANCE_FREQUENCY: Frecuencia con la que se paga en efectivo por adelantado.

CASH_ADVANCE_TRX: Número de transacciones realizadas con "Efectivo por adelantado"

PURCHASES_TRX: Número de transacciones de compra realizadas.

CREDIT_LIMIT: Límite de la tarjeta de crédito del usuario.

PAYMENTS: Importe de los pagos realizados por el usuario.

MINIMUM_PAYMENTS: Importe mínimo de pagos realizados por el usuario.

PRC_FULL_PAYMENT: Porcentaje del pago total abonado por el usuario.

TENURE: Tenencia del servicio de tarjeta de crédito para el usuario.

[1]:
```
# Importaciones necesarias
import pandas as pd
import io
import matplotlib.pyplot as plt
import numpy as np
from sklearn.discriminant_analysis import StandardScaler
import seaborn as sns
from sklearn.metrics import confusion_matrix
from scipy import cluster
import sklearn.neighbors
from sklearn.decomposition import PCA
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import KMeans
from yellowbrick.cluster import SilhouetteVisualizer
from sklearn.impute import KNNImputer

import warnings
warnings.filterwarnings(action="ignore")
```

[2]:
```
# Importamos el
url1 = 'https://raw.githubusercontent.com/Ruben11040/Proyectos_Colab/main/
  ↪Segmentaci%C3%B3n%20de%20clientes/CC%20GENERAL.csv'
dfdata = pd.read_csv(url1)
```

[3]:
```
# Vista del dataframe sin modificaciónes
dfdata
```

[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	
…	…	…	…	…	…	
8945	C19186	28.493517	1.000000	291.12	0.00	
8946	C19187	19.183215	1.000000	300.00	0.00	
8947	C19188	23.398673	0.833333	144.40	0.00	
8948	C19189	13.457564	0.833333	0.00	0.00	
8949	C19190	372.708075	0.666667	1093.25	1093.25	

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	\
0	95.40	0.000000	0.166667	
1	0.00	6442.945483	0.000000	
2	0.00	0.000000	1.000000	
3	0.00	205.788017	0.083333	
4	0.00	0.000000	0.083333	

```
…                       …           …                    …
8945                  291.12    0.000000             1.000000
8946                  300.00    0.000000             1.000000
8947                  144.40    0.000000             0.833333
8948                    0.00   36.558778             0.000000
8949                    0.00  127.040008             0.666667

      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
…                          …                                …
8945                    0.000000                          0.833333
8946                    0.000000                          0.833333
8947                    0.000000                          0.666667
8948                    0.000000                          0.000000
8949                    0.666667                          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
…                      …                   …              …           …
8945                0.000000                 0              6        1000.0
8946                0.000000                 0              6        1000.0
8947                0.000000                 0              5        1000.0
8948                0.166667                 2              0         500.0
8949                0.333333                 2             23        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
…           …                …                …     …
8945   325.594462         48.886365          0.500000       6
8946   275.861322               NaN          0.000000       6
8947    81.270775         82.418369          0.250000       6
8948    52.549959         55.755628          0.250000       6
8949    63.165404         88.288956          0.000000       6

[8950 rows x 18 columns]
```

Estudio de los Datos El proyecto que vamos a realizar a continuación, será un proyecto de Clustering en este proyecto lo que generar es difearentes grupos o clusters donde habrán diferentes tipos de clientes según sus datos bancarios. Ahora pasaremos al estudio de los datos para poder comprobar que en ninguno de estos haya un problema y poder ajustar el dataframe de la mejor manera posible.

[4]:
```
dfdata.isnull().sum().sort_values(ascending=False)
```

[4]:
```
MINIMUM_PAYMENTS                    313
CREDIT_LIMIT                          1
CUST_ID                               0
BALANCE                               0
PRC_FULL_PAYMENT                      0
PAYMENTS                              0
PURCHASES_TRX                         0
CASH_ADVANCE_TRX                      0
CASH_ADVANCE_FREQUENCY                0
PURCHASES_INSTALLMENTS_FREQUENCY      0
ONEOFF_PURCHASES_FREQUENCY            0
PURCHASES_FREQUENCY                   0
CASH_ADVANCE                          0
INSTALLMENTS_PURCHASES                0
ONEOFF_PURCHASES                      0
PURCHASES                             0
BALANCE_FREQUENCY                     0
TENURE                                0
dtype: int64
```

Como podemos comprobar el valor de MINIMUM_PAYMENTS tiene muchos valores nulos con lo cual lo que voy a hacer es sustituir todos los valores nulos por la media pero, aparte si el valor de PAYMENTS es igual a 0 lo pondré a dicho valor. Porque para mi no tiene sentido que tu tengas un PAYMENTS de 0 y que tu MINIMUM_PAYMENT sea de 800 alomejor. Y para el CREDIT_LIMIT utilizaré la media.

[5]:
```
dfdata.loc[(dfdata['PAYMENTS'] == 0) & (dfdata['MINIMUM_PAYMENTS'].isnull()),␣
 ↪'MINIMUM_PAYMENTS'] = 0
dfdata.loc[(dfdata['MINIMUM_PAYMENTS'].isnull()),'MINIMUM_PAYMENTS'] =␣
 ↪dfdata['MINIMUM_PAYMENTS'].mean()
dfdata.loc[(dfdata['CREDIT_LIMIT'].
 ↪isnull()),'CREDIT_LIMIT']=dfdata['CREDIT_LIMIT'].mean()
```

Usamos el mismo código para comprobar que ya no hay ningún valor nulo

[6]:
```
dfdata.isnull().sum().sort_values(ascending=False).head()
```

[6]:
```
CUST_ID             0
BALANCE             0
PRC_FULL_PAYMENT    0
```

```
MINIMUM_PAYMENTS    0
PAYMENTS            0
dtype: int64
```

[7]:
```
# Eliminamos la columna 'CUST_ID' debido a que no nos proporciona ninguna␣
 ↪información

dfdata.drop('CUST_ID', axis = 1, inplace = True)
```

[8]:
```
dfdata.describe()
```

[8]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	\
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	1564.474828	0.877271	1003.204834	592.437371	
std	2081.531879	0.236904	2136.634782	1659.887917	
min	0.000000	0.000000	0.000000	0.000000	
25%	128.281915	0.888889	39.635000	0.000000	
50%	873.385231	1.000000	361.280000	38.000000	
75%	2054.140036	1.000000	1110.130000	577.405000	
max	19043.138560	1.000000	49039.570000	40761.250000	

	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	\
count	8950.000000	8950.000000	8950.000000	
mean	411.067645	978.871112	0.490351	
std	904.338115	2097.163877	0.401371	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.083333	
50%	89.000000	0.000000	0.500000	
75%	468.637500	1113.821139	0.916667	
max	22500.000000	47137.211760	1.000000	

	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	\
count	8950.000000	8950.000000	
mean	0.202458	0.364437	
std	0.298336	0.397448	
min	0.000000	0.000000	
25%	0.000000	0.000000	
50%	0.083333	0.166667	
75%	0.300000	0.750000	
max	1.000000	1.000000	

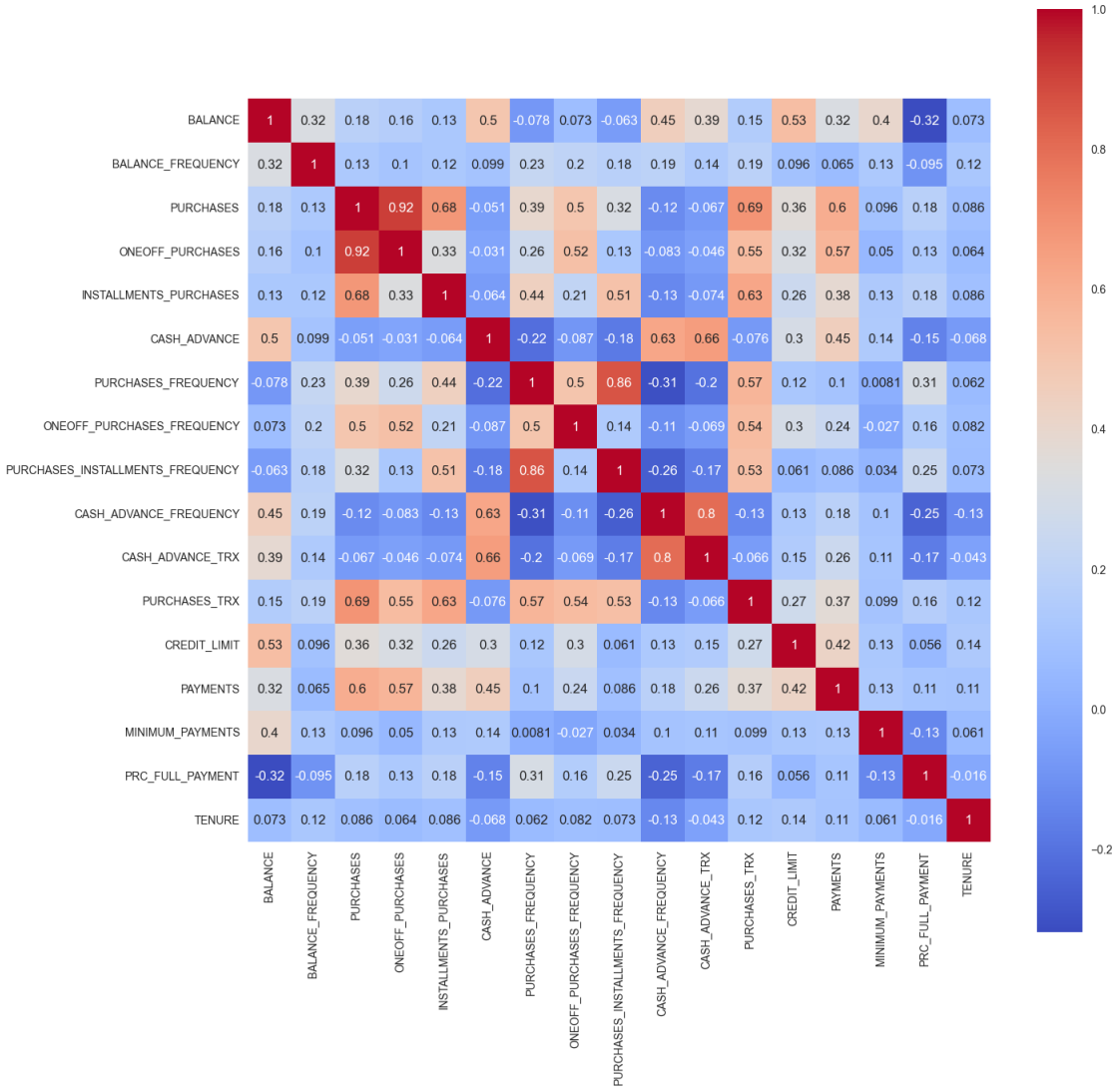
	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	\
count	8950.000000	8950.000000	8950.000000	8950.000000	
mean	0.135144	3.248827	14.709832	4494.449450	
std	0.200121	6.824647	24.857649	3638.612411	
min	0.000000	0.000000	0.000000	50.000000	
25%	0.000000	0.000000	1.000000	1600.000000	

50%	0.000000	0.000000	7.000000	3000.000000
75%	0.222222	4.000000	17.000000	6500.000000
max	1.500000	123.000000	358.000000	30000.000000

	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
count	8950.000000	8950.000000	8950.000000	8950.000000
mean	1733.143852	840.841715	0.153715	11.517318
std	2895.063757	2334.765209	0.292499	1.338331
min	0.000000	0.000000	0.000000	6.000000
25%	383.276166	164.653643	0.000000	12.000000
50%	856.901546	299.924288	0.000000	12.000000
75%	1901.134317	819.114121	0.142857	12.000000
max	50721.483360	76406.207520	1.000000	12.000000

0.1.1 Matríz de Correlación

[9]:
```
# Crear el heatmap
fig, ax = plt.subplots(figsize=(15,15))
sns.heatmap(dfdata.corr(), cmap='coolwarm', annot=True, square=True)
# Encontré un parámetro  que te lo realiza todo como una huella calorifica, es␣
 ↪decir,
# los que tienen más correlación te los pone en rojo y los que menos te los␣
 ↪pone en azul
plt.show()
```



Como podemos comprobar en la matriz de correlación, hay varias parejas de valores que tienen un alto valor de correlación, para ello nos vamos a fijar en la parejas con el valor con más correlación, pero esta vez no vamos a eliminar ninguna de ella por el hecho de que en mi opinión, el caracter que tenia pensado eliminar que era el *ONEOFF_PURCHASES* pero pensandolo mejor creo que no lo haré debido a que como explico al principio del proyecto, esta característica y la de *PURCHASES*, es que el *ONEOFF_PURCHASES* nos dice el gasto máximo de una sola vez. Mientras que el *PURCHASES* te dice el gasto total, por eso yo considero que ambos son importantes y son cosas a tener en cuenta a la hora del estudio.

0.1.2 Dendograma

Normalizamos los valores.

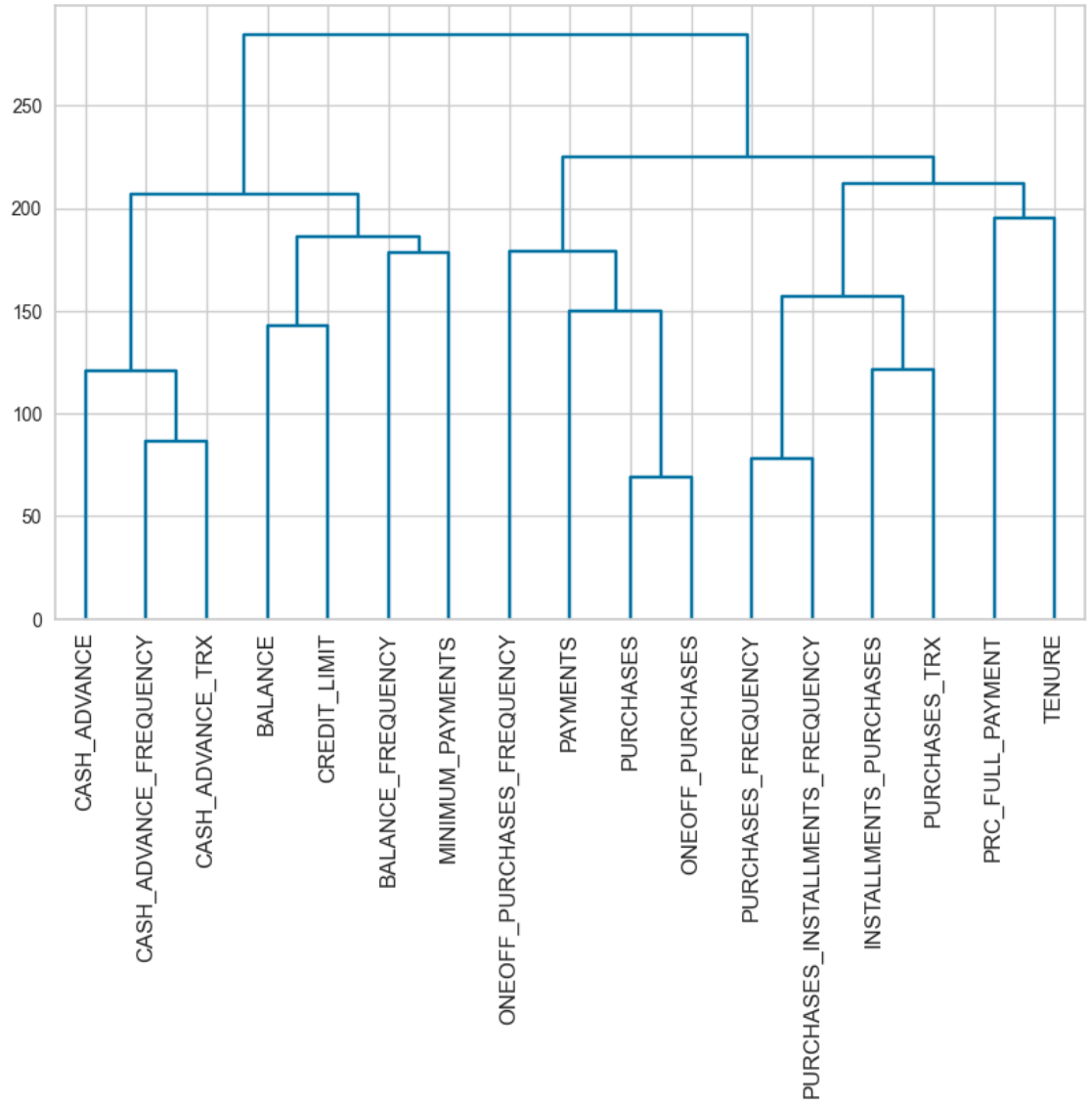
[10]:
```
scale = StandardScaler()
X = scale.fit_transform(dfdata)
X.shape
```

[10]: (8950, 17)

[11]:
```
df_T = X.T
```

[12]:
```
# Para obtener el dendograma con el agrupamiento por características, se hace␣
 ↪necesario obtener la transpuesta de la matriz del dataset

dist = sklearn.neighbors.DistanceMetric.get_metric('euclidean')
D = dist.pairwise(df_T)
avD = np.average(D)
print("Distancia Media\t {:6.2f}".format(avD))

# 2. Constuimos el dendograma
plt.figure(figsize=(10, 6))
threshold_2 = 7
# Seleccionamos como distancia intercluster el vecino más alejado (complete)
clusters = cluster.hierarchy.linkage(D, method='complete')
cluster.hierarchy.dendrogram(clusters, color_threshold=threshold_2,␣
 ↪labels=dfdata.columns.to_list(), leaf_rotation=90)
plt.show()
```

```
Distancia Media  113.01
```



Como podemos comprobar tanto en la matriz de correlación y en el dendograma hay otra pareja de valores que voy a eliminar uno de los dos valores ya que ambas características tienen una alta correlación y dependen mucho de la otra. Las dos características son *CASH_ADVANCE_FRECUENCY* y *CASH _ADVANCE_TRX*, *de ambas voy a elegir quedarme con la de* CASH_ADVANCE_FRECUENCY* porque considero que es más importante la frecuencia con que lo hagas que las veces que lo hayas hecho. Por lo cual eliminaré *CASH_ADVANCE_TRX*.

[13]:
```
# Eliminamos la columna 'CASH_ADVANCE_TRX'
dfdata.drop('CASH_ADVANCE_TRX', axis = 1, inplace = True)
```

Buscando outliers Usando el rango intercuartílico (IQR), podemos seguir el siguiente enfoque para encontrar outliers:

- Calcula el primer y tercer cuartil (Q1 y Q3).
- Luego, evalúa el rango intercuartílico, IQR = Q3-Q1.
- Estima el límite inferior, el límite inferior = Q1*1.5.
- Estima el límite superior, el límite superior = Q3*1.5.
- Los puntos de datos que se encuentran fuera del límite inferior y el límite superior son outliers.

[14]:
```
#Esta función se encarga de localizar los outliers usando los quantiles

def outlier_percent(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    minimum = Q1 - (1.5 * IQR)
    maximum = Q3 + (1.5 * IQR)
    num_outliers =  np.sum((data < minimum) |(data > maximum))
    num_total = data.count()
    return (num_outliers/num_total)*100
```

En esta parte me voy a encargar de localizar los outliers pasarlos a valor NAN y a posterior eliminarlos del dataframe.

[15]:
```
for column in dfdata.columns:
    data = dfdata[column]
    percent = str(round(outlier_percent(data), 2))
    print(f'Outliers in "{column}": {percent}%')
```

```
Outliers in "BALANCE": 7.77%
Outliers in "BALANCE_FREQUENCY": 16.68%
Outliers in "PURCHASES": 9.03%
Outliers in "ONEOFF_PURCHASES": 11.32%
Outliers in "INSTALLMENTS_PURCHASES": 9.69%
Outliers in "CASH_ADVANCE": 11.51%
Outliers in "PURCHASES_FREQUENCY": 0.0%
Outliers in "ONEOFF_PURCHASES_FREQUENCY": 8.74%
Outliers in "PURCHASES_INSTALLMENTS_FREQUENCY": 0.0%
Outliers in "CASH_ADVANCE_FREQUENCY": 5.87%
Outliers in "PURCHASES_TRX": 8.56%
Outliers in "CREDIT_LIMIT": 2.77%
Outliers in "PAYMENTS": 9.03%
Outliers in "MINIMUM_PAYMENTS": 9.45%
Outliers in "PRC_FULL_PAYMENT": 16.47%
Outliers in "TENURE": 15.26%
```

[16]:
```
for column in dfdata.columns:
    data = dfdata[column]

    Q1 = data.quantile(0.15)
    Q3 = data.quantile(0.85)
    IQR = Q3 - Q1
    minimum = Q1 - (1.5 * IQR)
    maximum = Q3 + (1.5 * IQR)

    outliers = ((data < minimum) |(data > maximum))
    dfdata[column].loc[outliers] = np.nan

dfdata.isna().sum()
```

[16]:
```
BALANCE                             186
BALANCE_FREQUENCY                   147
PURCHASES                           335
ONEOFF_PURCHASES                    420
INSTALLMENTS_PURCHASES              323
CASH_ADVANCE                        329
PURCHASES_FREQUENCY                   0
ONEOFF_PURCHASES_FREQUENCY            0
PURCHASES_INSTALLMENTS_FREQUENCY      0
CASH_ADVANCE_FREQUENCY              125
PURCHASES_TRX                       335
CREDIT_LIMIT                         43
PAYMENTS                            351
MINIMUM_PAYMENTS                    368
PRC_FULL_PAYMENT                      0
TENURE                              765
dtype: int64
```

Al principio mi intención era eliminar todos los outliers pero de esta forma elimino más de la mitad de todas las filas.

Voy a usar para rellenarlo *KNN imputer* que hace que los valores faltantes de cada muestra se imputan usando el valor medio de los n_neighbors vecinos más cercanos encontrados en el conjunto de entrenamiento.

[17]:
```
imputer = KNNImputer()
dfdata = pd.DataFrame(imputer.fit_transform(dfdata), columns=dfdata.columns)
dfdata.isna().sum()
```

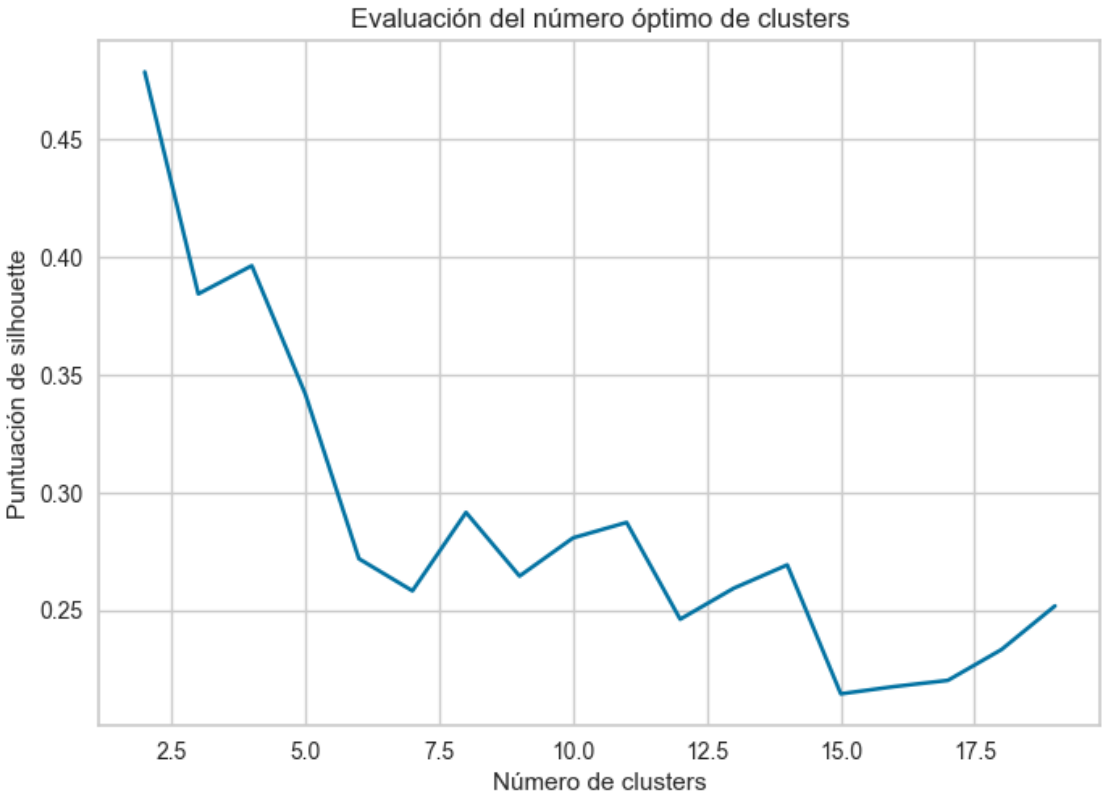
[17]:
```
BALANCE                     0
BALANCE_FREQUENCY           0
PURCHASES                   0
ONEOFF_PURCHASES            0
INSTALLMENTS_PURCHASES      0
```

```
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY          0
PURCHASES_INSTALLMENTS_FREQUENCY    0
CASH_ADVANCE_FREQUENCY              0
PURCHASES_TRX                       0
CREDIT_LIMIT                        0
PAYMENTS                            0
MINIMUM_PAYMENTS                    0
PRC_FULL_PAYMENT                    0
TENURE                              0
dtype: int64
```

K-MEANS

[18]:
```
values_silhouette = []
for k in range(2,20):
    grouper = KMeans(n_clusters=k, n_init='auto')
    labels = grouper.fit_predict(dfdata)
    mean_silhouette = silhouette_score(dfdata, labels)
    values_silhouette.append(mean_silhouette)

# encontrar el número óptimo de clusters
optimal_k = 2 + values_silhouette.index(max(values_silhouette))
print(f' El mejor valor de Silhouette Score es {max(values_silhouette)} con k =␣
 ↪{optimal_k}')

# trazar los valores de la puntuación de silhouette
plt.plot(range(2,20), values_silhouette, 'bx-')
plt.xlabel('Número de clusters')
plt.ylabel('Puntuación de silhouette')
plt.title('Evaluación del número óptimo de clusters')
plt.show()
```

```
El mejor valor de Silhouette Score es 0.47843164426676127 con k = 2
```



[19]:
```
# x clusters
group = KMeans(n_clusters = optimal_k, n_init= 'auto')

# Haz x grupos y crea una variable con las etiquetas.
group.fit(dfdata)
labels = group.labels_
print("Labels K-means: ",labels)
```

```
Labels K-means:  [1 0 0 … 1 1 1]
```

[20]:
```
# Silhouette Coefficient K-mean

print("The Silhouette Coefficient K-mean is:", silhouette_score(dfdata, labels))
```

```
The Silhouette Coefficient K-mean is: 0.47760508159793497
```

PCA

[21]:
```
X = scale.fit_transform(dfdata)
```

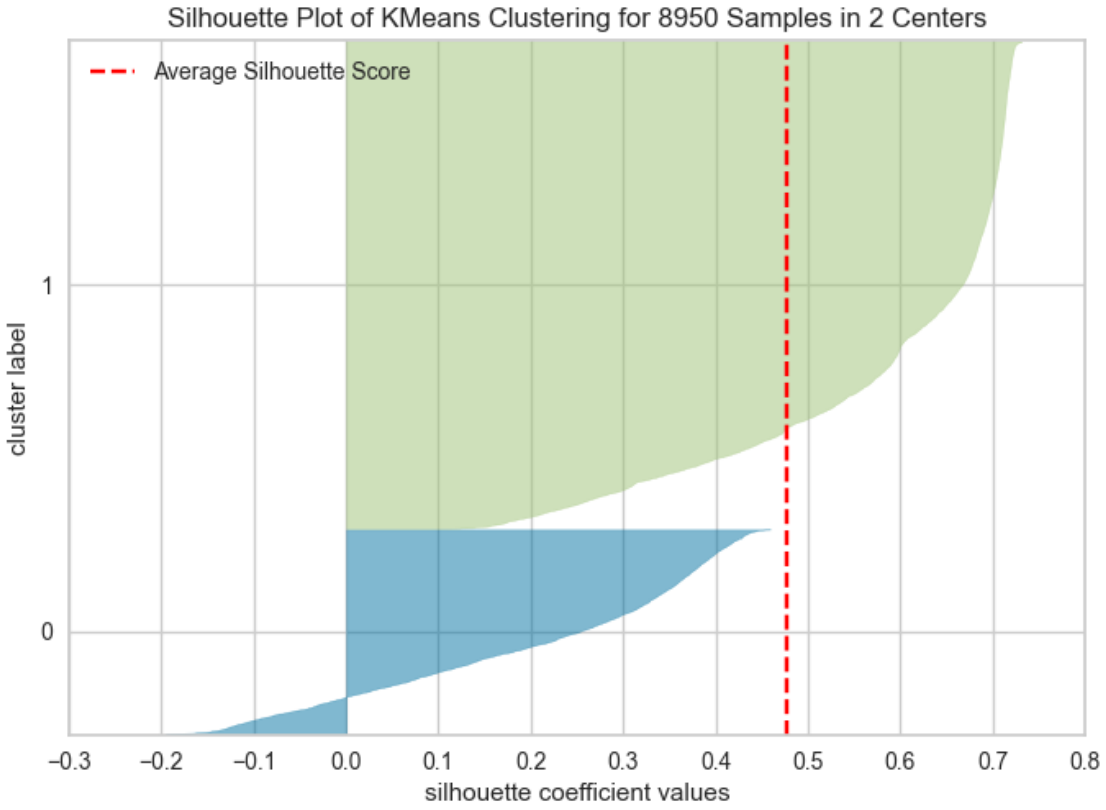
[22]:
```
dist = 1 - cosine_similarity(X)
pca = PCA(2)
```

```
pca.fit(dist)
X_PCA = pca.transform(dist)
X_PCA.shape
```

[22]: (8950, 2)

[23]:
```
x, y = X_PCA[:, 0], X_PCA[:, 1]
df = pd.DataFrame({'x': x, 'y':y, 'label':labels})
groups = df.groupby('label')
```

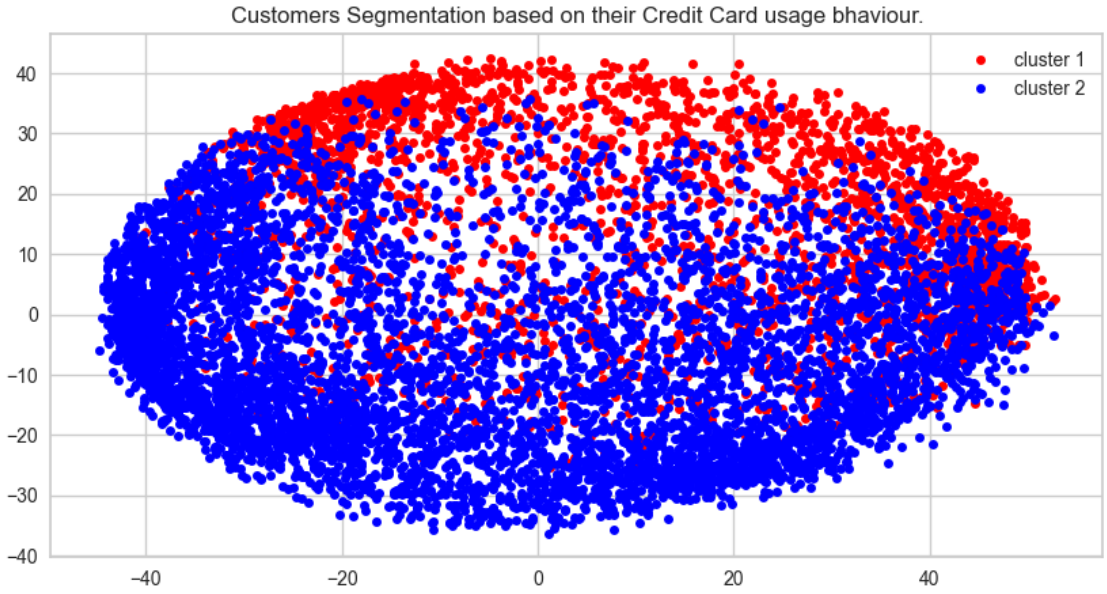
[24]:
```
visualizer = SilhouetteVisualizer(group, colors='yellowbrick')
visualizer.fit(dfdata)
visualizer.show()
```



[24]: <Axes: title={'center': 'Silhouette Plot of KMeans Clustering for 8950 Samples in 2 Centers'}, xlabel='silhouette coefficient values', ylabel='cluster label'>

[25]:
```
colors = {0: 'red',
          1: 'blue',}

names = {0: 'cluster 1',
```

```
         1: 'cluster 2',}

fig, ax = plt.subplots(figsize=(10, 5))

for name, group in groups:
    ax.plot(group.x, group.y, marker='o', linestyle='', ms=5,
            color=colors[name],label=names[name], mec='none')
    ax.set_aspect('auto')
    ax.
 ↪tick_params(axis='x',which='both',bottom='off',top='off',labelbottom='off')
    ax.tick_params(axis= 'y',which='both',left='off',top='off',labelleft='off')

ax.legend()
ax.set_title("Customers Segmentation based on their Credit Card usage bhaviour.
 ↪")
plt.show()
```



[26]:
```
dfdata["cluster"] = labels
dfdata.groupby("cluster").describe()
```

[26]:

	BALANCE						\
	count	mean	std	min	25%	50%	
cluster							
0	2649.0	2890.758097	2293.311443	0.0	541.240795	2689.040221	
1	6301.0	858.845067	923.024955	0.0	77.285458	559.944819	

	BALANCE_FREQUENCY	…	\

```
                 75%          max             count      mean  …
cluster                                                       …
0        4802.507595  7888.028118            2649.0  0.936505  …
1        1347.505936  4972.108843            6301.0  0.867043  …

        PRC_FULL_PAYMENT       TENURE                                        \
                     75%  max   count       mean       std   min   25%   50%
cluster
0                0.090909  1.0  2649.0  11.920951  0.335268  10.0  12.0  12.0
1                0.166667  1.0  6301.0  11.873258  0.398239  10.0  12.0  12.0


          75%   max
cluster
0        12.0  12.0
1        12.0  12.0

[2 rows x 128 columns]
```

[27]:
```
dfdata.groupby("cluster")["BALANCE"].describe()
```

[27]:
```
           count         mean          std  min         25%          50%  \
cluster
0         2649.0  2890.758097  2293.311443  0.0  541.240795  2689.040221
1         6301.0   858.845067   923.024955  0.0   77.285458   559.944819

                 75%          max
cluster
0        4802.507595  7888.028118
1        1347.505936  4972.108843
```

[28]:
```
# Obtener la ruta completa a la carpeta de descargas del usuario
import os


desktop_path = os.path.join(os.path.expanduser('~'), 'Downloads')

# Pasar el dataframe a csv y guardar el archivo en la carpeta de descargas
dfdata.to_excel(os.path.join(desktop_path, 'clientes.xlsx'), index=False)
```