Actividad modulo #29 - Clustering

In [ ]:
```
# Impot Libraries
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
import os

# Librerias para Clustering
from sklearn.cluster import KMeans
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.cluster.hierarchy import fcluster
from scipy.cluster.hierarchy import single, cophenet
from scipy.spatial.distance import pdist, squareform
```

I took the dataset from:

- https://www.kaggle.com/datasets/arjunbhasin2013/ccdata

Here is the Data Dictionary for the Credit Card dataset :

- CUST_ID : Identification of Credit Card holder (Categorical)
- BALANCE : Balance amount left in their account to make purchases (
- BALANCE_FREQUENCY : How frequently the Balance is updated, score between 0 and 1 (1 = frequently updated, 0 = not frequently updated)
- PURCHASES : Amount of purchases made from account
- ONEOFF_PURCHASES : Maximum purchase amount done in one-go
- INSTALLMENTS_PURCHASES : Amount of purchase done in installment
- CASH_ADVANCE : Cash in advance given by the user
- PURCHASES_FREQUENCY : How frequently the Purchases are being made, score between 0 and 1 (1 = frequently purchased, 0 = not frequently purchased)
- ONEOFFPURCHASESFREQUENCY : How frequently Purchases are happening in one-go (1 = frequently purchased, 0 = not frequently purchased)
- PURCHASESINSTALLMENTSFREQUENCY : How frequently purchases in installments are being done (1 = frequently done, 0 = not frequently done)
- CASHADVANCEFREQUENCY : How frequently the cash in advance being paid
- CASHADVANCETRX : Number of Transactions made with "Cash in Advanced"
- PURCHASES_TRX : Numbe of purchase transactions made
- CREDIT_LIMIT : Limit of Credit Card for user
- PAYMENTS : Amount of Payment done by user
- MINIMUM_PAYMENTS : Minimum amount of payments made by user
- PRCFULLPAYMENT : Percent of full payment paid by user
- TENURE : Tenure of credit card service for user

Exploratory Data Analysis y Analisis Univariado

In [ ]:
```
# import dataset
os.chdir('E:\WORK IN PROGRESS\Data Analytics course\parte 2 python\week 29')
# Se usa la funcion read_csv para leer el archivo . csv
```

```
# Validar los campos y sus rangos
df = pd.read_csv('CC_GENERAL.csv')

df.head(5)
```

Out[ ]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	0.000000	0.083333
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	0.000000	0.000000
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	1.000000	0.000000
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	0.083333	0.000000
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	0.083333	0.000000

In [ ]:

```
df = df.round(4)
df.head(5)
```

Out[ ]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY
0	C10001	40.9007	0.8182	95.40	0.00	95.4	0.0000	0.1667	0.0000	0.0833
1	C10002	3202.4674	0.9091	0.00	0.00	0.0	6442.9455	0.0000	0.0000	0.0000
2	C10003	2495.1489	1.0000	773.17	773.17	0.0	0.0000	1.0000	1.0000	0.0000
3	C10004	1666.6705	0.6364	1499.00	1499.00	0.0	205.7880	0.0833	0.0833	0.0000
4	C10005	817.7143	1.0000	16.00	16.00	0.0	0.0000	0.0833	0.0833	0.0000

In [ ]:

```
df.shape
```

Out[ ]:

```
(8950, 18)
```

insights:

- la primera c0lumna se puede eliminar (ID column).

In [ ]:

```
df.info()
```

```
<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
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
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
```

Insights:

- Todas las columnas estan ya en formato numerico
- Dos columna tienen valores nulos: "Minimum_payments"
- No es necesario revisar el balanceo de las clases porque no hay variables categoricas

In [ ]:
```
df.nunique()
```

Out[ ]:
```
CUST_ID                             8950
BALANCE                             8865
BALANCE_FREQUENCY                     43
PURCHASES                           6203
ONEOFF_PURCHASES                    4014
INSTALLMENTS_PURCHASES              4452
CASH_ADVANCE                        4323
PURCHASES_FREQUENCY                   47
ONEOFF_PURCHASES_FREQUENCY            47
PURCHASES_INSTALLMENTS_FREQUENCY      47
CASH_ADVANCE_FREQUENCY                54
CASH_ADVANCE_TRX                      65
PURCHASES_TRX                        173
CREDIT_LIMIT                         205
PAYMENTS                            8709
MINIMUM_PAYMENTS                    8633
PRC_FULL_PAYMENT                      47
TENURE                                 7
dtype: int64
```

In [ ]:
```
df.describe().T
```

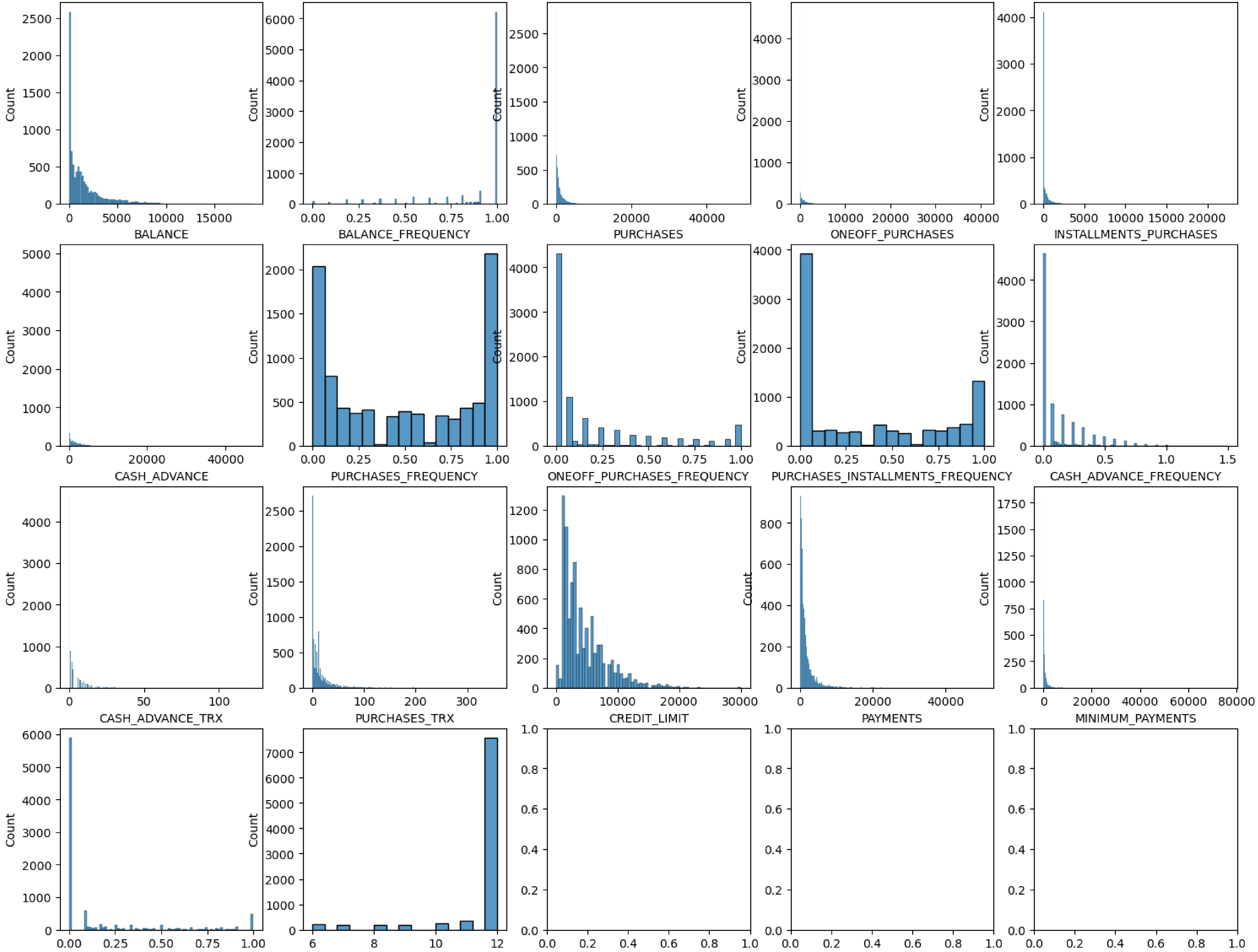
Out[ ]:

	count	mean	std	min	25%	50%	75%	max
BALANCE	8950.0	1564.474828	2081.531879	0.0000	128.281950	873.38525	2054.140000	19043.1386
BALANCE_FREQUENCY	8950.0	0.877272	0.236906	0.0000	0.888900	1.00000	1.000000	1.0000
PURCHASES	8950.0	1003.204834	2136.634782	0.0000	39.635000	361.28000	1110.130000	49039.5700
ONEOFF_PURCHASES	8950.0	592.437371	1659.887917	0.0000	0.000000	38.00000	577.405000	40761.2500
INSTALLMENTS_PURCHASES	8950.0	411.067645	904.338115	0.0000	0.000000	89.00000	468.637500	22500.0000
CASH_ADVANCE	8950.0	978.871113	2097.163877	0.0000	0.000000	0.00000	1113.821175	47137.2118
PURCHASES_FREQUENCY	8950.0	0.490349	0.401373	0.0000	0.083300	0.50000	0.916700	1.0000
ONEOFF_PURCHASES_FREQUENCY	8950.0	0.202455	0.298338	0.0000	0.000000	0.08330	0.300000	1.0000
PURCHASES_INSTALLMENTS_FREQUENCY	8950.0	0.364438	0.397449	0.0000	0.000000	0.16670	0.750000	1.0000
CASH_ADVANCE_FREQUENCY	8950.0	0.135142	0.200122	0.0000	0.000000	0.00000	0.222200	1.5000
CASH_ADVANCE_TRX	8950.0	3.248827	6.824647	0.0000	0.000000	0.00000	4.000000	123.0000
PURCHASES_TRX	8950.0	14.709832	24.857649	0.0000	1.000000	7.00000	17.000000	358.0000
CREDIT_LIMIT	8949.0	4494.449450	3638.815726	50.0000	1600.000000	3000.00000	6500.000000	30000.0000
PAYMENTS	8950.0	1733.143852	2895.063757	0.0000	383.276125	856.90155	1901.134300	50721.4834
MINIMUM_PAYMENTS	8637.0	864.206542	2372.446607	0.0192	169.123700	312.34390	825.485500	76406.2075
PRC_FULL_PAYMENT	8950.0	0.153713	0.292500	0.0000	0.000000	0.00000	0.142900	1.0000
TENURE	8950.0	11.517318	1.338331	6.0000	12.000000	12.00000	12.000000	12.0000

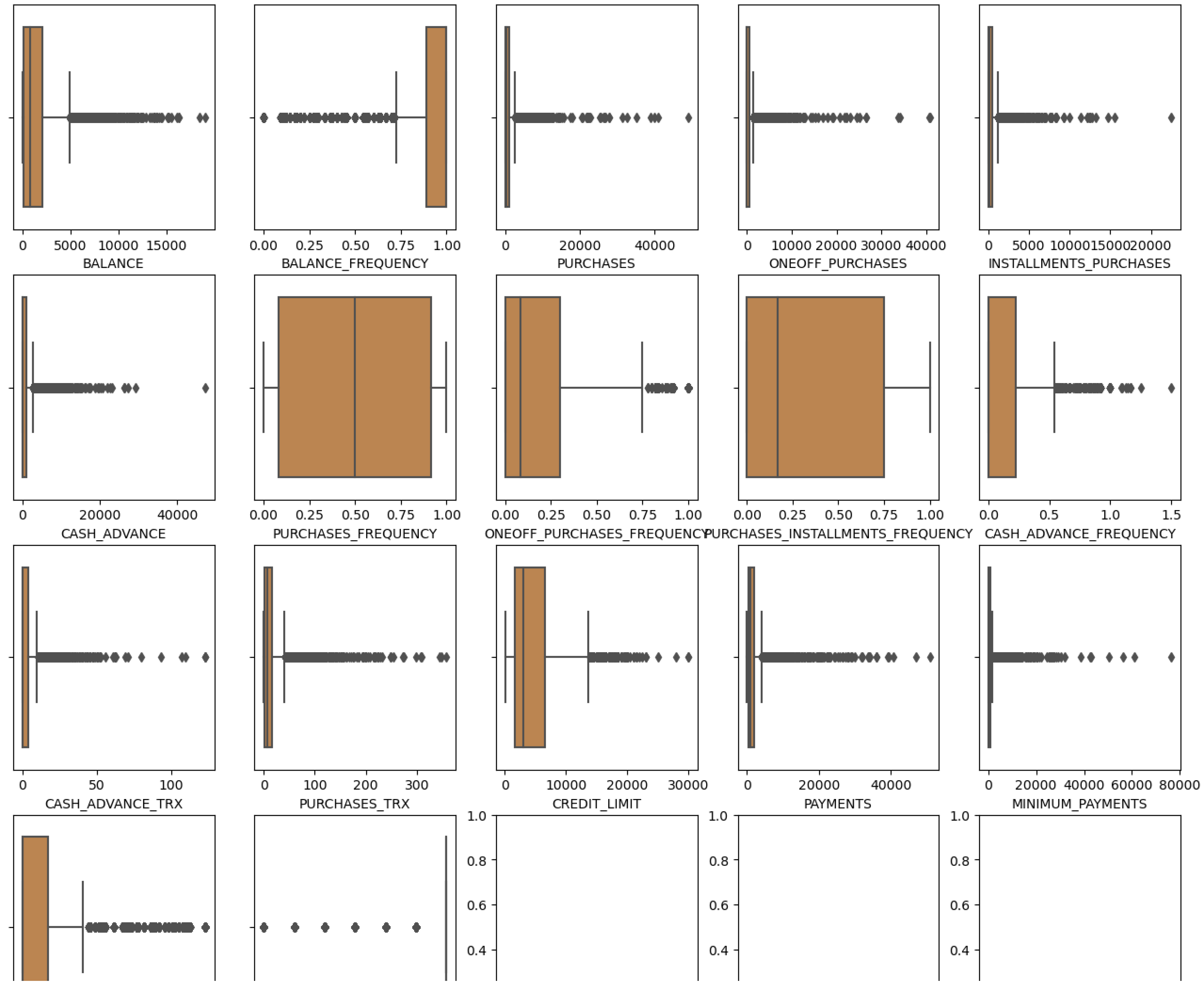
In [ ]:
```
cols=df.columns.to_list()
cols.remove('CUST_ID')
cols
```

Out[ ]:
```
['BALANCE',
 'BALANCE_FREQUENCY',
 'PURCHASES',
 'ONEOFF_PURCHASES',
 'INSTALLMENTS_PURCHASES',
 'CASH_ADVANCE',
 'PURCHASES_FREQUENCY',
 'ONEOFF_PURCHASES_FREQUENCY',
 'PURCHASES_INSTALLMENTS_FREQUENCY',
 'CASH_ADVANCE_FREQUENCY',
 'CASH_ADVANCE_TRX',
 'PURCHASES_TRX',
 'CREDIT_LIMIT',
 'PAYMENTS',
 'MINIMUM_PAYMENTS',
 'PRC_FULL_PAYMENT',
 'TENURE']
```

In [ ]:
```
# Grafica exploratoria de todas las columnas
fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(18,18))
for i,column in enumerate(cols):
    sns.histplot(df[column],ax=axes[i//5,i%5],kde=False)
```


Count
BALANCE
BALANCE_FREQUENCY
PURCHASES
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS

In [ ]:
```
fig,axes= plt.subplots(nrows=5,ncols=5, figsize=(16,18))
for i,column in enumerate (cols):
    sns.boxplot(x=df[column], color='peru',ax=axes[i//5,i%5])
```

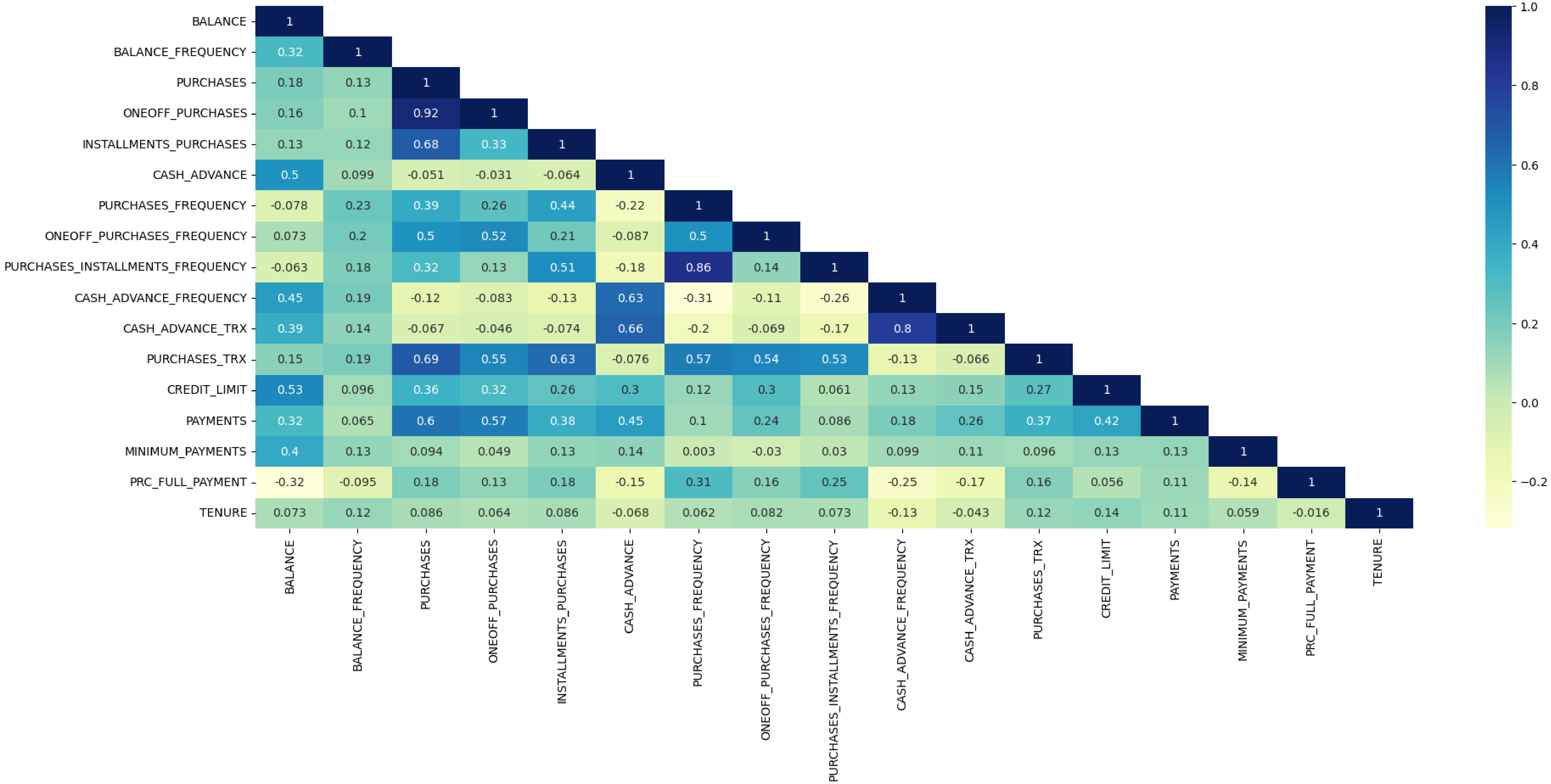

0 5000 10000 15000
BALANCE
0.00 0.25 0.50 0.75 1.00
BALANCE_FREQUENCY
0 20000 40000
PURCHASES
0 10000 20000 30000 40000
ONEOFF_PURCHASES
0 5000 10000 15000 20000
INSTALLMENTS_PURCHASES
0 20000 40000
CASH_ADVANCE
0.00 0.25 0.50 0.75 1.00
PURCHASES_FREQUENCY
0.00 0.25 0.50 0.75 1.00
ONEOFF_PURCHASES_FREQUENCY
0.00 0.25 0.50 0.75 1.00
PURCHASES_INSTALLMENTS_FREQUENCY
0.0 0.5 1.0 1.5
CASH_ADVANCE_FREQUENCY
0 50 100
CASH_ADVANCE_TRX
0 100 200 300
PURCHASES_TRX
0 10000 20000 30000
CREDIT_LIMIT
0 20000 40000
PAYMENTS
0 20000 40000 60000 80000
MINIMUM_PAYMENTS
1.0 0.8 0.6 0.4
1.0 0.8 0.6 0.4
1.0 0.8 0.6 0.4

Insights:

- La mayoria (por no decir todas) de las variables tienen un sesgo considerable con uotliers.
- Se puede realizar una transformacion con log.

Correlación

In [ ]:
```
# Grafica de correlacion
plt.figure(figsize=(22,8))
corr_df = corr = df.corr(method='pearson')
df_lt= corr_df.where(np.tril(np.ones(corr_df.shape)).astype(bool))
hmap=sns.heatmap(df_lt, cmap='YlGnBu',annot=True)
```



Insights:

- Hay dos coeficientes que indican que hay varaibles que estan altamente correlacionadas:
    - "PURCHASES" y "ONEOFF_PURCHASES" (0.92)
    - "PURCHASES_FREQUENCY" y "PURCHASES_INSTALLMENTS_FREQUENCY" (0.86)

Feature engineering

In [ ]:
```
df2 = df.copy()
```

In [ ]:
```
df3 = df.copy()
```

- Eliminación de columna (Id)

In [ ]:
```
df2.drop(columns=['CUST_ID'],inplace=True)
df2.sample(5)
```

Out[ ]:

	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_
5304	11.1169	1.0000	168.00	0.00	168.00	0.0000	1.0000	0.0000	1.0000	
7774	1420.6564	0.8750	0.00	0.00	0.00	1773.7695	0.0000	0.0000	0.0000	
3391	962.8555	0.9091	1839.69	1501.05	338.64	0.0000	0.8333	0.2500	0.5833	
8320	1079.1067	1.0000	0.00	0.00	0.00	390.4850	0.0000	0.0000	0.0000	
4348	5259.1404	0.8889	2659.35	813.00	1846.35	4851.3920	0.8889	0.2222	0.7778	

- Imputacion de datos para las dos columnas con valores nulos

In [ ]:
```
df2.isnull().sum()
```

Out[ ]:
```
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
```

In [ ]:
```
print(df2.columns[df2.isnull().any()])
```

```
Index(['CREDIT_LIMIT', 'MINIMUM_PAYMENTS'], dtype='object')
```

In [ ]:
```
fill_col=df2.columns[df2.isnull().any()].to_list()
```

In [ ]:
```
for col in fill_col:
    df2[col].fillna(value=df2[col].mean(),inplace=True)
```

In [ ]:
```
df2.isnull().sum()
```

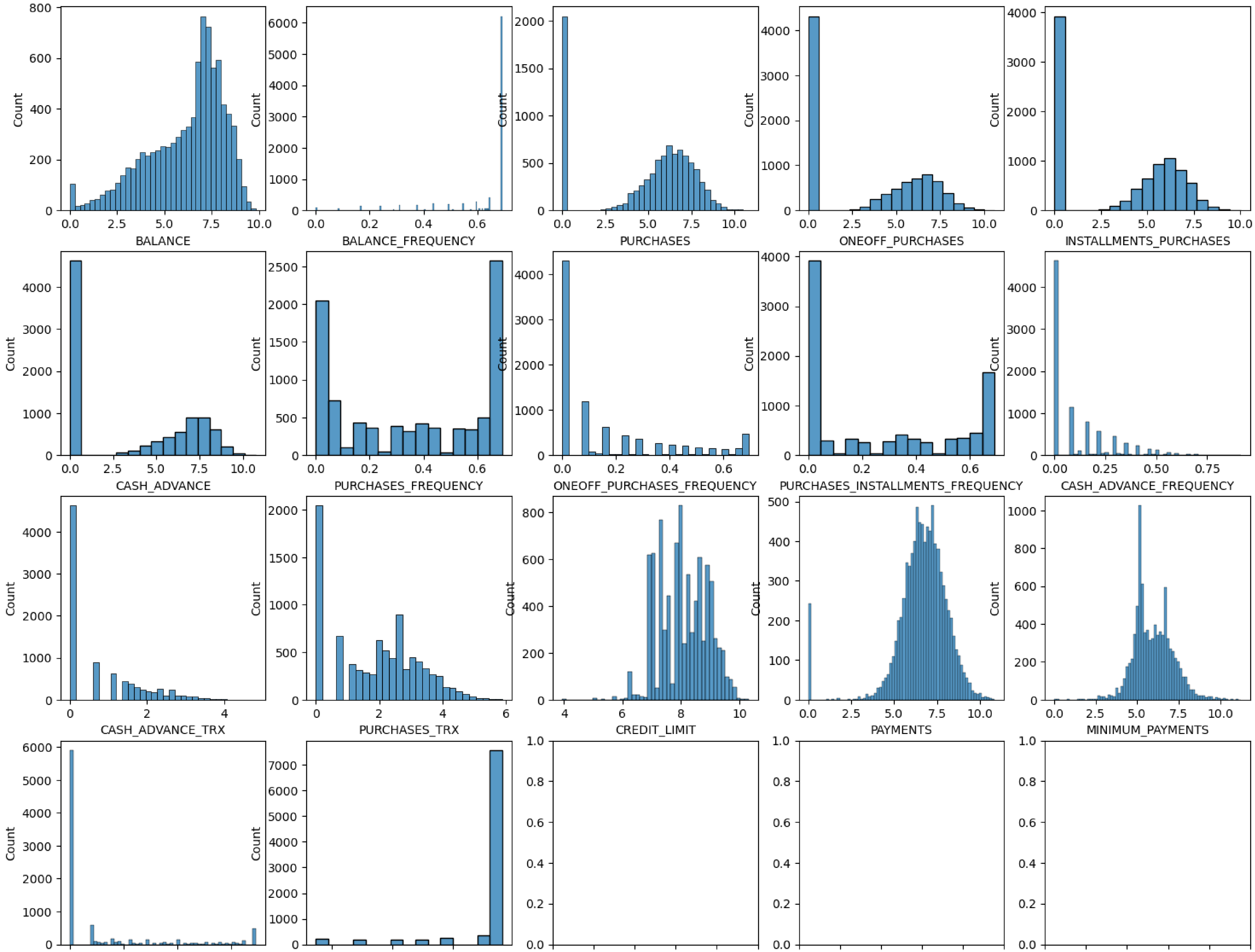
Out[ ]:
```
BALANCE                             0
BALANCE_FREQUENCY                   0
PURCHASES                           0
ONEOFF_PURCHASES                    0
INSTALLMENTS_PURCHASES              0
CASH_ADVANCE                        0
PURCHASES_FREQUENCY                 0
ONEOFF_PURCHASES_FREQUENCY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

- Transformación logarítmica para todas las columnas del dataframe

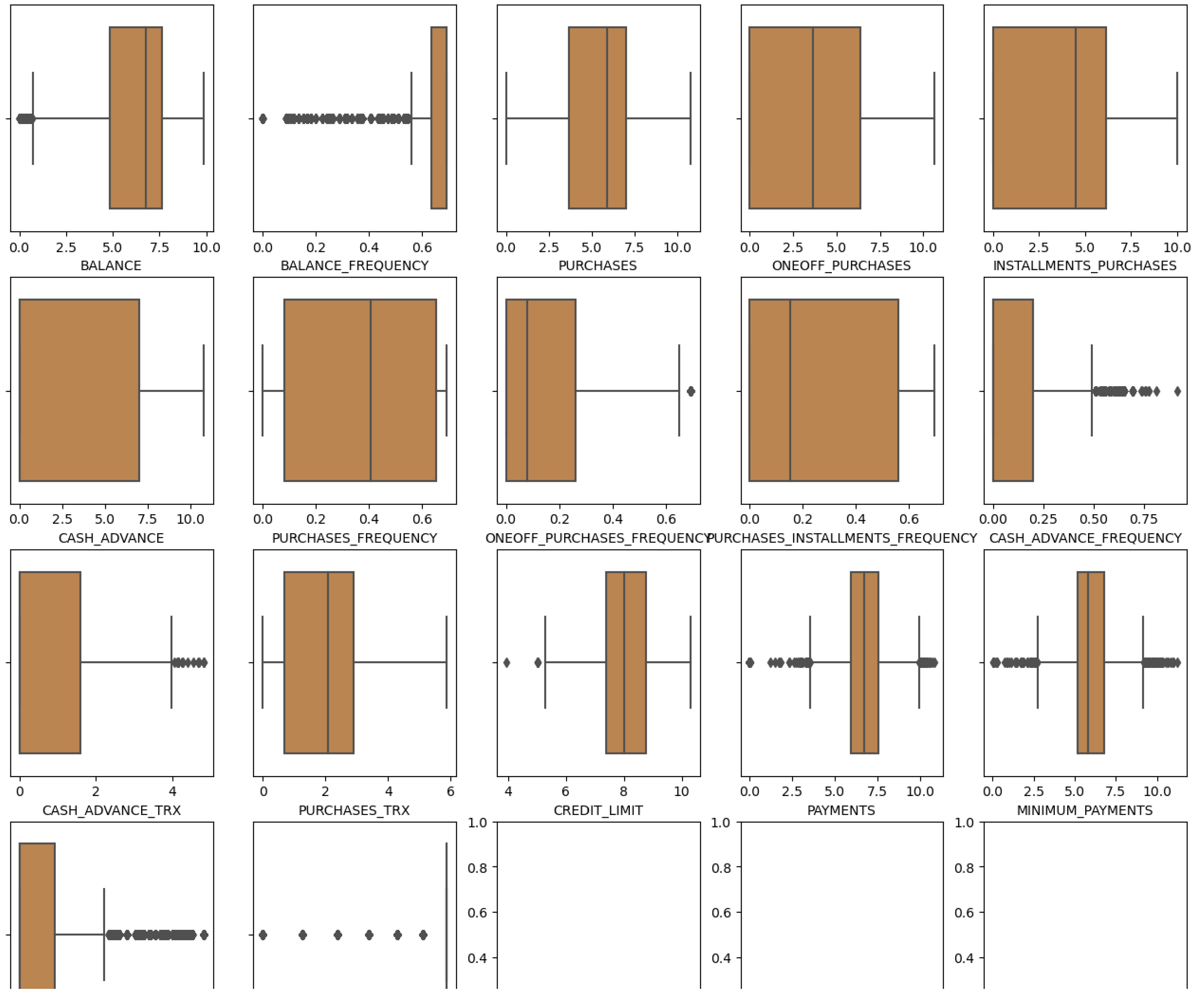
In [ ]:
```
for col in cols:
    df2[col]=np.log(df2[col]+1)

# Ahora, Todas las columnas de df2 estan en logaritmo
```

In [ ]:
```
fig, axes = plt.subplots(nrows=5, ncols=5, figsize=(18,18))
for i,column in enumerate(cols):
    sns.histplot(df2[column],ax=axes[i//5,i%5],kde=False)
```


BALANCE
BALANCE_FREQUENCY
PURCHASES
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS
Count

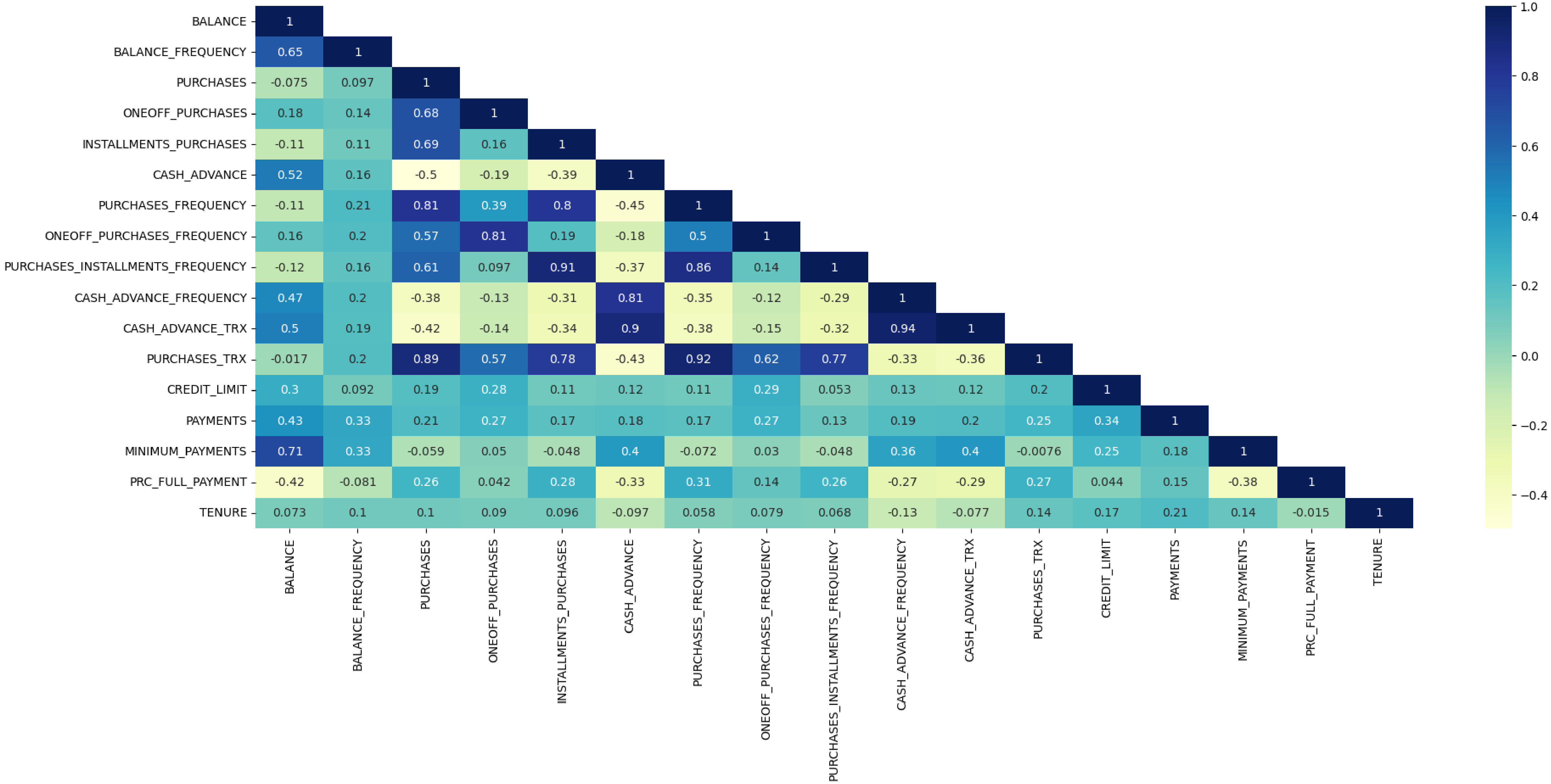
In [ ]:
```
fig,axes= plt.subplots(nrows=5,ncols=5, figsize=(16,18))
for i,column in enumerate (cols):
    sns.boxplot(x=df2[column], color='peru',ax=axes[i//5,i%5])
```


BALANCE
BALANCE_FREQUENCY
PURCHASES
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS

Insights:

- Aún haciendo la transformación logarítmica, hay variables con outliers y sesgos considerables.

In [ ]:
```
# Grafica de correlacion
# Dado que se hicieron varias transformaciones conviene revisar de nuevo la correlacion.
plt.figure(figsize=(22,8))
corr_df = corr = df2.corr(method='pearson')
df_lt= corr_df.where(np.tril(np.ones(corr_df.shape)).astype(bool))
hmap=sns.heatmap(df_lt, cmap='YlGnBu',annot=True)
```

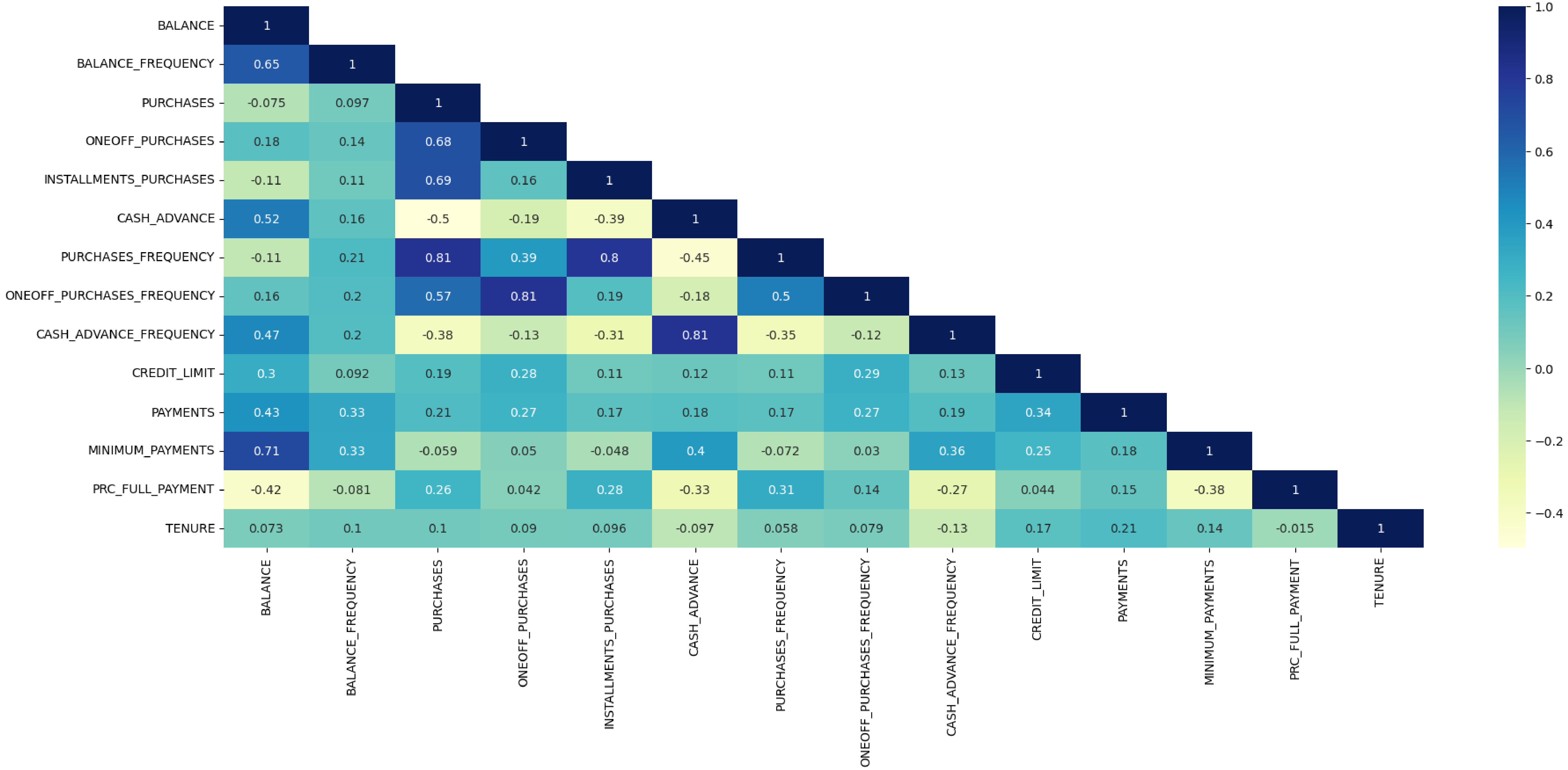


Insights:

- Dado que existe multioclinearidad entre varias variables se procede a eliminar algunas de ellas.

In [ ]:
```
drop_cols=['PURCHASES_TRX','CASH_ADVANCE_TRX','PURCHASES_INSTALLMENTS_FREQUENCY']
df2.drop(columns=drop_cols,inplace=True)
```

In [ ]:
```
# Grafica de correlacion
# Dado que se hicieron varias transformaciones conviene revisar de nuevo la correlacion.
plt.figure(figsize=(22,8))
corr_df = corr = df2.corr(method='pearson')
df_lt= corr_df.where(np.tril(np.ones(corr_df.shape)).astype(bool))
hmap=sns.heatmap(df_lt, cmap='YlGnBu',annot=True)
```



- Escalamiento

In [ ]:
```
# Escalamiento
# Se aplica el escalamiento tipo z-score antes del clustering
from scipy.stats import zscore
df2 = df2.apply(zscore)
```

In [ ]:
```
# Se validan los nuevos rangos, escalados
df2.describe().T
```

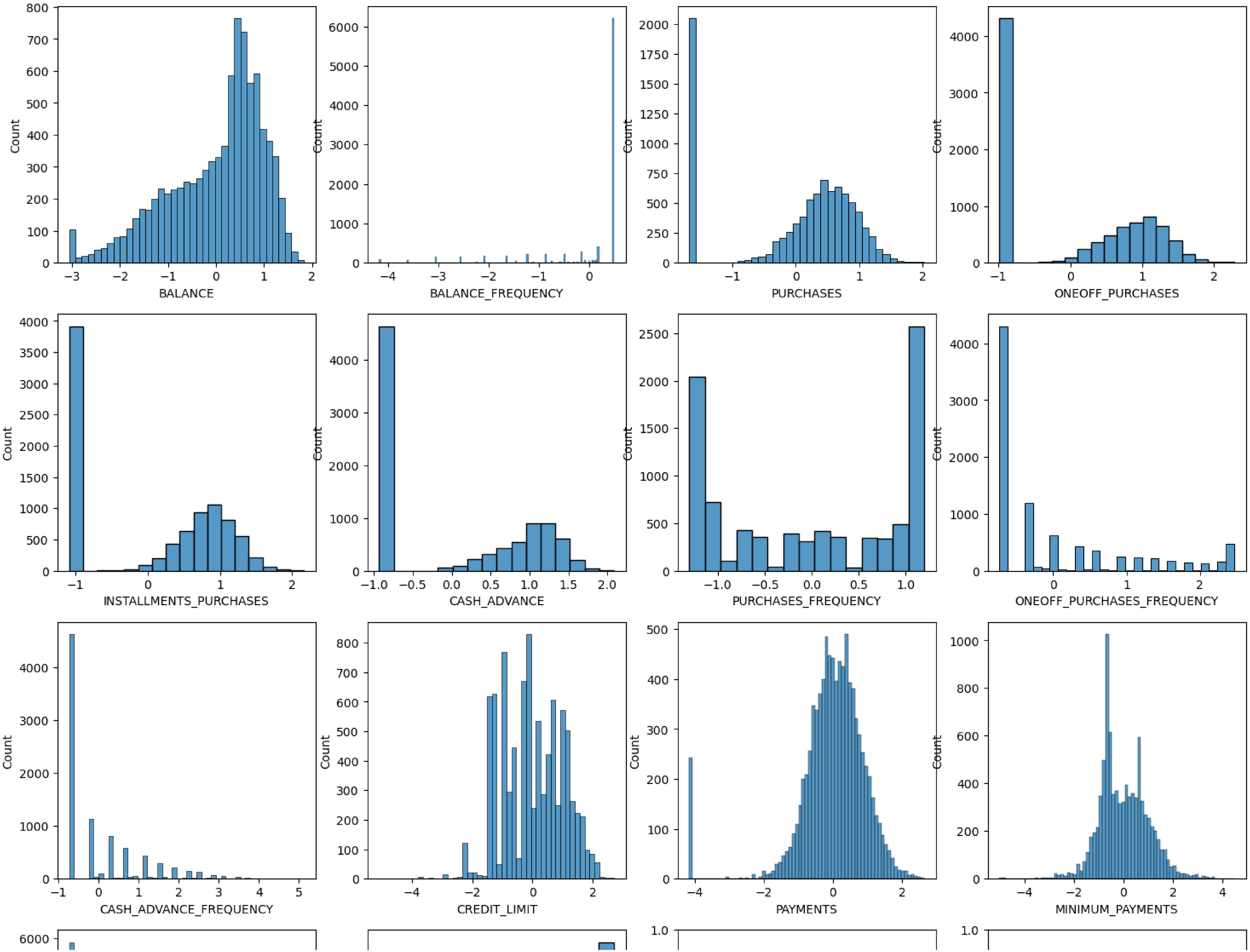
Out[ ]:

	count	mean	std	min	25%	50%	75%	max
BALANCE	8950.0	-1.714829e-16	1.000056	-3.060633	-0.645563	0.303937	0.728427	1.834341
BALANCE_FREQUENCY	8950.0	7.907269e-16	1.000056	-4.172328	0.108052	0.492701	0.492701	0.492701
PURCHASES	8950.0	6.986342e-17	1.000056	-1.679855	-0.409715	0.340373	0.724613	2.023087
ONEOFF_PURCHASES	8950.0	0.000000e+00	1.000056	-0.987090	-0.987090	0.141485	0.972218	2.283062
INSTALLMENTS_PURCHASES	8950.0	1.034058e-16	1.000056	-1.087454	-1.087454	0.372196	0.908121	2.163264
CASH_ADVANCE	8950.0	-8.097805e-17	1.000056	-0.930733	-0.930733	-0.930733	1.036809	2.086805
PURCHASES_FREQUENCY	8950.0	1.341695e-16	1.000056	-1.302784	-1.014248	0.159389	1.043403	1.196817
ONEOFF_PURCHASES_FREQUENCY	8950.0	-6.351220e-17	1.000056	-0.732464	-0.732464	-0.363169	0.478478	2.466756
CASH_ADVANCE_FREQUENCY	8950.0	3.969512e-17	1.000056	-0.724345	-0.724345	-0.724345	0.556078	5.122777
CREDIT_LIMIT	8950.0	-1.282946e-15	1.000056	-5.079426	-0.874201	-0.107577	0.835591	2.701494
PAYMENTS	8950.0	4.255317e-16	1.000056	-4.161996	-0.422938	0.081643	0.581898	2.644753
MINIMUM_PAYMENTS	8950.0	1.270244e-16	1.000056	-5.029405	-0.682389	-0.112429	0.687847	4.486544
PRC_FULL_PAYMENT	8950.0	-3.175610e-17	1.000056	-0.556360	-0.556360	-0.556360	0.074856	2.719297
TENURE	8950.0	1.422673e-15	1.000056	-4.401425	0.347262	0.347262	0.347262	0.347262

In [ ]:

```
# Grafica exploratoria de todas las columnas
col_df2=df2.columns.to_list()

fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(18,18))
for i,column in enumerate(col_df2):
    sns.histplot(df2[column],ax=axes[i//4,i%4],kde=False)
```


Count
BALANCE
BALANCE_FREQUENCY
PURCHASES
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
CASH_ADVANCE_FREQUENCY
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS

Recapitulando las transformaciones hechas:

- Se eliminaron 4 columnas:
  - 'CUST_ID, PURCHASES_TRX', 'CASH_ADVANCE_TRX', 'PURCHASES_INSTALLMENTS_FREQUENCY'
- A las columnas con valores nulos ('CREDIT_LIMIT', 'MINIMUM_PAYMENTS') se les imputó la media correspondiente a esa misma columna.
- A todas las columnas del dataframe se les realizó una transformación logarítmica.
- Finalmente, se hizo un escalonamiento a todos los datos.

Modelo de Clustering K-Means

In [ ]:

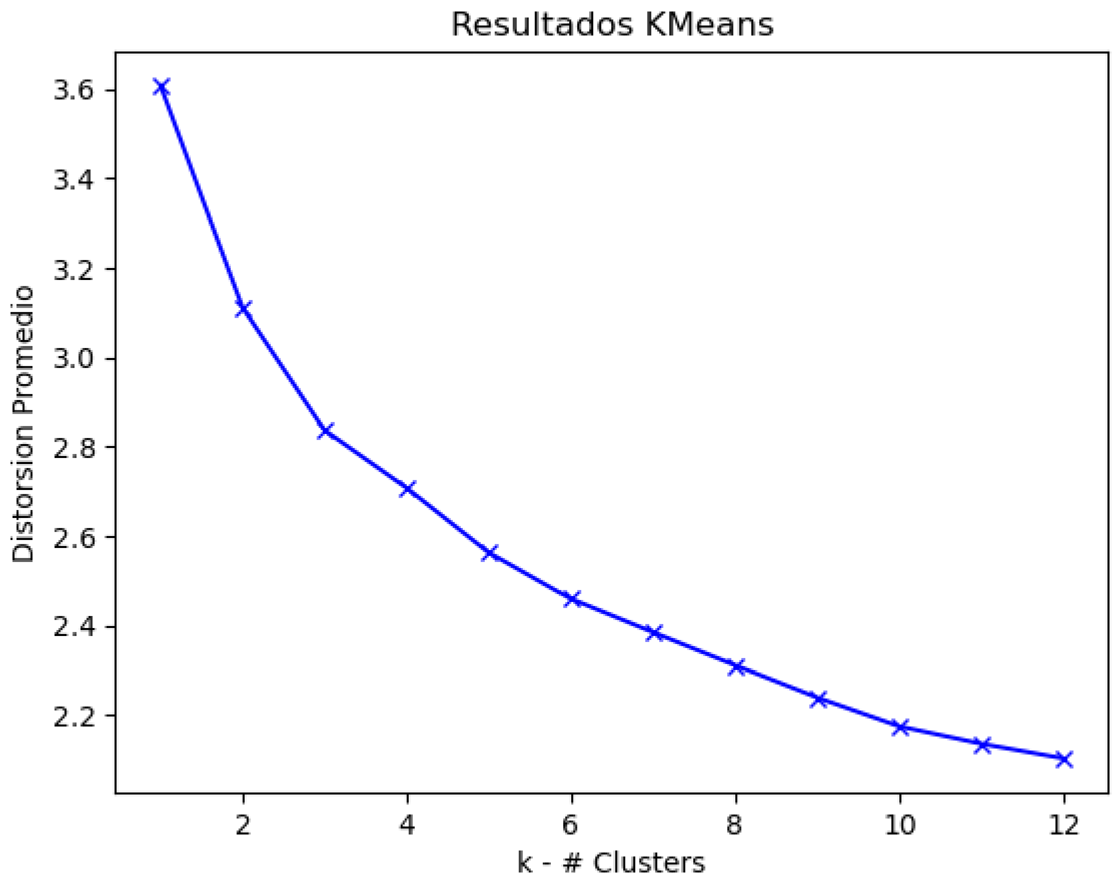
```
# Buscando el optimo numero de Clusters
from scipy.spatial.distance import cdist
clusters=range(1,13)
meanDistortion=[]

for k in clusters:
    model=KMeans(n_clusters=k)
    model.fit(df2)
    prediction=model.predict(df2)
    # Generates the average distortion calculation for each one of the cluster points
    # Compares each data point with the cluster centers and obtains the minimum and divides it for mydata
    meanDistortion.append(sum(np.min(cdist(df2,model.cluster_centers_,'euclidean'), axis=1))/df2.shape[0])

#Plots the scree graphic
# Distortion decreases as the number ofclusters increase, until the number of clusters = number of points
plt.plot(clusters, meanDistortion, 'bx-')
plt.xlabel('k - # Clusters')
plt.ylabel('Distorsion Promedio')
plt.title ('Resultados KMeans')
```

Out[ ]:

```
Text(0.5, 1.0, 'Resultados KMeans')
```



In [ ]:
```
prediction
```

Out[ ]:
```
array([1, 5, 9, ..., 7, 7, 7])
```

Main Insights:

- Por inspeccion visual, un numero de 3 clusters parece el indicado, y donde se forma el'codo' en el grafico.
- Se hizo un analisis entre 1 a 12 clusters

In [ ]:
```
# Se genera el numero de clusters = 3
# El parametro n_init significa que se probara con 15 diferentes inicializaciones y se tomaran las mejores
kmeans = KMeans(n_clusters=3,n_init=15,random_state=1)
kmeans.fit(df2)
```

Out[ ]:
```
KMeans
KMeans(n_clusters=3, n_init=15, random_state=1)
```

In [ ]:
```
# Se genera un dataframe para los labels de los clusters y se los convierte en categorias
# Asi, cada registro tiene un cluster asociado
df_labels=pd.DataFrame(kmeans.labels_,columns=list(['labels']))
df_labels['labels']=df_labels['labels'].astype('category')
```

In [ ]:
```
# Hace un union del dataframe de las etiquetas con el de datos
df_labeled = df3.join(df_labels)
```

In [ ]:
```
df_labeled.head()
```

Out[ ]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY
0	C10001	40.9007	0.8182	95.40	0.00	95.4	0.0000	0.1667	0.0000	0.0833
1	C10002	3202.4674	0.9091	0.00	0.00	0.0	6442.9455	0.0000	0.0000	0.0000
2	C10003	2495.1489	1.0000	773.17	773.17	0.0	0.0000	1.0000	1.0000	0.0000
3	C10004	1666.6705	0.6364	1499.00	1499.00	0.0	205.7880	0.0833	0.0833	0.0000
4	C10005	817.7143	1.0000	16.00	16.00	0.0	0.0000	0.0833	0.0833	0.0000

In [ ]:

```
df_labeled.columns
```

Out[ ]:

```
Index(['CUST_ID', 'BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES',
       'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE',
       'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY',
       'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FREQUENCY',
       'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'CREDIT_LIMIT', 'PAYMENTS',
       'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT', 'TENURE', 'labels'],
      dtype='object')
```

In [ ]:

```
# Division de los clusters
# Numero de registros con cada uno de los clusters
df_labeled['labels'].value_counts()
```

Out[ ]:

```
1    3407
0    2842
2    2701
Name: labels, dtype: int64
```

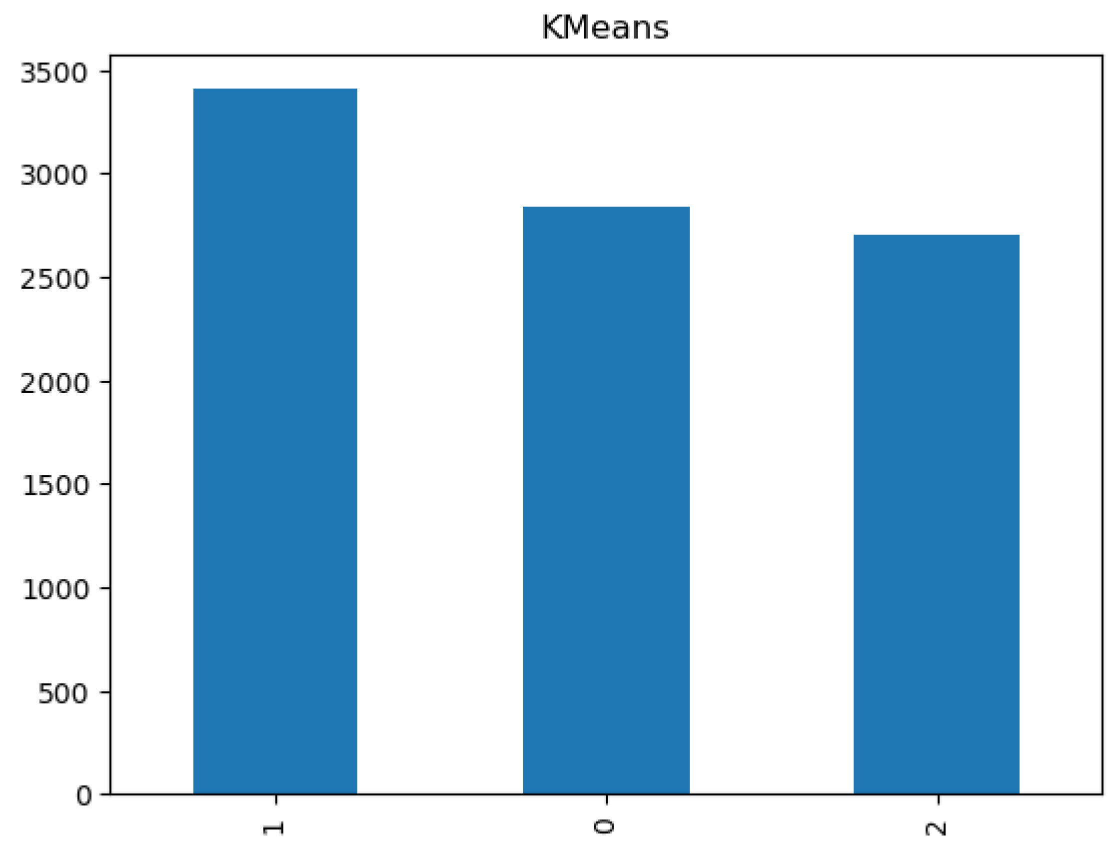
Visualización de los resultados del modelo de clustering: bar plots y clases – identificación y perfilamiento de clases

In [ ]:

```
# Grafico de clusters por registro
df_labeled['labels'].value_counts().plot(kind='bar').set_title('KMeans')
```

Out[ ]:

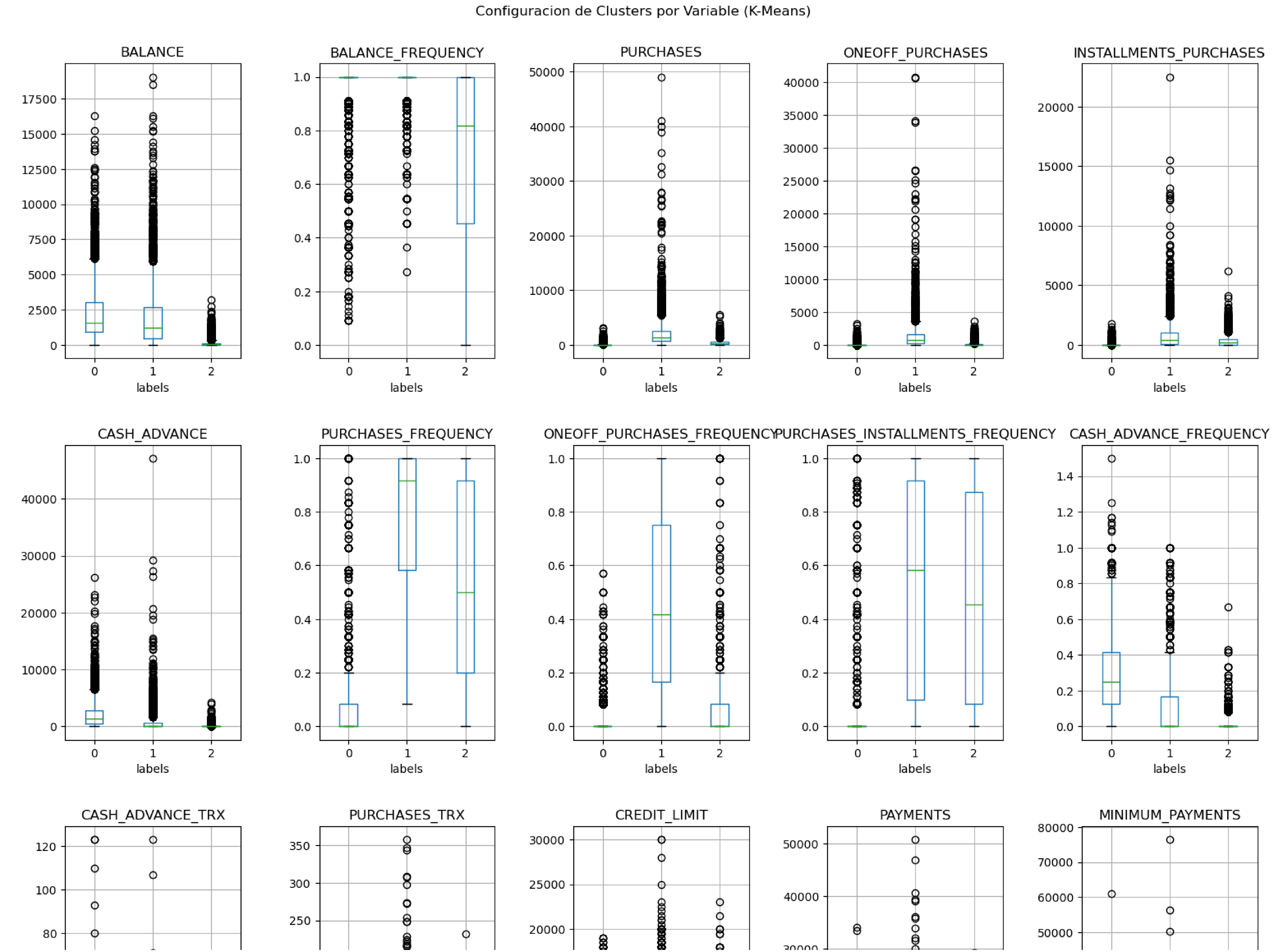
```
Text(0.5, 1.0, 'KMeans')
```



In [ ]:
```
cols_final = df_labeled.columns.to_list()
cols_final.remove('labels')
cols_final.remove('CUST_ID')
print(cols_final)
print(len(cols_final))
```

```
['BALANCE', 'BALANCE_FREQUENCY', 'PURCHASES', 'ONEOFF_PURCHASES', 'INSTALLMENTS_PURCHASES', 'CASH_ADVANCE', 'PURCHASES_FREQUENCY', 'ONEOFF_PURCHASES_FREQUENCY', 'PURCHASES_INSTALLMENTS_FREQUENC
Y', 'CASH_ADVANCE_FREQUENCY', 'CASH_ADVANCE_TRX', 'PURCHASES_TRX', 'CREDIT_LIMIT', 'PAYMENTS', 'MINIMUM_PAYMENTS', 'PRC_FULL_PAYMENT', 'TENURE']
17
```

In [ ]:
```
fig, ax = plt.subplots(nrows=5,ncols=5,figsize=(16,24))
for i,col in enumerate(cols_final):
    df_labeled.boxplot(col,'labels',ax=ax[i//5,i%5])
    #sns.boxplot(df_labeled(col),'labels',ax=ax[i//5,i%5])
fig.suptitle('Configuracion de Clusters por Variable (K-Means)')
fig.tight_layout(pad=3.0)
```


Configuracion de Clusters por Variable (K-Means)
BALANCE
BALANCE_FREQUENCY
PURCHASES
ONEOFF_PURCHASES
INSTALLMENTS_PURCHASES
CASH_ADVANCE
PURCHASES_FREQUENCY
ONEOFF_PURCHASES_FREQUENCY
PURCHASES_INSTALLMENTS_FREQUENCY
CASH_ADVANCE_FREQUENCY
CASH_ADVANCE_TRX
PURCHASES_TRX
CREDIT_LIMIT
PAYMENTS
MINIMUM_PAYMENTS
labels

Insights

- A partir de las gráfica anterior se puede extraer a grandes líneas el perfil de los 3 clusters:

Cluster 0:

- Con respecto al balance (dinero en el banco) tienen la media más alta con un sesgo positivo. Lo cual podría indicar que es un grupo con un nivel socioeconomico medio-alto.
- Es el grupo con la menor cantidad de compras en promedio y la menor desviación estandar. Esto podría indicar que aunque si tienen un alto nivel de ingresos no son reconocidos como compradores compulsivos.
- Además, en el momento en el que compran, prefieren pagar en una sola cuota en vez de generar un credito con varias cuotas futuras. Personas de una cierta edad?

En base al perfil: Dadas las caracteristicas dichas anteriormente, A este grupo se les podrían plantear estratégias de inversión a medio y largo plazo como estrategia de marketing. Ya que son individuos más conservadores que estan más inclinados por ahorrar en vez de gastar.

Cluster 1:

- Con respecto al balance (dinero en el banco) tienen la segunda media más alta muy cerca al cluster 0 con un sesgo también positivo. Lo cual podría indicar que es un grupo con un nivel socioeconomico medio-alto.
- En comparación con el cluster 0, En promedio los individuos de este grupo realizan más compras con un sesgo decidimamente positivo. Esto indica que hay una inclinación para ser catalogados como compradores compulsivos.
- Además, en el momento en el que compran, prefieren pagar a varias cuotas en vez de pagar todo de una vez. Personas jovenes?

En base al perfil: Dadas las caracteristicas dichas anteriormente, A este grupo se les podrían plantear estratégias de credito ("tarjetas de cerdito", "beneficios por realizar un #top de compras") como estrategia de marketing. Ya que son individuos que son más propensos a gastar.

Cluster 2:

- Con respecto al balance (dinero en el banco) tienen la media más baja con un sesgo nínimo positivo. Lo cual podría indicar que es un grupo con un nivel socioeconomico medio-bajo.

- Al no tener un alto poder adquisitivo no es notoría la cantidad de compras hechas. Sin embargo, realizan más compras de las personas del cluster 0 y prefieren pagar a cuotas.

En base al perfil: No es un cluster muy atractivo en términos de marketing. Tendría que hacerse un análisis al interno del cluster para entender mejor la distribución del mismo. Estratégias de credito serían bien recibidas por las personas de este cluster. Sin embargo, al no tener un alto nivel de poder adquisitivo, dichos creditos no pueden ser muy elevados y no tendrán una incidencia como lo serán para los del cluster #1

- Si hubiesemos tenido la variable edad nos habría ayudado a perfilar mejor cada cluster.
- Para los dos primeros clusters (0 y 1) sería interesante realizar un EDA para cada uno. Ya que seguramente las propuestas de marketing hechas para el cluster 1 serían bien recibidas para algunas personas del cluster 0 y viceversa.

In [ ]:
```
df_labeled.groupby('labels').mean()
```

Out[ ]:

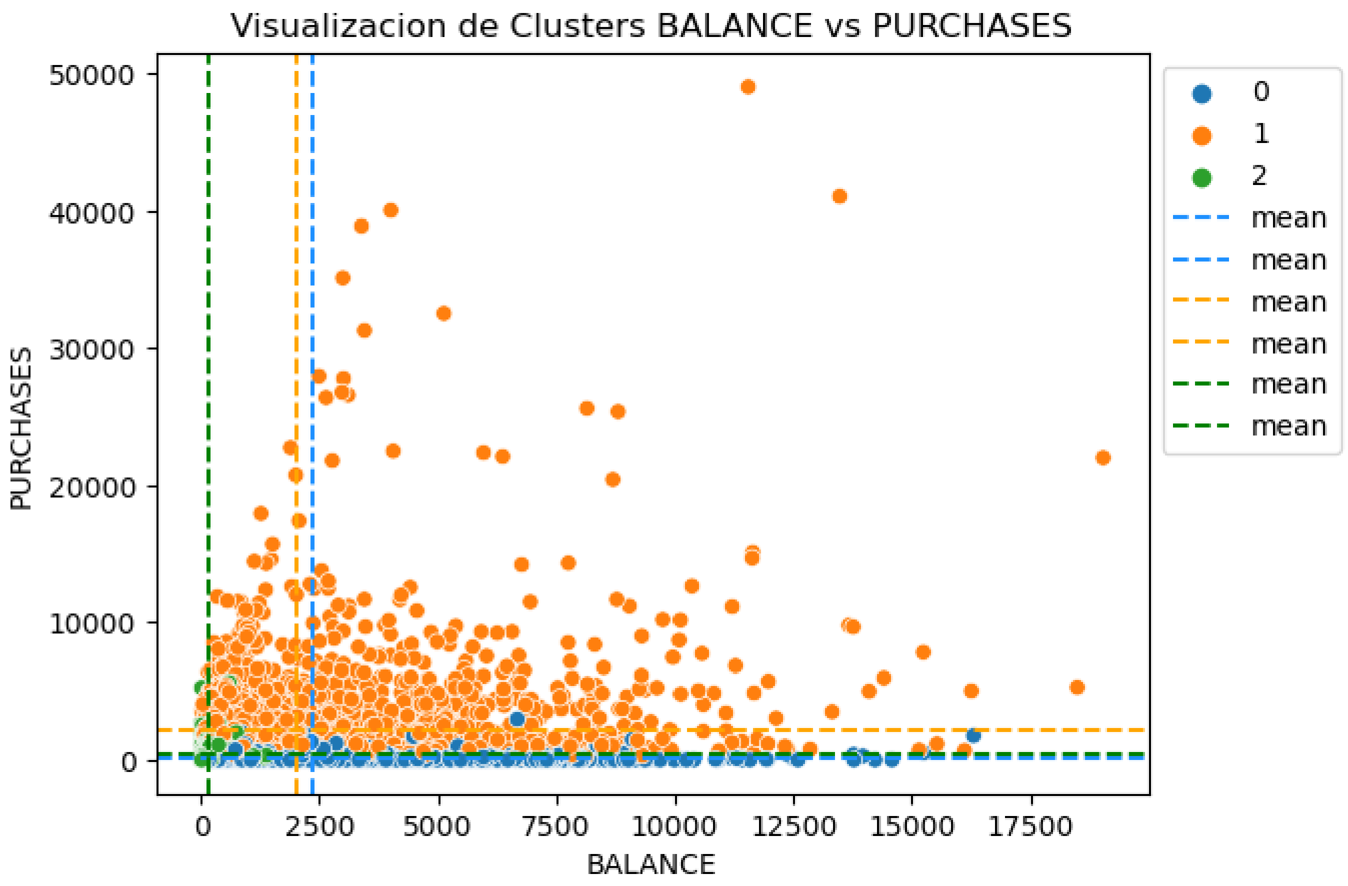
labels	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CAS
0	2342.916858	0.926562	83.648346	57.947699	25.848593	2114.175998	0.080324	0.031781	0.044305	
1	2020.430984	0.975839	2197.835609	1408.862257	789.129117	773.717328	0.780123	0.457722	0.539898	
2	170.259768	0.701077	463.875876	125.003110	339.515298	43.077874	0.556263	0.060049	0.479962	

In [ ]:
```
df_labeled.groupby('labels')['PURCHASES'].mean()
```

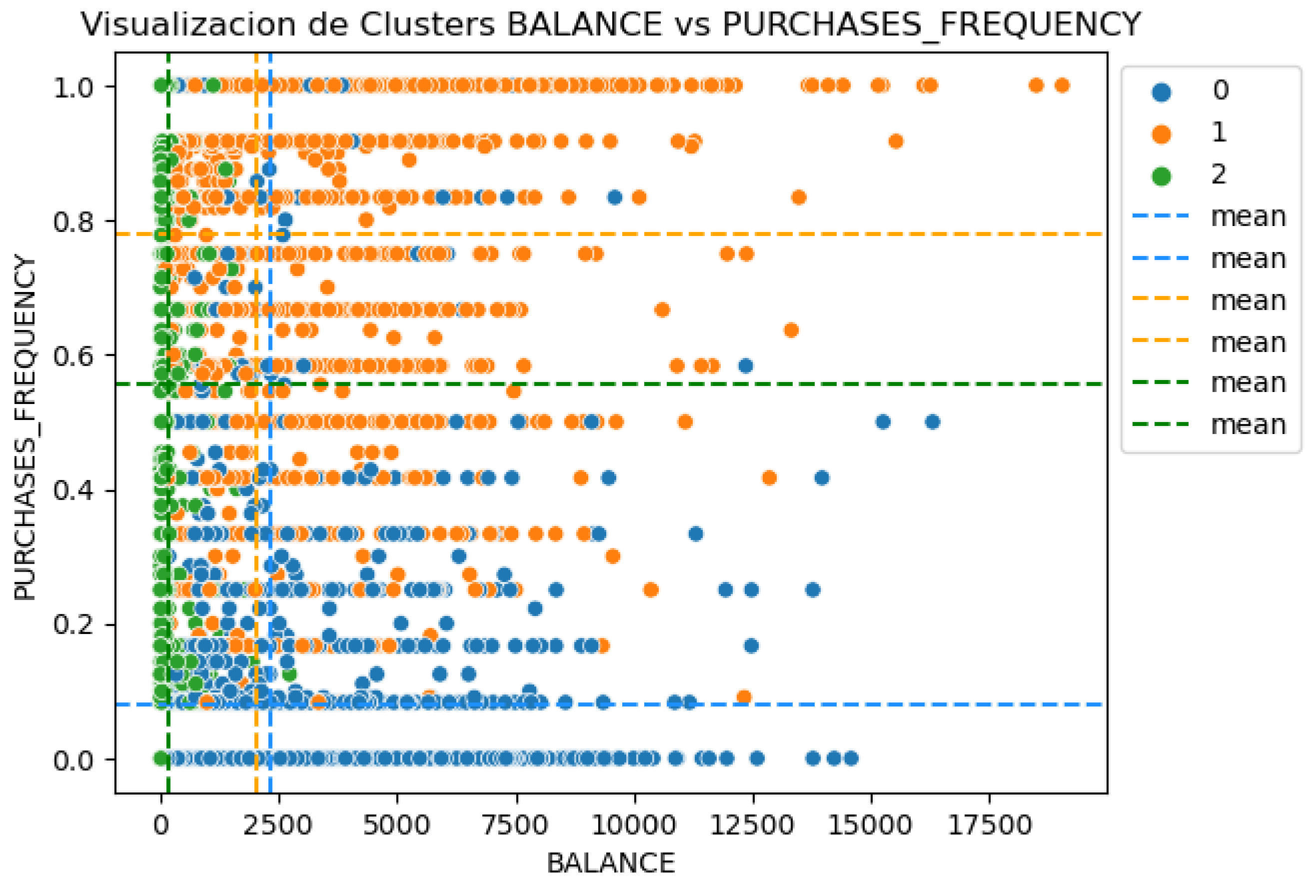
Out[ ]:
```
labels
0      83.648346
1    2197.835609
2     463.875876
Name: PURCHASES, dtype: float64
```

A continuación se presentan algunos gráficos que corroboran lo expresado anteriormente en los perfiles.

In [ ]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
scatter = sns.scatterplot(x=df_labeled['BALANCE'],y=df_labeled['PURCHASES'],hue=df_labeled['labels'])#, fit_reg=False)
ax.set_title('Visualizacion de Clusters BALANCE vs PURCHASES')
colors=['dodgerblue','orange','green']
for i in range(0,3):
    ax.axhline(y=df_labeled.groupby('labels')['PURCHASES'].mean()[i],color=colors[i], ls='--', label='mean')
    ax.axvline(x=df_labeled.groupby('labels')['BALANCE'].mean()[i],color=colors[i], ls='--', label='mean')
plt.legend(bbox_to_anchor = (1, 1), loc = 'upper left')
plt.show()
```

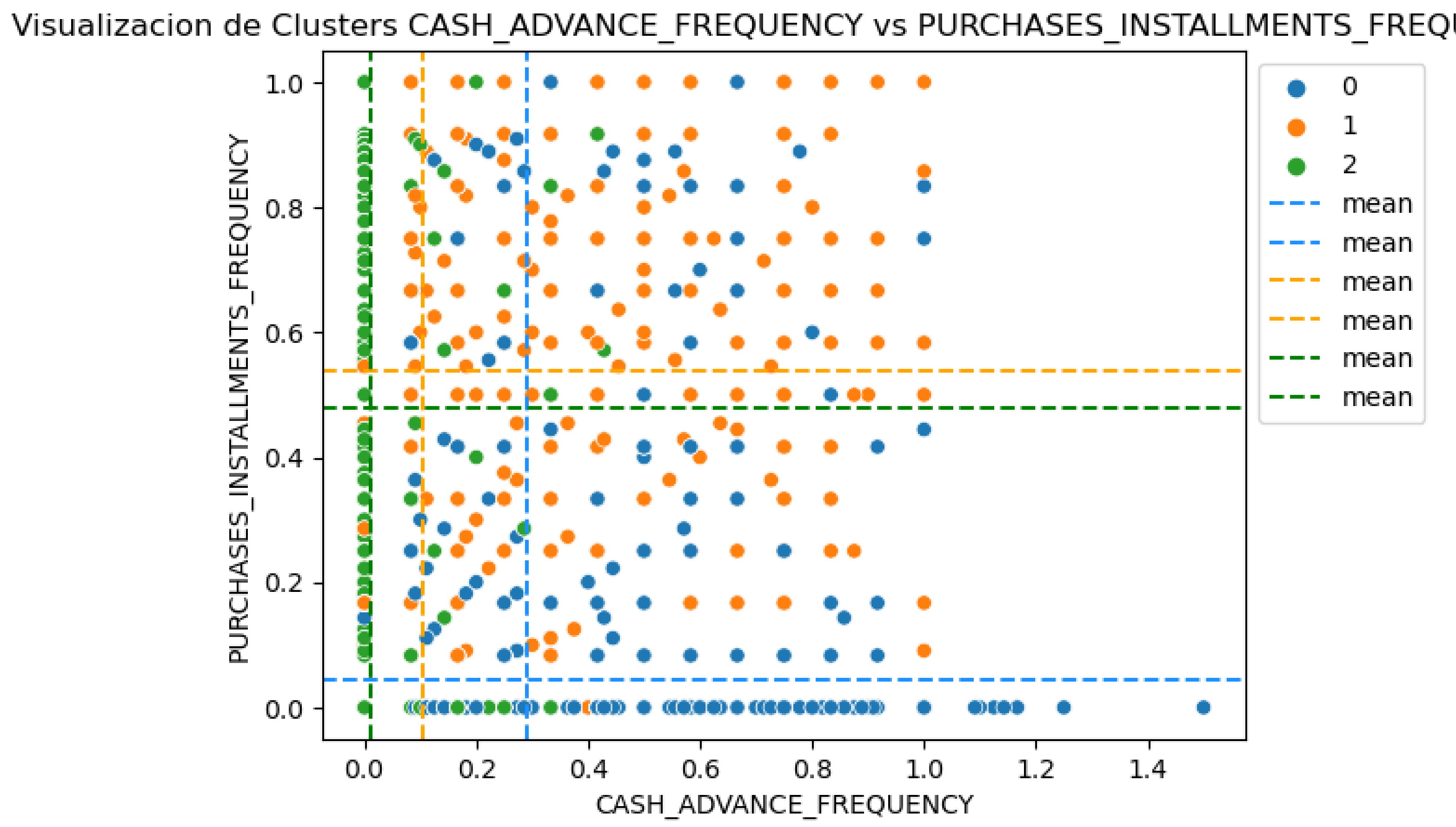


In [ ]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
scatter = sns.scatterplot(x=df_labeled['BALANCE'],y=df_labeled['PURCHASES_FREQUENCY'],hue=df_labeled['labels'])#, fit_reg=False)
ax.set_title('Visualizacion de Clusters BALANCE vs PURCHASES_FREQUENCY')
colors=['dodgerblue','orange','green']
for i in range(0,3):
    ax.axhline(y=df_labeled.groupby('labels')['PURCHASES_FREQUENCY'].mean()[i],color=colors[i], ls='--', label='mean')
    ax.axvline(x=df_labeled.groupby('labels')['BALANCE'].mean()[i],color=colors[i], ls='--', label='mean')
plt.legend(bbox_to_anchor = (1, 1), loc = 'upper left')
plt.show()
```



In [ ]:

```
fig = plt.figure()
ax = fig.add_subplot(111)
scatter = sns.scatterplot(x=df_labeled['CASH_ADVANCE_FREQUENCY'],y=df_labeled['PURCHASES_INSTALLMENTS_FREQUENCY'],hue=df_labeled['labels'])#, fit_reg=False)
ax.set_title('Visualizacion de Clusters CASH_ADVANCE_FREQUENCY vs PURCHASES_INSTALLMENTS_FREQUENCY')
colors=['dodgerblue','orange','green']
for i in range(0,3):
    ax.axhline(y=df_labeled.groupby('labels')['PURCHASES_INSTALLMENTS_FREQUENCY'].mean()[i],color=colors[i], ls='--', label='mean')
    ax.axvline(x=df_labeled.groupby('labels')['CASH_ADVANCE_FREQUENCY'].mean()[i],color=colors[i], ls='--', label='mean')
plt.legend(bbox_to_anchor = (1, 1), loc = 'upper left')
plt.show()
```



Dendogramas y validación de la calidad del fit

In [ ]:

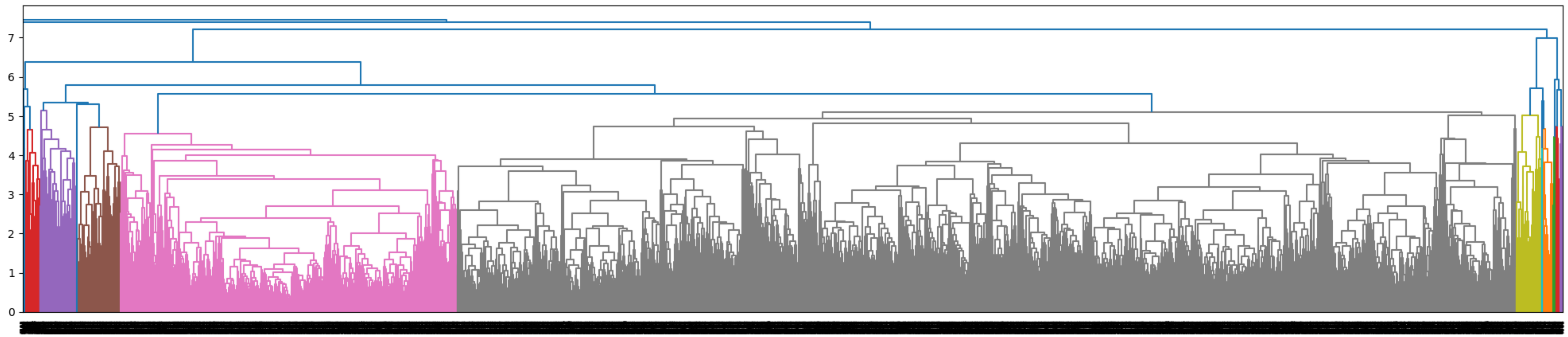
```
# Se genera la matriz de enlaces
from scipy.cluster.hierarchy import dendrogram, linkage
Z = linkage(df2,'average', metric='euclidean')
Z.shape
```

Out[ ]:

```
(8949, 4)
```

In [ ]:

```
# Generate Dendrogram
plt.figure(figsize=(25,5))
dendrogram(Z)
plt.show()
```



In [ ]:
```
# Se calculo el Coeficiente Cofenetico para validar la calidad del fit del dendograma con
# los pares de datos in ordenar
# El maximo del coeficiente es 1
c, coph_dists=cophenet(Z,pdist(df2))
print('Cophenetic Coefficient:' , format(c,'.4f'))
```

```
Cophenetic Coefficient: 0.6734
```

In [ ]:
```
# Genera un dataframe para el resultado
sil_df = pd.DataFrame({},columns=['model','n_clusters','score'], index=None)
```

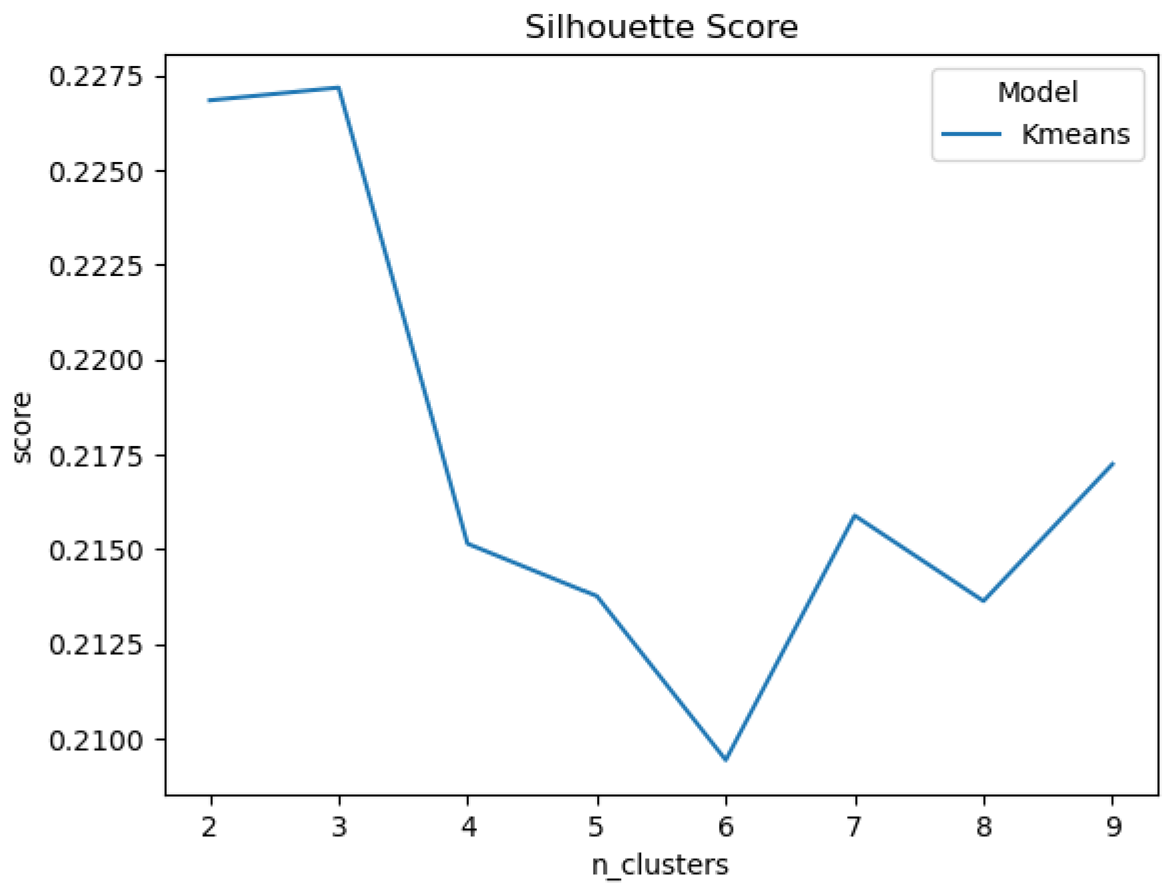
In [ ]:
```
from sklearn.metrics import silhouette_score

# Resultado de KMeans de 2 a 10 clusters
clusters=range(2,10)
for n_clusters in clusters:
    clusterer = KMeans(n_clusters=n_clusters, random_state=1)
    preds = clusterer.fit_predict(df2)
    centers= clusterer.cluster_centers_

    score=silhouette_score(df2,clusterer.labels_, metric='euclidean')
    # Adds result to sil_df results table
    sil_df = sil_df.append({'Model':'Kmeans','n_clusters':n_clusters,'score':score}, ignore_index=True)
```

In [ ]:
```
sns.lineplot(data=sil_df, x='n_clusters', y='score', hue='Model', style='Model', ci=None). set_title('Silhouette Score')
```

Out[ ]:
```
Text(0.5, 1.0, 'Silhouette Score')
```



Main insights:

- Se confirma con el Silhouette Score que con k=3 se tiene el mejor desempeno