Actividad Módulo 29: Introducción al Clustering

- Importación de un modelo con datos de tarjetas de crédito (cc general.csv)
- EDA completo siguiento los modelos anteriores
- Feature engineering - estandarización, eliminación, nuevas columnas
- Uso del algorítmo K-Means: elbow plot y análisis
- Visualización de los resultados del modelo de clustering: bar plots y clases - identificación y perfilamiento de clases
- Dendogramas y validación de la calidad del fit
- Silhouette score
- Visualización simple de los resultados del modelo de clustering
- Interpretación final de resultados de los clusters generados por el modelo

CSV CC GENERAL.csv

- 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

Imporación del Módelo

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Importación del modelo
df = pd.read_csv("CC GENERAL.csv")
df.sample(5)
```

Out[ ]:

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMEI
2458	C12532	7.153512	0.300000	104.95	0.00	
7996	C18212	187.536914	0.818182	530.04	0.00	
4225	C14343	226.785380	1.000000	210.64	210.64	
5020	C15161	62.996845	1.000000	779.70	290.00	
6343	C16518	10.259525	0.181818	119.80	119.80	

EDA

In [ ]:
```
df.shape
```

Out[ ]: (8950, 18)

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.describe().T
```

Out[ ]:

	count	mean	std	min	25%
BALANCE	8950.0	1564.474828	2081.531879	0.000000	128.281915
BALANCE_FREQUENCY	8950.0	0.877271	0.236904	0.000000	0.888889
PURCHASES	8950.0	1003.204834	2136.634782	0.000000	39.635000
ONEOFF_PURCHASES	8950.0	592.437371	1659.887917	0.000000	0.000000
INSTALLMENTS_PURCHASES	8950.0	411.067645	904.338115	0.000000	0.000000
CASH_ADVANCE	8950.0	978.871112	2097.163877	0.000000	0.000000
PURCHASES_FREQUENCY	8950.0	0.490351	0.401371	0.000000	0.083333
ONEOFF_PURCHASES_FREQUENCY	8950.0	0.202458	0.298336	0.000000	0.000000
PURCHASES_INSTALLMENTS_FREQUENCY	8950.0	0.364437	0.397448	0.000000	0.000000
CASH_ADVANCE_FREQUENCY	8950.0	0.135144	0.200121	0.000000	0.000000
CASH_ADVANCE_TRX	8950.0	3.248827	6.824647	0.000000	0.000000
PURCHASES_TRX	8950.0	14.709832	24.857649	0.000000	1.000000
CREDIT_LIMIT	8949.0	4494.449450	3638.815725	50.000000	1600.000000
PAYMENTS	8950.0	1733.143852	2895.063757	0.000000	383.276166
MINIMUM_PAYMENTS	8637.0	864.206542	2372.446607	0.019163	169.123707
PRC_FULL_PAYMENT	8950.0	0.153715	0.292499	0.000000	0.000000
TENURE	8950.0	11.517318	1.338331	6.000000	12.000000

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
```

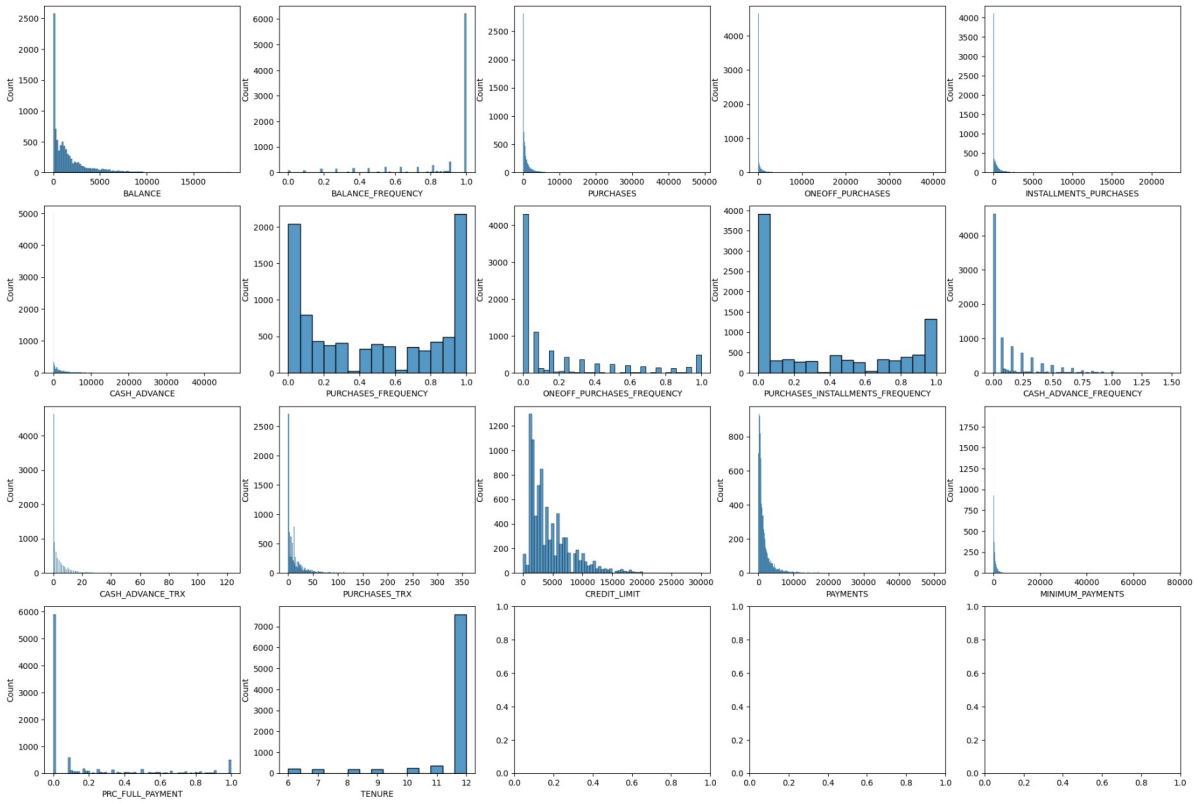
Insights

- Hay variables que necesitarán ser normalizadas puesto que algunas van de 0 a 76K mientras que otras van de 0 a 1
- Hay 313 registros nulos en MINIMUM_PAYMENTS y 1 registro nulo en CREDIT_LIMIT
- Los registros faltantes serán imputados con el promedio de la respectiva variable

In [ ]:
```
# Imputación de datos
df['MINIMUM_PAYMENTS'].fillna(value=df['MINIMUM_PAYMENTS'].mean(), inplace=True)
df['CREDIT_LIMIT'].fillna(value=df['CREDIT_LIMIT'].mean(), inplace=True)
df.isna().sum().sum()
```

Out[ ]: 0

In [ ]:
```
# Análisis univariado
fig, ax = plt.subplots(nrows=4, ncols=5, figsize=(25,17))
for i, column in enumerate(df.iloc[:, 1:].columns):
    sns.histplot(df[column], ax=ax[i//5, i%5], kde=False)
```



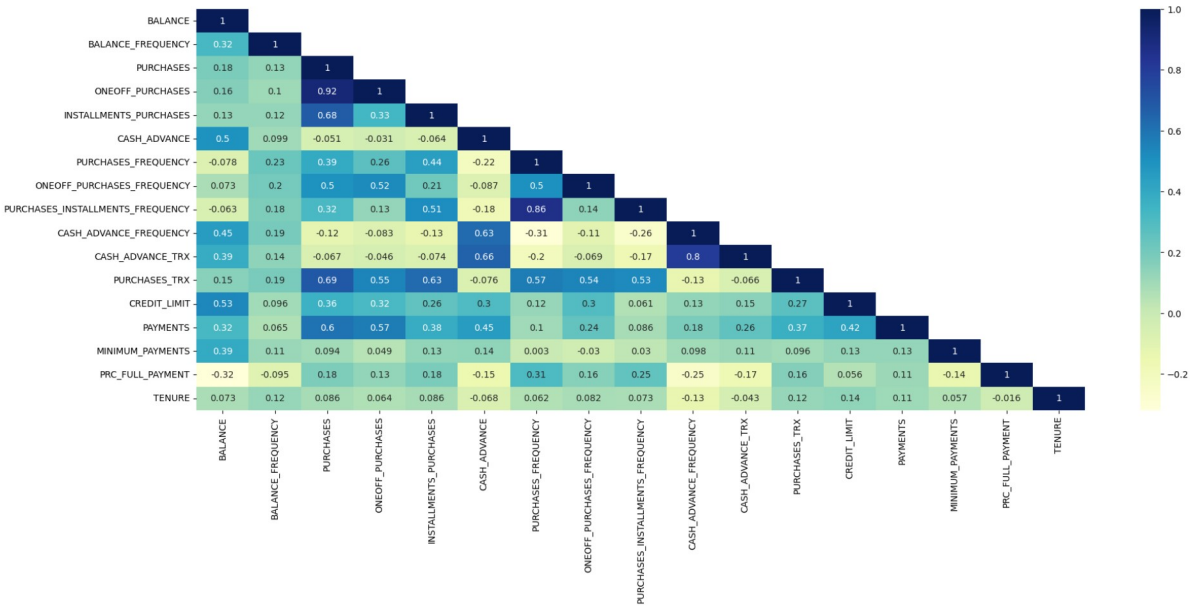
Insights:

- Todas las variables tienen un alto sesgo
- La mayoría de las viriables están positivamente sesgadas, en muchos casos el sesgo viene del alto número de registros de la variable en 0
- Para las variables limitadas de 0 a 1 (Purchases frequency, oneoff purchases frequency y purchases installments frequency) la mayoría de los registros se encuentran en los extremos (0.0 y 1.0)

In [ ]:
```
# Gráfica de correlación
import numpy as np

plt.figure(figsize=(22,8))
corr_df  = df.corr(method='pearson')
df_lt = corr_df.where(np.tril(np.ones(corr_df.shape)).astype(bool))
hmap = sns.heatmap(data=df_lt, cmap='YlGnBu', annot=True)
```

```
C:\Users\USER\AppData\Local\Temp\ipykernel_14964\1135197015.py:5: FutureWarning: Th
e default value of numeric_only in DataFrame.corr is deprecated. In a future versio
n, it will default to False. Select only valid columns or specify the value of nume
ric_only to silence this warning.
  corr_df  = df.corr(method='pearson')
```



Insights:

- Se encontraron relaciones entre variables on alta correlación:
  - .92 - ONEOFF_PURCHASES, PURCHASES
  - .86 - PURCHASES_INSTALLMENTS_FREQUENCY, PURCHASES_FREQUENCY
  - .80 - CASH_ADVANCE_TRX, CASH_ADVANCE_FREQUENCY

Feature Engineering

Eliminación de columnas

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

# Eliminación de variables no necesarias
df2.drop(columns='CUST_ID', inplace=True)

# Elimincación de variables con alta correlación
df2.drop(columns=['PURCHASES', 'PURCHASES_INSTALLMENTS_FREQUENCY', 'CASH_ADVANCE_FR
```

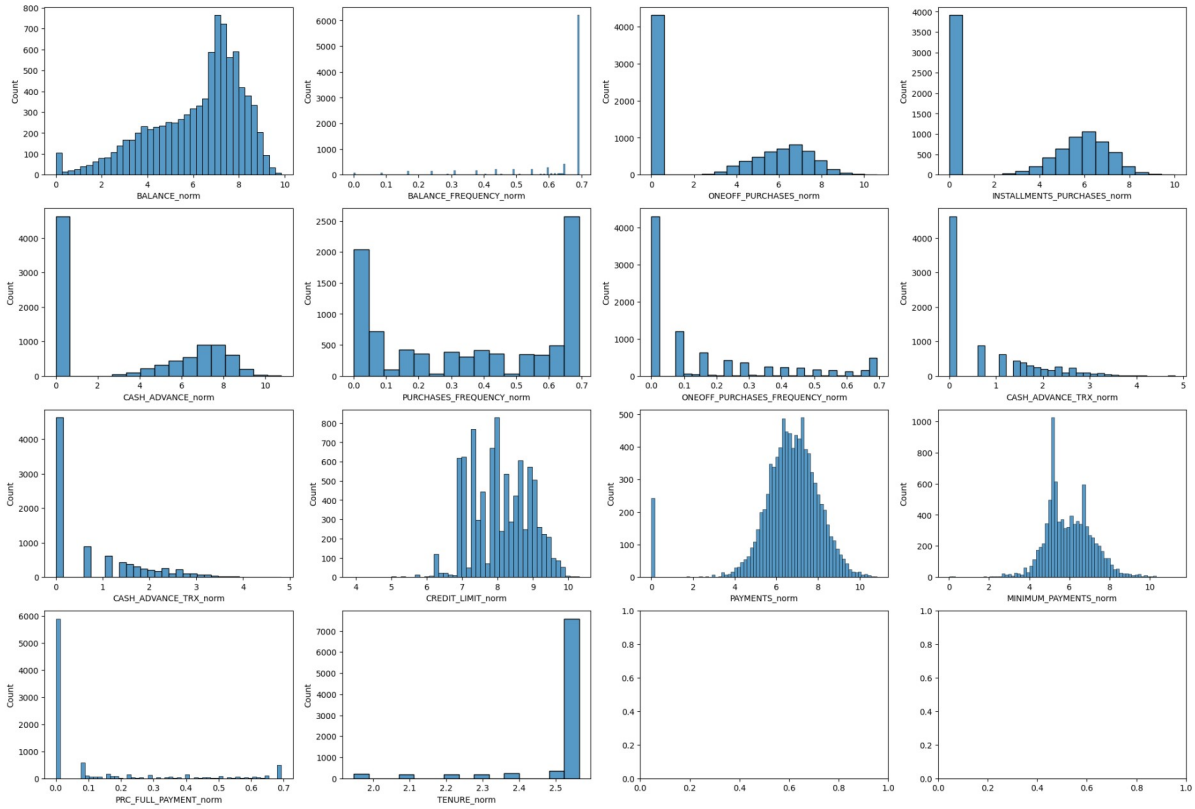
Normalización de columnas

- Balance
- Balance frequency (negative)
- One off purchases
- Installments purchases
- Cash advance
- Purchases Frequency (both)
- One off purchases frequency
- Cash advance trx
- Purchases trx
- Credit Limit
- Payments
- Minimum payments
- PRC full payment
- Tenure (negative)

In [ ]:
```
# Transform skewness with natural logarithm
df2['BALANCE_norm'] = np.log(df['BALANCE'] + 1)
df2['BALANCE_FREQUENCY_norm'] = np.log(df['BALANCE_FREQUENCY'] + 1)
df2['ONEOFF_PURCHASES_norm'] = np.log(df['ONEOFF_PURCHASES'] + 1)
df2['INSTALLMENTS_PURCHASES_norm'] = np.log(df['INSTALLMENTS_PURCHASES'] + 1)
df2['CASH_ADVANCE_norm'] = np.log(df['CASH_ADVANCE'] + 1)
df2['PURCHASES_FREQUENCY_norm'] = np.log(df['PURCHASES_FREQUENCY'] + 1)
df2['ONEOFF_PURCHASES_FREQUENCY_norm'] = np.log(df['ONEOFF_PURCHASES_FREQUENCY'] +
df2['CASH_ADVANCE_TRX_norm'] = np.log(df['CASH_ADVANCE_TRX'] + 1)
df2['PURCHASES_TRX'] = np.log(df['CASH_ADVANCE_TRX'] + 1)
df2['CREDIT_LIMIT_norm'] = np.log(df['CREDIT_LIMIT'] + 1)
df2['PAYMENTS_norm'] = np.log(df['PAYMENTS'] + 1)
df2['MINIMUM_PAYMENTS_norm'] = np.log(df['MINIMUM_PAYMENTS'] + 1)
df2['PRC_FULL_PAYMENT_norm'] = np.log(df['PRC_FULL_PAYMENT'] + 1)
df2['TENURE_norm'] = np.log(df['TENURE'] + 1)
```

In [ ]:
```
new_cols = ['BALANCE_norm',
            'BALANCE_FREQUENCY_norm',
            'ONEOFF_PURCHASES_norm',
            'INSTALLMENTS_PURCHASES_norm',
            'CASH_ADVANCE_norm',
            'PURCHASES_FREQUENCY_norm',
            'ONEOFF_PURCHASES_FREQUENCY_norm',
            'CASH_ADVANCE_TRX_norm',
            'CASH_ADVANCE_TRX_norm',
            'CREDIT_LIMIT_norm',
            'PAYMENTS_norm',
            'MINIMUM_PAYMENTS_norm',
            'PRC_FULL_PAYMENT_norm',
            'TENURE_norm']

fig, ax = plt.subplots(nrows=4, ncols=4, figsize=(25,17))
for i, column in enumerate(new_cols):
    sns.histplot(df2[column], ax=ax[i//4, i%4], kde=False)
```



Insights:

- Algunas de las variables se arreglaron correctamente y parecen tener una distribución más parecida a la Normal
- Hay otras, como INSTALLMENTS_PURCHASES_norm que sigue teniendo una cantidad alta de 0s lo cual la sesga, sin embargo, el resto de valores diferentes a cero que toma la serie sigue una distribución normal

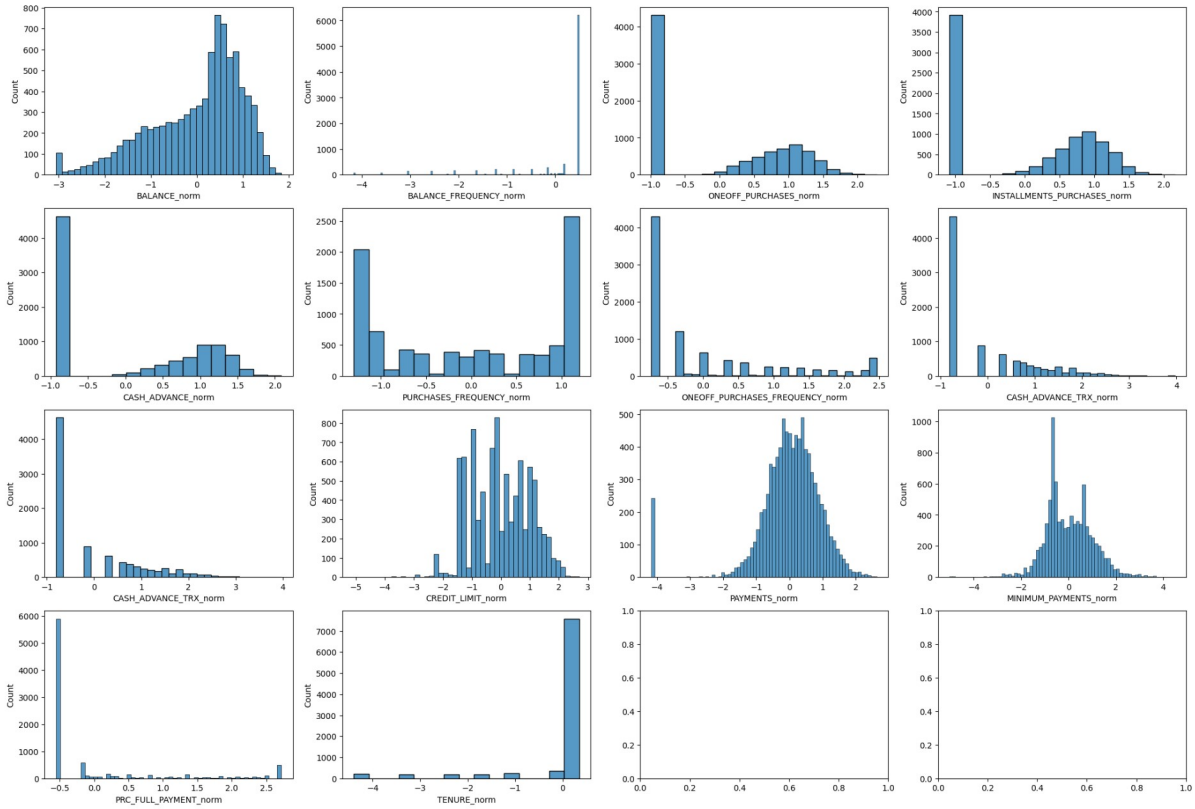
---

Estandarización de Variables

In [ ]:
```
from scipy.stats import zscore

df2 = df2.apply(zscore)
```

In [ ]:
```
fig, ax = plt.subplots(nrows=4, ncols=4, figsize=(25,17))
for i, column in enumerate(new_cols):
    sns.histplot(df2[column], ax=ax[i//4, i%4], kde=False)
```



Algorítmo de K-Means

In [ ]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 27 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   BALANCE                           8950 non-null   float64
 1   BALANCE_FREQUENCY                 8950 non-null   float64
 2   ONEOFF_PURCHASES                  8950 non-null   float64
 3   INSTALLMENTS_PURCHASES            8950 non-null   float64
 4   CASH_ADVANCE                      8950 non-null   float64
 5   PURCHASES_FREQUENCY               8950 non-null   float64
 6   ONEOFF_PURCHASES_FREQUENCY        8950 non-null   float64
 7   CASH_ADVANCE_TRX                  8950 non-null   float64
 8   PURCHASES_TRX                     8950 non-null   float64
 9   CREDIT_LIMIT                      8950 non-null   float64
 10  PAYMENTS                          8950 non-null   float64
 11  MINIMUM_PAYMENTS                  8950 non-null   float64
 12  PRC_FULL_PAYMENT                  8950 non-null   float64
 13  TENURE                            8950 non-null   float64
 14  BALANCE_norm                      8950 non-null   float64
 15  BALANCE_FREQUENCY_norm            8950 non-null   float64
 16  ONEOFF_PURCHASES_norm             8950 non-null   float64
 17  INSTALLMENTS_PURCHASES_norm       8950 non-null   float64
 18  CASH_ADVANCE_norm                 8950 non-null   float64
 19  PURCHASES_FREQUENCY_norm          8950 non-null   float64
 20  ONEOFF_PURCHASES_FREQUENCY_norm   8950 non-null   float64
 21  CASH_ADVANCE_TRX_norm             8950 non-null   float64
 22  CREDIT_LIMIT_norm                 8950 non-null   float64
 23  PAYMENTS_norm                     8950 non-null   float64
 24  MINIMUM_PAYMENTS_norm             8950 non-null   float64
 25  PRC_FULL_PAYMENT_norm             8950 non-null   float64
 26  TENURE_norm                       8950 non-null   float64
dtypes: float64(27)
memory usage: 1.8 MB
```

In [ ]:

```
df3 = df2.copy()
df3.drop(columns=[
    'BALANCE',
    'BALANCE_FREQUENCY',
    'ONEOFF_PURCHASES',
    'INSTALLMENTS_PURCHASES',
    'CASH_ADVANCE',
    'PURCHASES_FREQUENCY',
    'ONEOFF_PURCHASES_FREQUENCY',
    'CASH_ADVANCE_TRX',
    'PURCHASES_TRX',
    'CREDIT_LIMIT',
    'PAYMENTS',
    'MINIMUM_PAYMENTS',
    'PRC_FULL_PAYMENT',
    'TENURE'
], inplace=True)
```

In [ ]:
```
from scipy.spatial.distance import cdist
from sklearn.cluster import KMeans
mean_distortions = []

clusters = range(1,20)

for k in clusters:
    model = KMeans(n_clusters=k, n_init='auto')
    model.fit(df3)
    prediction = model.predict(df3)

    mean_distortions.append(sum(np.min(cdist(df3, model.cluster_centers_, 'euclidea

# Plot Scree Graph
plt.plot(clusters, mean_distortions, 'bx-')
plt.xlabel('k - # Clusters')
plt.ylabel('Distorsión Promedio')
plt.title('Resultados KMeans')
```

Out[ ]: Text(0.5, 1.0, 'Resultados KMeans')

In [ ]:
```
# Se genera el número de clusters = 8
kmeans = KMeans(n_clusters=3, n_init=15, random_state=1)
kmeans.fit(df3)
```

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

In [ ]:
```
centroids = kmeans.cluster_centers_
centroids
```

Out[ ]:
```
array([[ 0.34878789,  0.40357056,  1.01982723,  0.55304714, -0.36437249,
         0.8078559 ,  1.08566631, -0.33431607,  0.42316326,  0.46523803,
         0.18859237,  0.0962272 ,  0.22891693],
       [ 0.61346973,  0.26265939, -0.50370047, -0.71180394,  0.96063052,
        -0.83832325, -0.51893625,  0.88395126, -0.03493113,  0.08192124,
         0.46215884, -0.43769794, -0.09015574],
       [-1.10980452, -0.74116517, -0.45541352,  0.2856434 , -0.78517441,
         0.17266639, -0.50579932, -0.72351688, -0.40032766, -0.5862321 ,
        -0.75847457,  0.43077695, -0.13004021]])
```

In [ ]:
```
centroid_df = pd.DataFrame(centroids)
centroid_df
```

Out[ ]:

	0	1	2	3	4	5	6	7	8
0	0.348788	0.403571	1.019827	0.553047	-0.364372	0.807856	1.085666	-0.334316	0.423163
1	0.613470	0.262659	-0.503700	-0.711804	0.960631	-0.838323	-0.518936	0.883951	-0.034931
2	-1.109805	-0.741165	-0.455414	0.285643	-0.785174	0.172666	-0.505799	-0.723517	-0.400328

In [ ]:
```
df_labels = pd.DataFrame(kmeans.labels_, columns=list(['labels']))
df_labels['labels'] = df_labels['labels'].astype('category')
df_labels.value_counts()
```

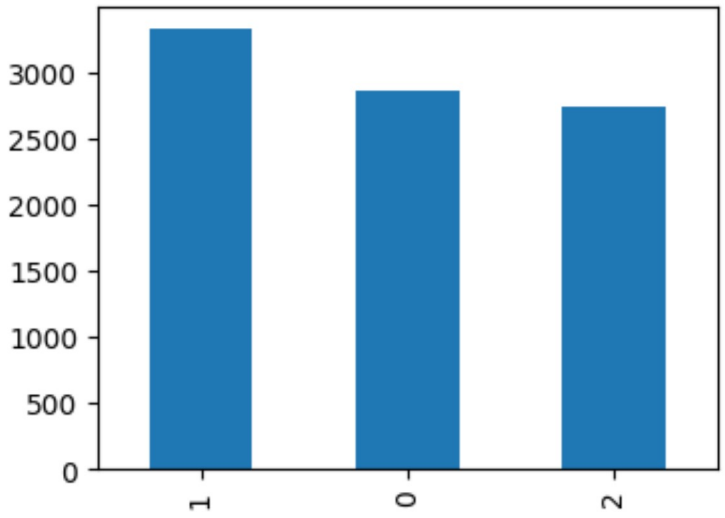
Out[ ]:
```
labels
1         3332
0         2873
2         2745
dtype: int64
```

In [ ]:
```
# Se une la predicción del bloque anterior con el registro en sí
df_labeled = df3.reset_index().join(other=df_labels, how='left')
df_labeled['labels'].value_counts()
```

Out[ ]:
```
1    3332
0    2873
2    2745
Name: labels, dtype: int64
```

In [ ]:
```
# Gráfico de clusers por registro
fig, ax = plt.subplots(1,1, figsize=(4,3))
pd.value_counts(df_labeled['labels']).plot(kind='bar')
```

Out[ ]: <AxesSubplot: >

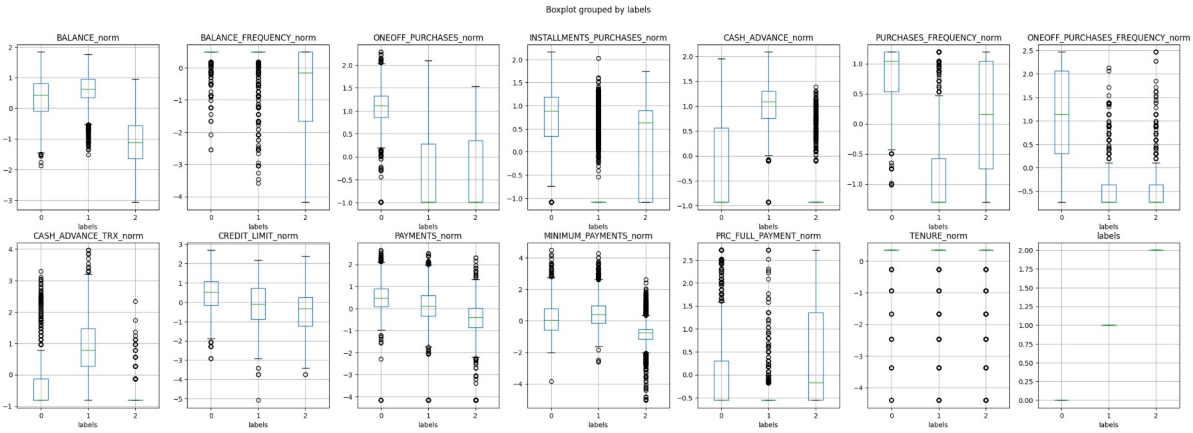


In [ ]:
```
df_labeled.drop(columns='index', inplace=True)
```

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

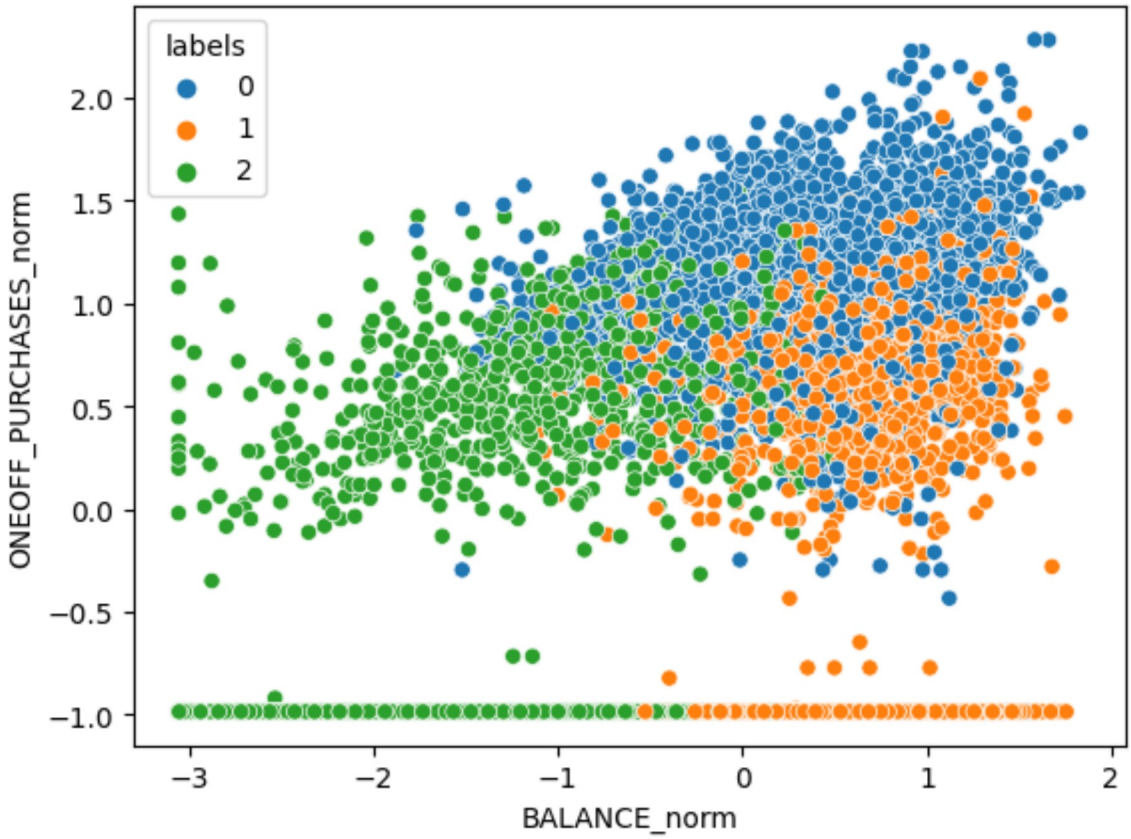
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8950 entries, 0 to 8949
Data columns (total 14 columns):
 #   Column                            Non-Null Count  Dtype
---  ------                            --------------  -----
 0   BALANCE_norm                      8950 non-null   float64
 1   BALANCE_FREQUENCY_norm            8950 non-null   float64
 2   ONEOFF_PURCHASES_norm             8950 non-null   float64
 3   INSTALLMENTS_PURCHASES_norm       8950 non-null   float64
 4   CASH_ADVANCE_norm                 8950 non-null   float64
 5   PURCHASES_FREQUENCY_norm          8950 non-null   float64
 6   ONEOFF_PURCHASES_FREQUENCY_norm   8950 non-null   float64
 7   CASH_ADVANCE_TRX_norm             8950 non-null   float64
 8   CREDIT_LIMIT_norm                 8950 non-null   float64
 9   PAYMENTS_norm                     8950 non-null   float64
 10  MINIMUM_PAYMENTS_norm             8950 non-null   float64
 11  PRC_FULL_PAYMENT_norm             8950 non-null   float64
 12  TENURE_norm                       8950 non-null   float64
 13  labels                            8950 non-null   category
dtypes: category(1), float64(13)
memory usage: 918.0 KB
```

In [ ]:
```
# Gráfico de clases
fig, ax = plt.subplots(nrows=2, ncols=7, figsize=(30,10))
fig.suptitle('Configuración de clusters por variable')
for i, col in enumerate(df_labeled.columns):
    df_labeled.boxplot(col, 'labels', ax=ax[i//7, i%7])
```



In [ ]:
```
# Scatter plot comparando las clases en dentro de dos variables
# Balance y one of purchases
sns.scatterplot(data=df_labeled, x='BALANCE_norm', y='ONEOFF_PURCHASES_norm', hue='
```

Out[ ]: <AxesSubplot: xlabel='BALANCE_norm', ylabel='ONEOFF_PURCHASES_norm'>



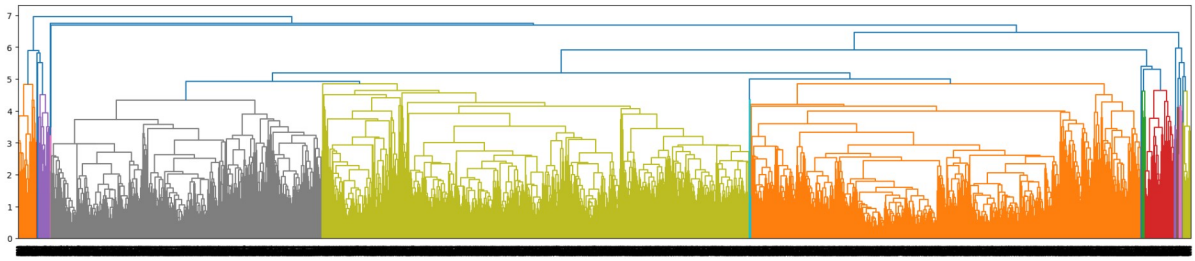
Insights:

- Se puede observar, por ejemplo, como la clase 2 se encuentra principalmente en los niveles más bajos de la variable BALANCE y niveles bajos de la variable ONEOFF_PURCHASES, mientras que la clase 0 se encuentra casi exclusivamente en valores altos de la variable BALANCEy principalmente en la región donde la variable BALANCE_norm es mayor

In [ ]:
```
# Dendogram
from scipy.cluster.hierarchy import dendrogram, linkage

z = linkage(df3, 'average', metric='euclidean')

plt.figure(figsize=(25,5))
dendrogram(z)
plt.show()
```



In [ ]:
```
# Validación de la calidad del fit (coeficiente cofenético)
from scipy.cluster.hierarchy import cophenet
from scipy.spatial.distance import pdist

c, coph_dists = cophenet(z, pdist(df3))
print('Cophenetic Coefficient: ', format(c, '.4f'))
```

```
Cophenetic Coefficient:  0.6725
```

In [ ]:
```
# Silhouette score
from sklearn.metrics import silhouette_score

sil_df = pd.DataFrame({}, columns=['Model', 'n_clusters', 'Score'])

for n_clusters in range(2,20):
    clusterer = KMeans(n_clusters=n_clusters, random_state=1, n_init='auto')
    predictions = clusterer.fit_predict(df3)
    centers = clusterer.cluster_centers_

    score = silhouette_score(df3, clusterer.labels_, metric='euclidean')

    # Add score to sil_df
    sil_df = pd.concat(objs=[sil_df, pd.DataFrame([['KMeans', n_clusters, score]],
```

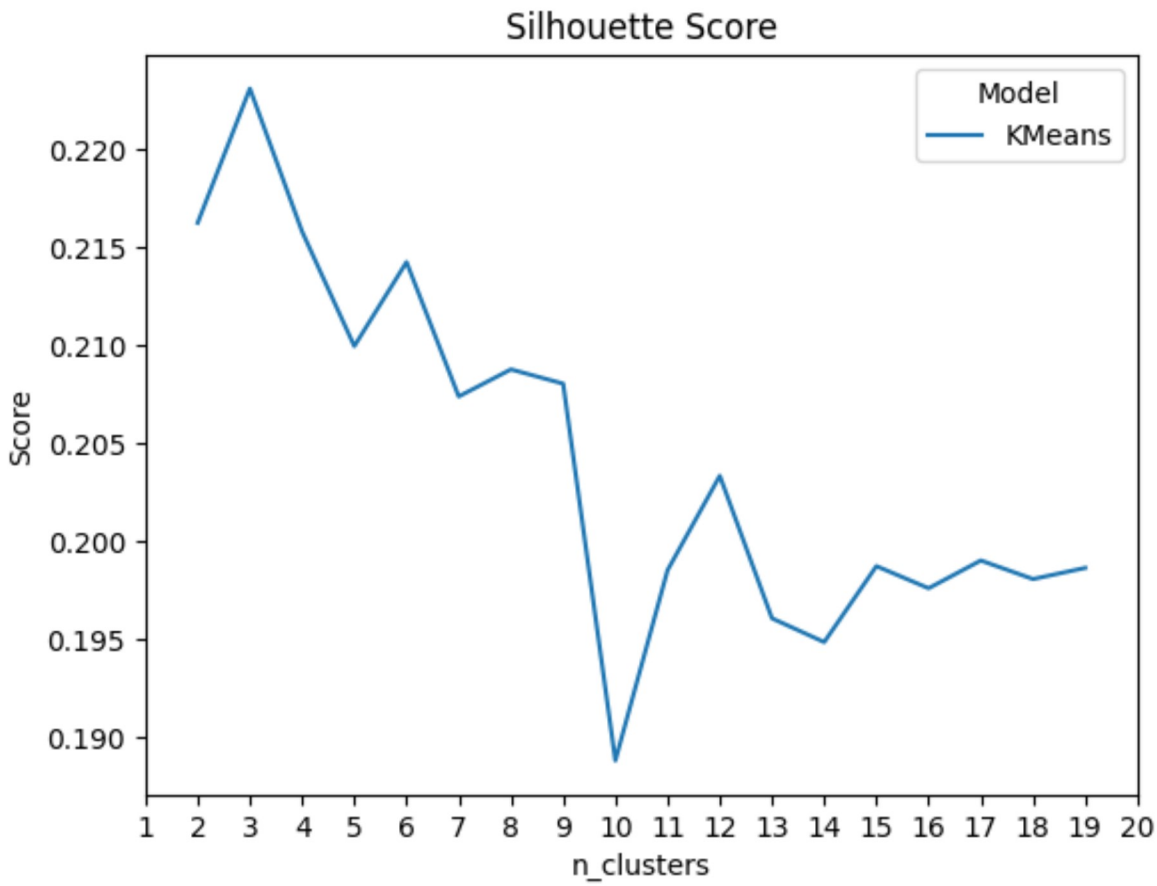
In [ ]:
```
sil_df
```

Out[ ]:

	Model	n_clusters	Score
0	KMeans	2	0.216238
0	KMeans	3	0.223088
0	KMeans	4	0.215789
0	KMeans	5	0.209942
0	KMeans	6	0.214231
0	KMeans	7	0.207373
0	KMeans	8	0.208755
0	KMeans	9	0.208027
0	KMeans	10	0.188791
0	KMeans	11	0.198508
0	KMeans	12	0.203323
0	KMeans	13	0.196050
0	KMeans	14	0.194837
0	KMeans	15	0.198714
0	KMeans	16	0.197585
0	KMeans	17	0.199002
0	KMeans	18	0.198055
0	KMeans	19	0.198623

In [ ]:

```
sns.lineplot(data=sil_df, x='n_clusters', y='Score', hue='Model', style='Model').se
plt.xticks(ticks=np.arange(1,21,1))
plt.show()
```



Insights

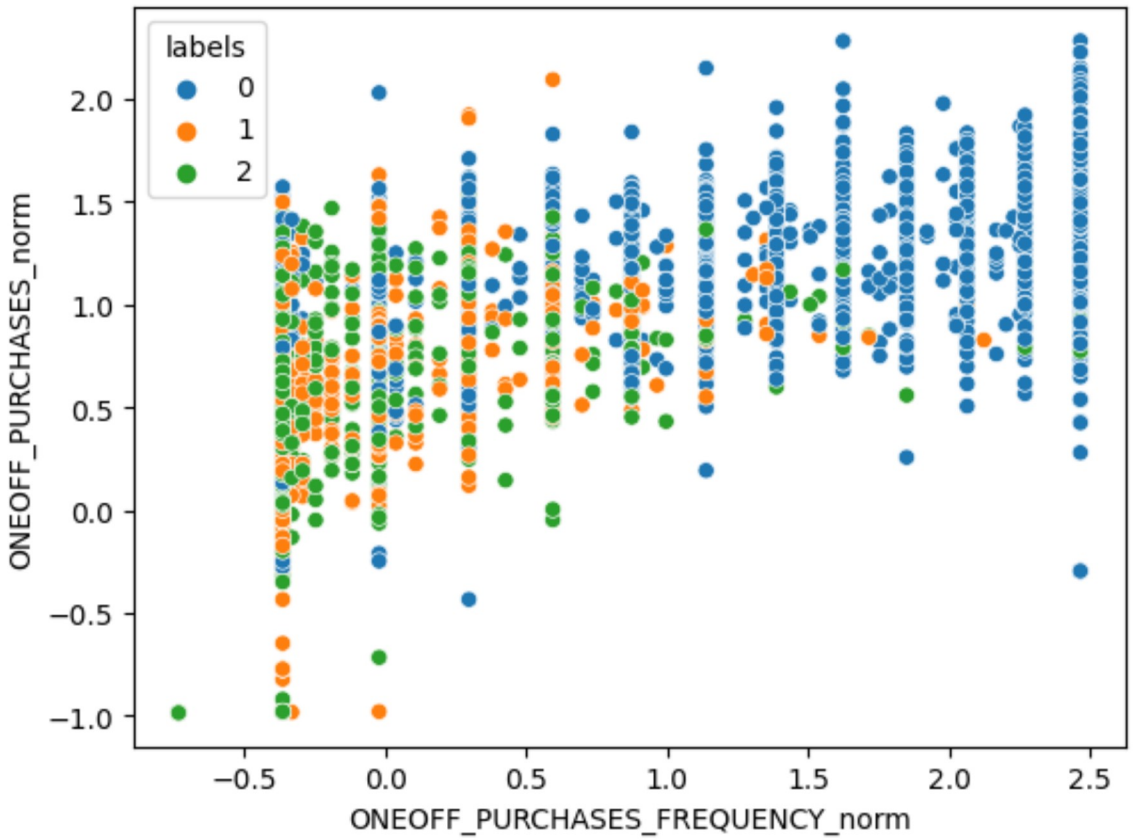
- Originalmente el modelo se corrió con 8 clusters, sin embargo, despues de revisar el gráfico con el Silhouette Score por n_clusters, se vió que el mejor número de clusters debería ser 3 por lo que se corrigió el código de arriba y las conclusiones a las que se llegaron en cada paso

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

Out[ ]:
```
BALANCE_norm                      8871
BALANCE_FREQUENCY_norm              43
ONEOFF_PURCHASES_norm             4014
INSTALLMENTS_PURCHASES_norm       4452
CASH_ADVANCE_norm                 4323
PURCHASES_FREQUENCY_norm            47
ONEOFF_PURCHASES_FREQUENCY_norm     47
CASH_ADVANCE_TRX_norm               65
CREDIT_LIMIT_norm                  206
PAYMENTS_norm                     8711
MINIMUM_PAYMENTS_norm             8637
PRC_FULL_PAYMENT_norm               47
TENURE_norm                          7
labels                               3
dtype: int64
```

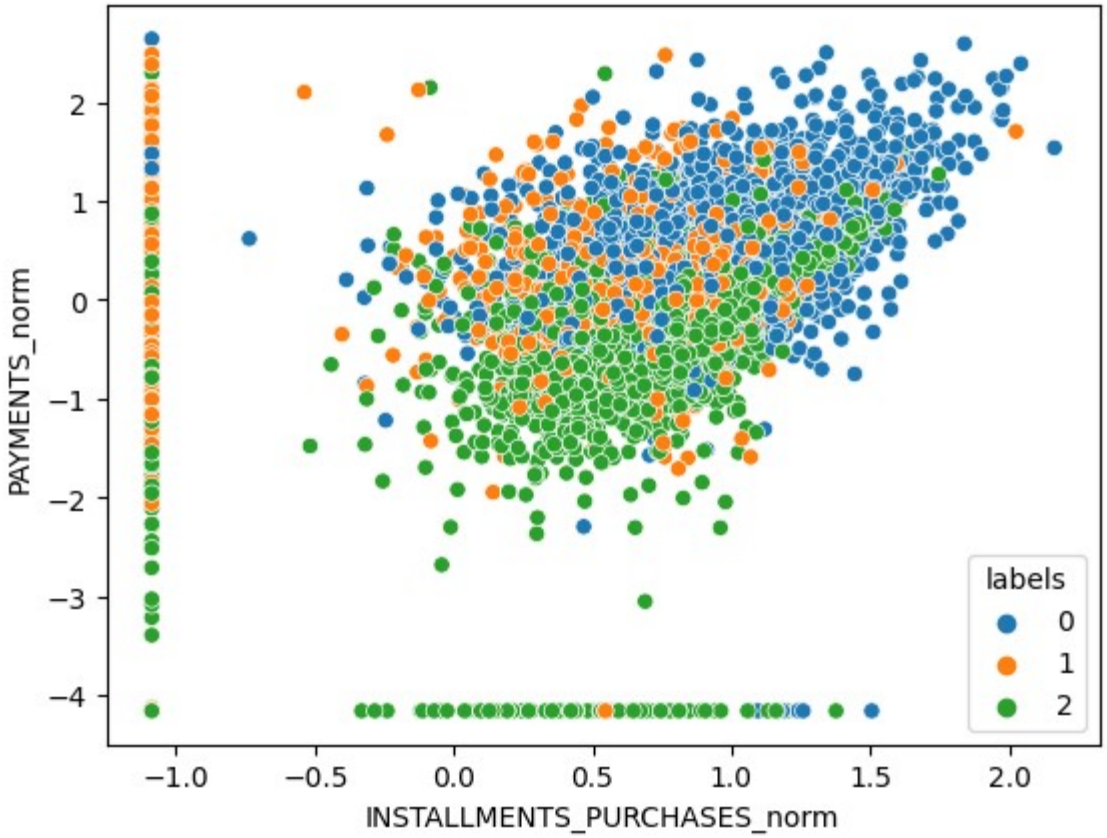
In [ ]:
```
# Otras visualizaciones de los resultados del modelo de clustering (scatterplots co
sns.scatterplot(data=df_labeled, x='ONEOFF_PURCHASES_FREQUENCY_norm', y='ONEOFF_PUR
```

Out[ ]:
```
<AxesSubplot: xlabel='ONEOFF_PURCHASES_FREQUENCY_norm', ylabel='ONEOFF_PURCHASES_no
rm'>
```



In [ ]:
```
sns.scatterplot(data=df_labeled, x='INSTALLMENTS_PURCHASES_norm', y='PAYMENTS_norm'
```

Out[ ]:
```
<AxesSubplot: xlabel='INSTALLMENTS_PURCHASES_norm', ylabel='PAYMENTS_norm'>
```



Insights:

- Como se puede ver en los dos gráficos finales, los cuales fueron escogidos puesto que en la visualización de gráficos de caja se notaba mayor diferencia entre los grupos, los clusters se diferencían por medio de los colores. En el primer gráfico se ve como el grupo 0 predomina en valores altos de ONEOFF_PURCHASES_FREQUENCY_norm, mientras que los otros dos grupos se mezclan más en valores pequeños de dicha variable. En el segundo gráfico se puede ver como el grupo 2 predocumna en valores bajos de la variable PAYMENTS_norm mientras que el grupo 0 predomina en valores altos de ambas variables graficadas
- En un inicio el gráfico de codo sirvió para hacer un primer análisis donde se escogió dividir la muestra en 8 clusters, sin embargo, al realizar el diagrama de Silhouette Score contra el número de clusters, se puede ver que el número óptimo de clusters es 3 por lo que se hicieron las correcciones al modelo y se reinterpretaron los resultados en cada sección de insights