

Master UNIR

Presentado por:

Fecha: 10/01/2023

Importación de librerías

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
import matplotlib as mpl

from datetime import datetime
from datetime import date
import os

import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)
warnings.filterwarnings('ignore')

from sklearn.preprocessing import OrdinalEncoder
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import AgglomerativeClustering

from sklearn.metrics import silhouette_score
```

```
In [2]: #Obtener el directorio de trabajo y contenido del mismo
cwd = os.getcwd()
files = os.listdir(cwd)
print("Files in %r: %s" % (cwd, files))
```

```
Files in 'C:\\\\Users\\\\80900946\\\\tfm': ['.ipynb_checkpoints', 'ejem.csv', 'Exploracion - copia.ipynb', 'Exploracion.ipynb', 'Exploracion_30012024.ipynb', 'Exploracion_2PRU.ipynb', 'otros']
```

Cargar el Dataset

Se ha dispuesto información diaria de productos de ahorro de 321179 Clientes con datos que proveen el monto de compras, cantidad de transacciones, descripción del tipo de transacción, saldos por producto, caracterización demográfica del cliente de una entidad financiera en Colombia con los productos de ahorro para los meses de Marzo - Agosto - Septiembre - Octubre y Diciembre del 2021.

Información limitada por disposición de la empresa y a la cual nos referimos en el estudio de Segmentación y recomendación objeto del presente trabajo.

La información fue entregada en archivo plano CSV, el cual es fuente única para el análisis, tratamiento y adaptación del código implementado, no es posible publicar esta data y su análisis está restringido a la entrega de resultados propios del estudio.

De igual forma se realiza lectura del archivo de saldos diarios que contiene el saldo de los productos de cada cliente en el periodo de estudio

```
In [3]: from pathlib import Path
file_path=Path(r"D:\tfm\202112_SIN_DUP.csv")
#Lectura del archivo
data=pd.read_csv(file_path, sep=";", index_col=False, encoding='latin1')
```

```
In [4]: file_path_saldos=Path(r"D:\tfm\SALDOS.csv")
#Lectura del archivo para saldo de la cuenta
saldos=pd.read_csv(file_path_saldos, sep=",", index_col=False, encoding='latin1')
```

```
In [5]: saldos
```

	NO_PRODUCTO	PROM
0	210587036559	2.566076e+05
1	230445103823	2.309747e+05
2	230280253972	1.492769e+06
3	230410048987	3.193024e+05
4	230364657239	7.657032e+03
...
204514	210600323182	6.720105e+06
204515	500801198945	1.425504e+05
204516	260435124045	3.594816e+07
204517	260551191547	2.977027e+07
204518	260578019812	2.728442e+07

204519 rows × 2 columns

```
In [6]: # Dimension del Dataset  
data.shape
```

```
Out[6]: (892402, 30)
```

Caracterización del Dataset

Se incluye una descripción de los datos con:

- Número de instancias en total:
 - 892402 observaciones
- Número de atributos de entrada, su significado y tipo.
 - 30 variables de entrada.

- SK_CLIENTE: Identificoar el cliente
- NO_PRODUCTO: Numero el producto del cliente
- SK_TRANSACCION: Identificador d ela transaccion
- DS_NOMBRE: Descripcio nde la transaccion
- SK_FE_TRANSACCION: Fecha Transaccion
- SK_PRODUCTO_SERVICIO: Identificador del producto
- DS_LINEA_PRODUCTO: Linea del producto
- DS_FAMILIA: Familia del producto - valor único CAPTACION
- DS_CLASE: Cuenta corriente - Cuenta ahorros
- VALOR_MVTO: valor monto
- CANTIDAD_TRANSACCION: monto de la trasaccion
- DS_SECTOR_PIB: Sector de Servicio
- DS_TIPO_EMPRESA: sin valores
- DS_TIPO_PERSONA: Tpo persona cliente
- DS_GENERO: Genero Cliente
- FE_VINCULACION_CLIENTE: Fcha vinculacion del cliente
- DS_ESTADO_CIVIL: Estado civil del cliente
- FE_NACIMIENTO: Fe_nacimiento del cliente
- DS_PAIS_NACIMIENTO: Pais de nacimiento del cliente
- DS_NIVEL_ESTUDIOS: El ultimo Tipo de estudio finalizado
- DS_PROFESION: Profesion del cliente
- DS_OCUPACION: Ocupacion del cliente
- DS_TIPO_VIVIENDA: tipo de vivienda del cliente
- DS_RAZON_SOCIAL: Razon social del cliente
- ID_CLIENTE: Identificacion del cliente
- NO_PERSONAS_A_CARGO: Numero d epersonas a cargo del titular
- REF_NUM: Numero eidentificacion del cliente
- FE_APERTURA: Fecha de apertura del producto
- DK_PERSONA: Identificador d ela persona
- SK_RC: Identificador el producto

Se agrega la lectura de un nuevo archivo con el saldo promedio por producto, el cual será cruzado con la data transaccional para obtener el promedio de saldo por cliente

In [7]: #Código que responde a La descripción anterior

```
#Cantidad de Instancias y atributos
```

```
print('Cantidad de Instancias y columnas:',data.shape)
print('Nombre columnas:',data.columns)
```

Cantidad de Instancias y columnas: (892402, 30)

Nombre columnas: Index(['SK_CLIENTE', 'NO_PRODUCTO', 'SK_TRANSACCION', 'DS_NOMBRE',
 'SK_FE_TRANSACCION', 'SK_PRODUCTO_SERVICIO', 'DS_LINEA_PRODUCTO',
 'DS_FAMILIA', 'DS_CLASE', 'VALOR_MVTO', 'CANTIDAD_TRANSACCION',
 'DS_SECTOR_PIB', 'DS_TIPO_EMPRESA', 'DS_TIPO_PERSONA', 'DS_GENERO',
 'FE_VINCULACION_CLIENTE', 'DS_ESTADO_CIVIL', 'FE_NACIMIENTO',
 'DS_PAIS_NACIMIENTO', 'DS_NIVEL_ESTUDIOS', 'DS_PROFESION',
 'DS_OCUPACION', 'DS_TIPO_VIVIENDA', 'DS_RAZON_SOCIAL', 'ID_CLIENTE',
 'NO_PERSONAS_A_CARGO', 'REF_NUM', 'FE_APERTURA', 'DK_PERSONA', 'SK_RC'],
 dtype='object')

In [8]: #Primeros registros del data set

```
data.head()
```

Out[8]:

	SK_CLIENTE	NO_PRODUCTO	SK_TRANSACCION	DS_NOMBRE	SK_FE_TRANSACCION	SK_
0	2582480	110014145338		714	IVA SOBRE CHEQUERAS (IGUAL SER	20211206
1	2582480	110014145338		758	COBRO DE CHEQUERA (IGUAL SERIE	20211206
2	2582480	110014145338		644	ND.TIMBRES CHEQUERA	20211206
3	2582480	110014145338		2562	CONSIGNACION aval	20211206
4	2582480	110014145338		345	ABONOS POR A.C.H	20211206

5 rows × 30 columns

In [5]:

```
"""
Eliminar las columnas:

DS_FAMILIA ya que es un único valor
REF_NUM ya que identifica al cliente de forma unica función que tiene el campo
DS_NOMBRE y DS_RAZON_SOCIAL, ya que para el objeto del estudio no es prioritaria

"""

data= data.drop(columns=['DS_FAMILIA', 'REF_NUM', 'DS_NOMBRE', 'DS_RAZON_SOCIAL'])
data.head()
```

Out[5]:

	SK_CLIENTE	NO_PRODUCTO	SK_TRANSACCION	SK_FE_TRANSACCION	SK_PRODUCTO_SERV
0	2582480	110014145338		714	20211206
1	2582480	110014145338		758	20211206
2	2582480	110014145338		644	20211206
3	2582480	110014145338		2562	20211206
4	2582480	110014145338		345	20211206

5 rows × 26 columns

In [6]: #Porcentaje de Datos Nulos (0.21)

```
valores_null= data.isnull().sum()
celdas = np.product(data.shape)
total= valores_null.sum()

(total/celdas) * 100
```

Out[6]: 0.21065015025136138

In [9]:

```
"""
Aunque existen valores Null sobre los campos , Fecha de nacimiento, No_personas,
Ds_Clase y sk_rc, no se les hará tratamiento. Para la Fecha vinculacion "Null"
del producto (394 registros).
"""

total = data.isnull().sum().sort_values(ascending=False)
percent_1 = data.isnull().sum()/data.isnull().count()*100
percent_2 = (round(percent_1, 4)).sort_values(ascending=False)
missing_data = pd.concat([total, percent_2], axis=1, keys=['Total', '%'])
missing_data
```

Out[9]:

	Total	%
FE_NACIMIENTO	48464	5.4307
FE_VINCULACION_CLIENTE	394	0.0442
SK_RC	12	0.0013
NO_PERSONAS_A_CARGO	4	0.0004
DS_CLASE	2	0.0002
SK_FE_TRANSACCION	0	0.0000
SK_TRANSACCION	0	0.0000
DK_PERSONA	0	0.0000
FE_APERTURA	0	0.0000
ID_CLIENTE	0	0.0000
DS_TIPO_VIVIENDA	0	0.0000
DS_OCUPACION	0	0.0000
DS_PROFESION	0	0.0000
DS_NIVEL_ESTUDIOS	0	0.0000
DS_PAIS_NACIMIENTO	0	0.0000
DS_ESTADO_CIVIL	0	0.0000
SK_PRODUCTO_SERVICIO	0	0.0000
NO_PRODUCTO	0	0.0000
DS_GENERO	0	0.0000
DS_TIPO_PERSONA	0	0.0000
DS_TIPO_EMPRESA	0	0.0000
DS_SECTOR_PIB	0	0.0000
CANTIDAD_TRANSACCION	0	0.0000
VALOR_MVTO	0	0.0000
DS_LINEA_PRODUCTO	0	0.0000
SK_CLIENTE	0	0.0000

In [7]: *#valor asignado a la fecha de vinculacion cuando es Nula*

```
data['FE_VINCULACION_CLIENTE'].fillna('FE_APERTURA', inplace=True)
```

In [8]: *#Elimina Nulos*

```
data=data.dropna()
```

Elimina Duplicados

```
data=data.drop_duplicates()
```

In [83]: *# Identificar que tipo de dato referencia cada atributo*

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 843932 entries, 12 to 892401
Data columns (total 26 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   SK_CLIENTE        843932 non-null   int64  
 1   NO_PRODUCTO       843932 non-null   int64  
 2   SK_TRANSACCION    843932 non-null   int64  
 3   SK_FE_TRANSACCION 843932 non-null   int64  
 4   SK_PRODUCTO_SERVICIO 843932 non-null   int64  
 5   DS_LINEA_PRODUCTO 843932 non-null   object  
 6   DS_CLASE          843932 non-null   object  
 7   VALOR_MVTO         843932 non-null   float64 
 8   CANTIDAD_TRANSACCION 843932 non-null   int64  
 9   DS_SECTOR_PIB      843932 non-null   object  
 10  DS_TIPO_EMPRESA    843932 non-null   object  
 11  DS_TIPO_PERSONA    843932 non-null   object  
 12  DS_GENERO          843932 non-null   object  
 13  FE_VINCULACION_CLIENTE 843932 non-null   object  
 14  DS_ESTADO_CIVIL    843932 non-null   object  
 15  FE_NACIMIENTO      843932 non-null   object  
 16  DS_PAIS_NACIMIENTO 843932 non-null   object  
 17  DS_NIVEL_ESTUDIOS   843932 non-null   object  
 18  DS_PROFESION        843932 non-null   object  
 19  DS_OCUPACION        843932 non-null   object  
 20  DS_TIPO_VIVIENDA    843932 non-null   object  
 21  ID_CLIENTE          843932 non-null   int64  
 22  NO_PERSONAS_A_CARGO 843932 non-null   float64 
 23  FE_APERTURA         843932 non-null   object  
 24  DK_PERSONA          843932 non-null   object  
 25  SK_RC               843932 non-null   float64 

dtypes: float64(3), int64(7), object(16)
memory usage: 173.8+ MB
```

In [13]: *#data['FE_NACIMIENTO'] = data['FE_NACIMIENTO'].astype('str')*

```
#drop_column = ['ind_nuevo', 'indrel', 'indresi', 'indfall', 'tipodom', 'ind_empleado']
#df.drop(drop_column, axis=1, inplace = True)
```

```
In [14]: #data['FE_NACIMIENTO'].max()
#data['FE_APERTURA'].max()
#data['EDAD_ANIOS'].unique()
#data.loc[:, data.dtypes == 'object'] = data.select_dtypes(['object']).apply(lambda x: x.astype('category'))
#data[data.SK_CLIENTE.isin([1802756])]
```

In [9]: *#Se Convierte el formato de los campos de fecha*

```
data['FE_NACIMIENTO']= pd.to_datetime(data.FE_NACIMIENTO, format='%Y-%m-%d %H:%M:%S.%f')
data['FE_VINCULACION_CLIENTE']= pd.to_datetime(data.FE_VINCULACION_CLIENTE, format='%Y-%m-%d %H:%M:%S.%f')
data['FE_APERTURA']= pd.to_datetime(data.FE_APERTURA, format='%Y-%m-%d %H:%M:%S.%f')
data['SK_FE_TRANSACCION']= pd.to_datetime(data.SK_FE_TRANSACCION, format='%Y%m%d %H:%M:%S.%f')
```

In [10]: *#Se agregan 3 calculos para La Edad, antiguedad del cliente y tiempo de apertura de cuenta*

```
today = pd.datetime.now()
data["EDAD_ANIOS"] = np.floor(((today - data['FE_NACIMIENTO']).dt.days)/365)
data["ANTIGUEDAD_ANIOS"] = np.floor(((today - data['FE_VINCULACION_CLIENTE']).dt.days)/365)
data["APERTURA_ANIOS"] = np.floor(((today - data['FE_APERTURA']).dt.days)/365)

# Se extrae mes de la transaccion

data["Mes_Transact"]= pd.DatetimeIndex(data["SK_FE_TRANSACCION"]).month
```

In [11]: *#Se agrega a la data transaccional la informacion del promedio del saldo de cada producto*

```
data= pd.merge(data, saldos[['PROM', 'NO_PRODUCTO']],on='NO_PRODUCTO', how='left')
data['PROM'].fillna(0, inplace=True)
```

In [17]: `data.info()`

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 843932 entries, 0 to 843931
Data columns (total 31 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   SK_CLIENTE        843932 non-null   int64  
 1   NO_PRODUCTO       843932 non-null   int64  
 2   SK_TRANSACCION    843932 non-null   int64  
 3   SK_FE_TRANSACCION 843932 non-null   datetime64[ns] 
 4   SK_PRODUCTO_SERVICIO 843932 non-null   int64  
 5   DS_LINEA_PRODUCTO 843932 non-null   object  
 6   DS_CLASE          843932 non-null   object  
 7   VALOR_MVTO         843932 non-null   float64 
 8   CANTIDAD_TRANSACCION 843932 non-null   int64  
 9   DS_SECTOR_PIB      843932 non-null   object  
 10  DS_TIPO_EMPRESA    843932 non-null   object  
 11  DS_TIPO_PERSONA    843932 non-null   object  
 12  DS_GENERO          843932 non-null   object  
 13  FE_VINCULACION_CLIENTE 843889 non-null   datetime64[ns] 
 14  DS_ESTADO_CIVIL     843932 non-null   object  
 15  FE_NACIMIENTO       843900 non-null   datetime64[ns] 
 16  DS_PAIS_NACIMIENTO 843932 non-null   object  
 17  DS_NIVEL_ESTUDIOS   843932 non-null   object  
 18  DS_PROFESION        843932 non-null   object  
 19  DS_OCUPACION        843932 non-null   object  
 20  DS_TIPO_VIVIENDA    843932 non-null   object  
 21  ID_CLIENTE          843932 non-null   int64  
 22  NO_PERSONAS_A_CARGO 843932 non-null   float64 
 23  FE_APERTURA         843932 non-null   datetime64[ns] 
 24  DK_PERSONA          843932 non-null   object  
 25  SK_RC               843932 non-null   float64 
 26  EDAD_ANIOS          843900 non-null   float64 
 27  ANTIGUEDAD_ANIOS    843889 non-null   float64 
 28  APERTURA_ANIOS       843932 non-null   float64 
 29  Mes_Transact        843932 non-null   int64  
 30  PROM                843932 non-null   float64 
dtypes: datetime64[ns](4), float64(7), int64(7), object(13)
memory usage: 206.0+ MB
```

In [12]: `#Cantidad de Clientes --> 321203`

```
unique_customers = set(data.SK_CLIENTE.unique())
print("Numero de Clientes: ", len(unique_customers))
```

Numero de Clientes: 321203

In [13]: #Porcentaje de Distribucion por Linea de Producto

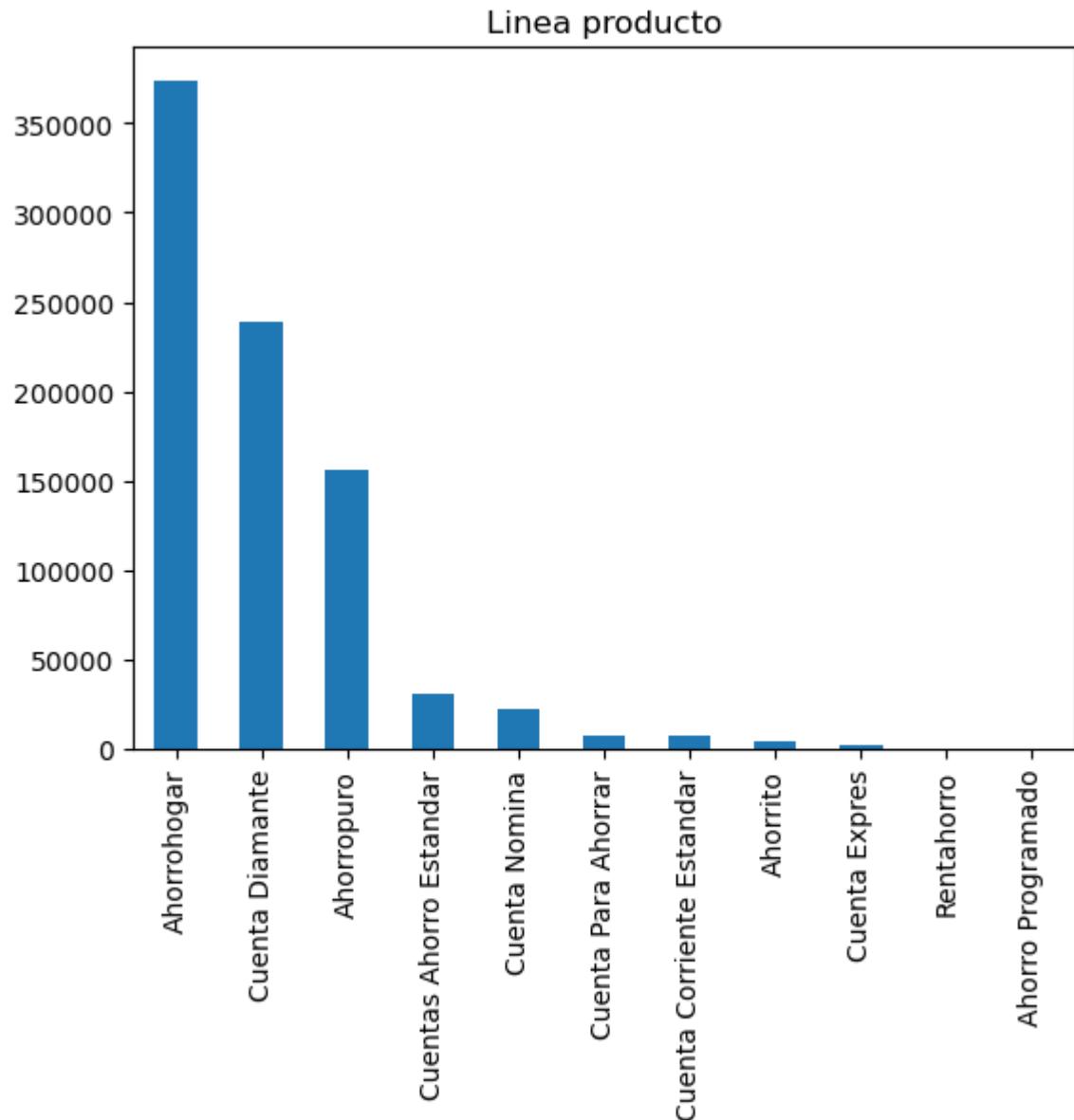
```
100*data['DS_LINEA_PRODUCTO'].value_counts()/len(data['DS_LINEA_PRODUCTO'])
```

Out[13]:

Ahorrohogar	44.274657
Cuenta Diamante	28.292090
Ahorropuro	18.551376
Cuentas Ahorro Estandar	3.699469
Cuenta Nomina	2.586701
Cuenta Para Ahorrar	0.938109
Cuenta Corriente Estandar	0.908130
Ahorrito	0.528479
Cuenta Expres	0.213169
Rentahorro	0.006754
Ahorro Programado	0.001066
Name: DS_LINEA_PRODUCTO, dtype: float64	

In [20]: #Gráfico de barras tendencia Linea Producto

```
plot=data['DS_LINEA_PRODUCTO'].value_counts().plot(kind="bar", title="Linea produ
```



In [21]:

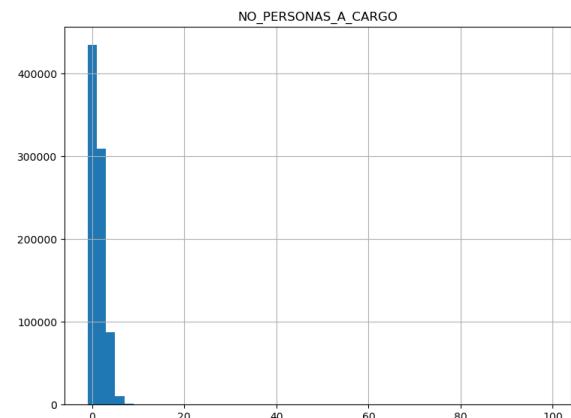
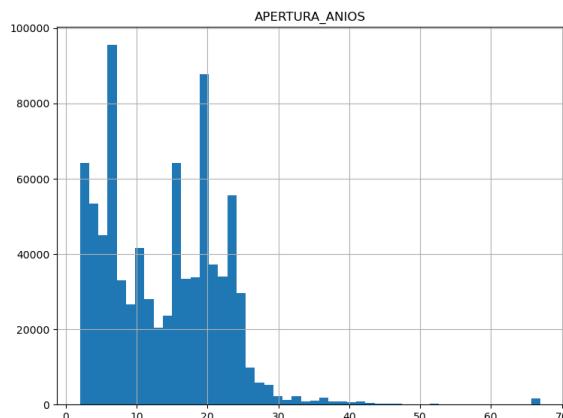
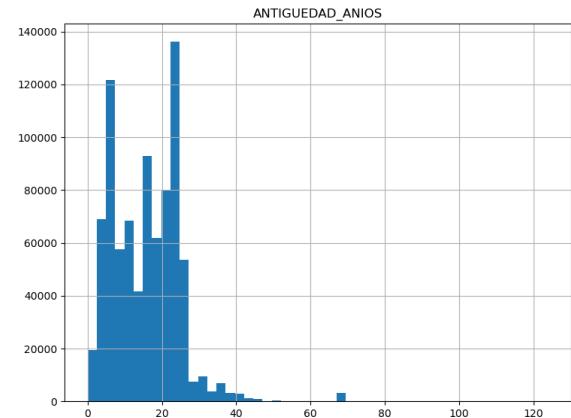
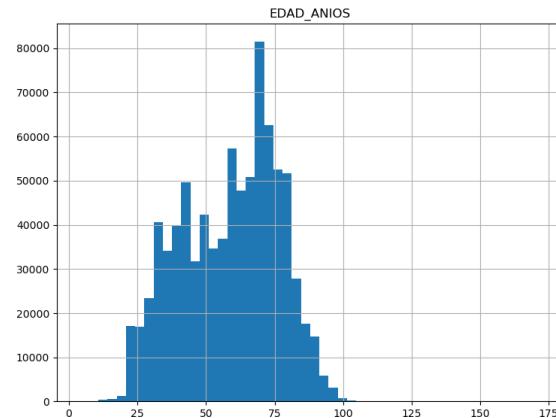
"""

Caracterizacion de los clientes:

- * El 14 % de las transacciones estan asociadas a clientes entre los 68 y 75 años
- * El 14 % de las transacciones estan asignadas a clientes con una antiguedad de 10 años o más
- * El 22% de las transacciones estan dados sobre productos con 4-5-6 y 7 años de antiguedad
- * El 50% de los clientes del análisis no tienen personas a cargo, hay un 36 % que tienen 1 persona

"""

data_pre=data[['EDAD_ANIOS', 'ANTIGUEDAD_ANIOS', 'APERTURA_ANIOS', 'NO_PERSONAS_A_CARGO']]

data_pre.hist(bins=50, figsize=(20,15))
plt.show()

In [88]: # Distribucion de las Edades

```
fig = plt.figure(figsize = (16,5))

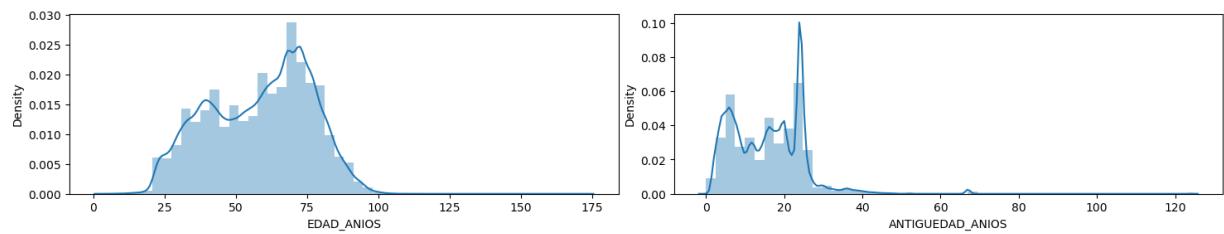
#distplot
ax1 = fig.add_subplot(121)
sns.distplot(data["EDAD_ANIOS"], kde=True)

#Cajas

ax1 = fig.add_subplot(122)
sns.distplot(data["ANTIGUEDAD_ANIOS"], kde=True)

# Mostrar figura
fig.suptitle("Distribucion De la Edad y Antiguedad", fontsize=20)
plt.tight_layout(pad=5, w_pad=0.5, h_pad=.1)
plt.show()
```

Distribucion De la Edad y Antiguedad



In [23]:

```
"""
Visualizacion valores por variable descriptiva
    * La clase de productos analizados son Cuentas de Ahorro (99%) y Cuentas Corriente (1%)
    * Los propietarios de los productos son en un 98% de Colombia y el restante en Venezuela

print("\nValores: Variable DS_LINEA_PRODUCTO: \n\n",data['DS_LINEA_PRODUCTO'].value_counts())
print("\nValores: Variable DS_CLASE: \n\n",data['DS_CLASE'].value_counts())
print("\nValores: Variable DS_PAIS_NACIMIENTO: \n\n",data['DS_PAIS_NACIMIENTO'].value_counts())
print("\nValores: Variable DS_NIVEL_ESTUDIOS: \n\n",data['DS_NIVEL_ESTUDIOS'].value_counts())
print("\nValores: Variable DS_OCUPACION: \n\n",data['DS_OCUPACION'].value_counts())
print("\nValores: Variable DS_ESTADO_CIVIL: \n\n",data['DS_ESTADO_CIVIL'].value_counts())

Name: DS_CLASE, dtype: int64

Valores: Variable DS_PAIS_NACIMIENTO:

      Colombia           842357
      Venezuela          447
      Ecuador            245
      ESPANA              122
      Italia              71
      Alemania             71
      Peru                 69
      Argentina            61
      Estados Unidos de America   61
      Panama               50
      Cuba                 49
      Chile                 44
      Francia               31
      Mexico                23
      -                      21
      Brasil                17
```

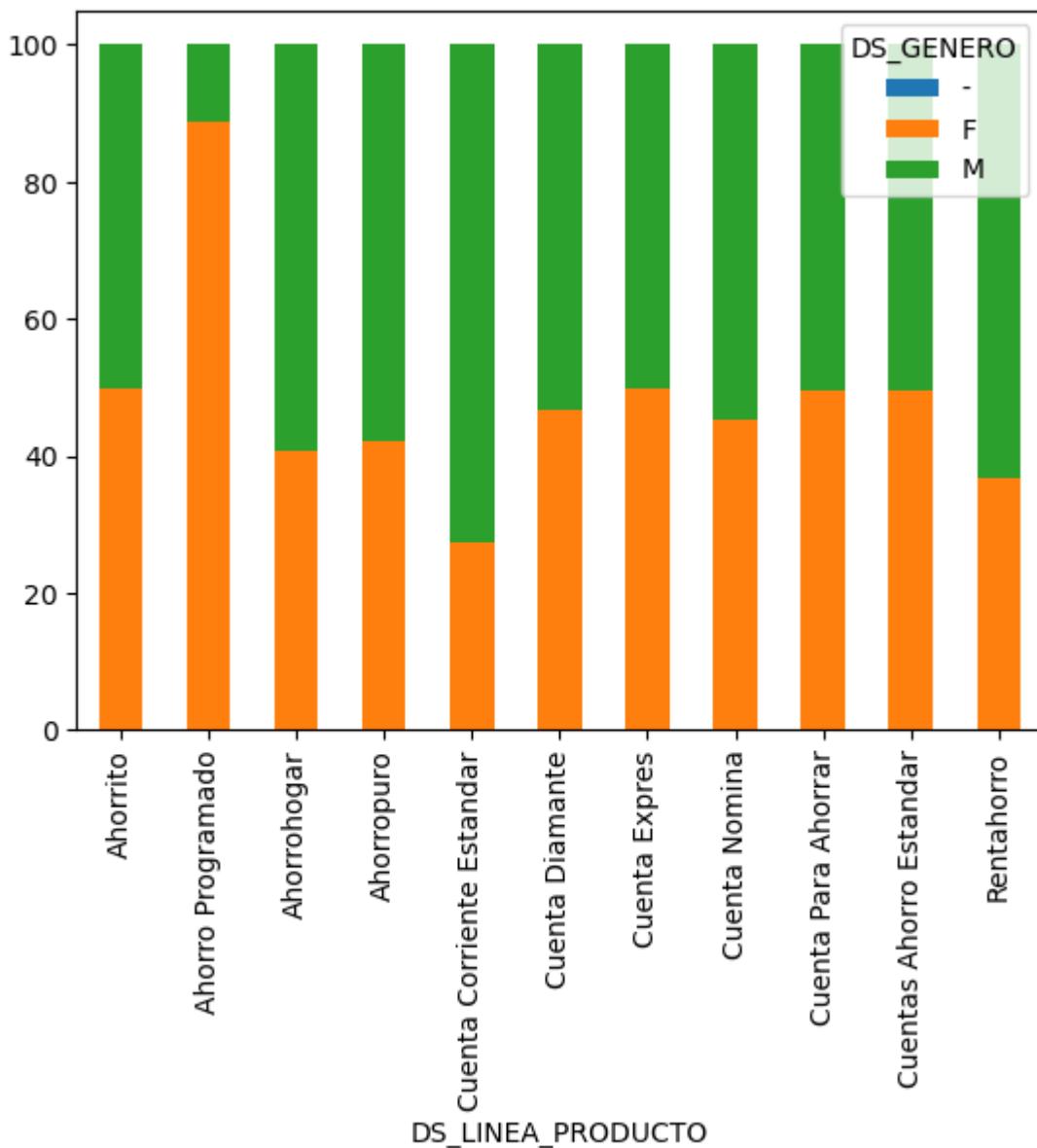
```
In [89]: """Las siguientes tablas de contingencia permiten observar la distribucion de algunos datos con respecto a la Linea de producto
"""

# Tabla de contingencia Linea Producto / Genero (Porcentaje)

# Aunque hay valores en el Genero sin determinar no se reemplazaran

pd.crosstab(index=data['DS_LINEA_PRODUCTO'],
             columns=data['DS_GENERO']).apply(lambda r: r/r.sum() *100, axis=1).round(2)
```

Out[89]: <Axes: xlabel='DS_LINEA_PRODUCTO'>

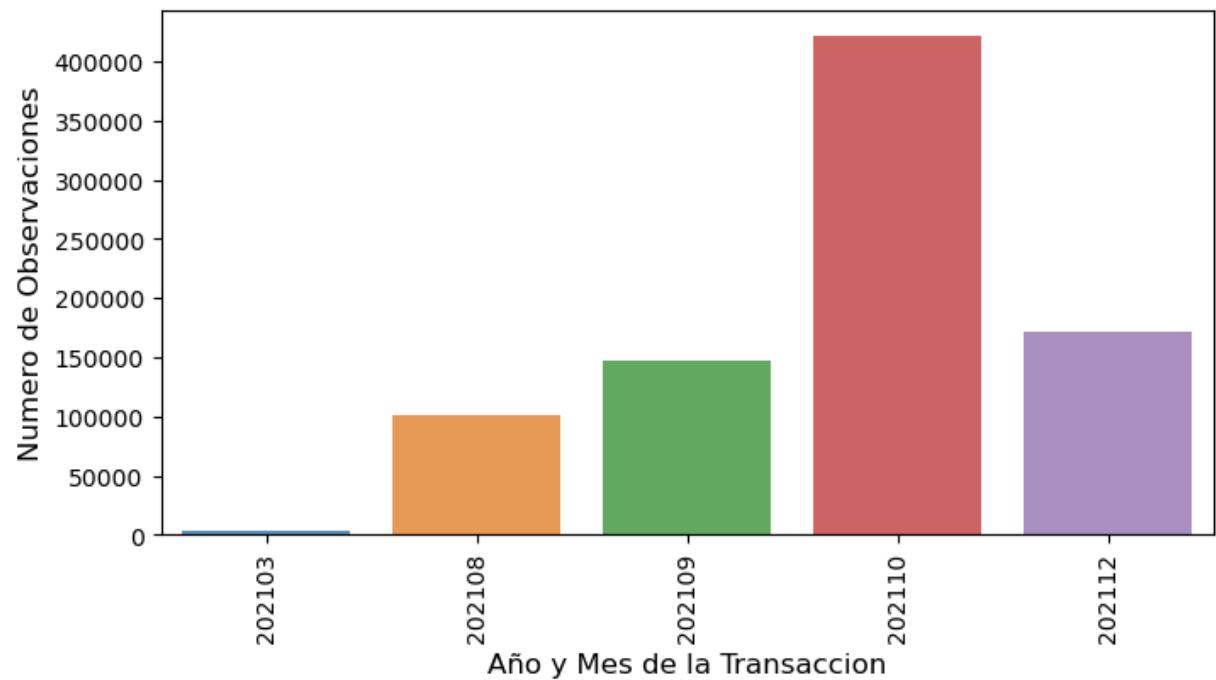


```
In [90]: # Transacciones por mes
```

```
import seaborn as sns

data['fecha_dato_yearmonth'] = data['SK_FE_TRANSACCION'].apply(lambda x: (100*x))
yearmonth = data['fecha_dato_yearmonth'].value_counts()

plt.figure(figsize=(8,4))
sns.barplot(x=yearmonth.index, y=yearmonth.values, alpha=0.8)
plt.xlabel('Año y Mes de la Transaccion', fontsize=12)
plt.ylabel('Numero de Observaciones', fontsize=12)
plt.xticks(rotation='vertical')
plt.show()
```



In [16]: #Funcion para definir categoria segun la Edad

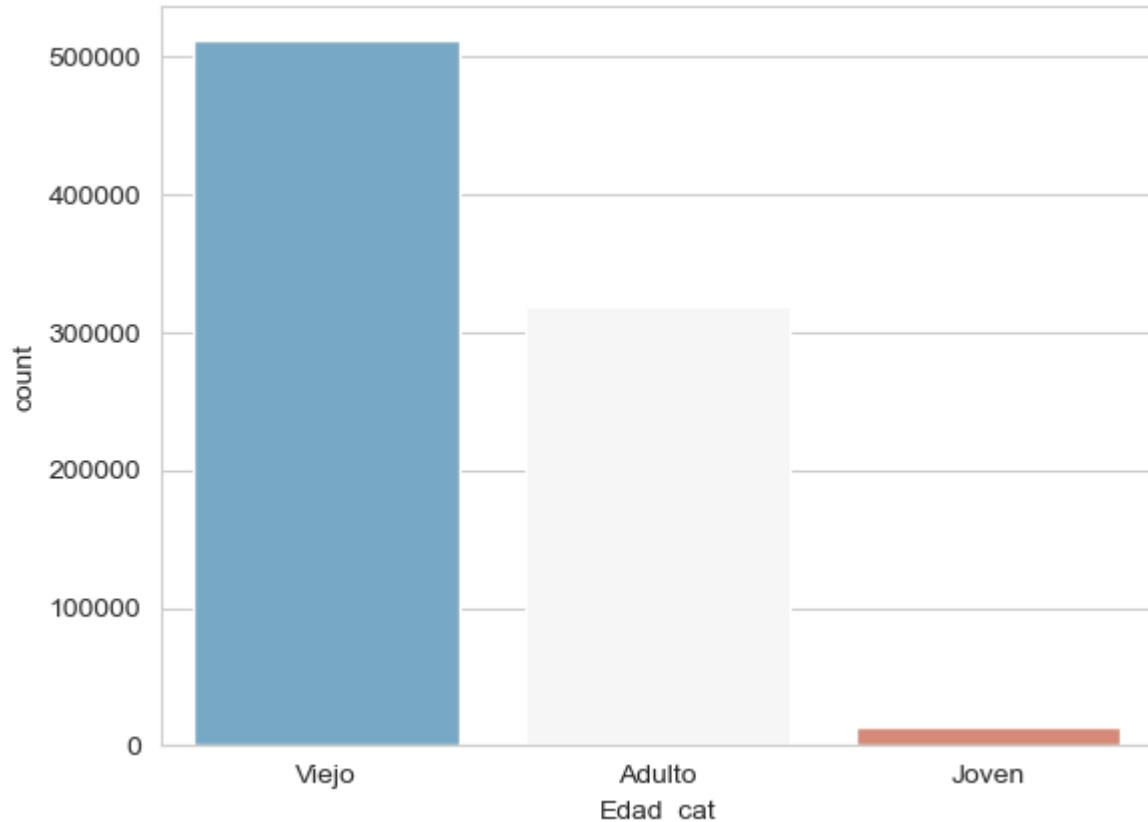
```
def age_cat(x):
    if int(x) <24:
        return('Joven')
    elif int(x) <55:
        return('Adulto')
    else:
        return('Viejo')

data=data.dropna()
data[ 'Edad_cat']=data[ 'EDAD_ANIOS'].apply(lambda x:age_cat(x))
```

In [17]: #Visualizacion categoria por Edad

```
sns.set_style('whitegrid')
sns.countplot(x='Edad_cat',data=data,palette='RdBu_r')
```

Out[17]: <Axes: xlabel='Edad_cat', ylabel='count'>

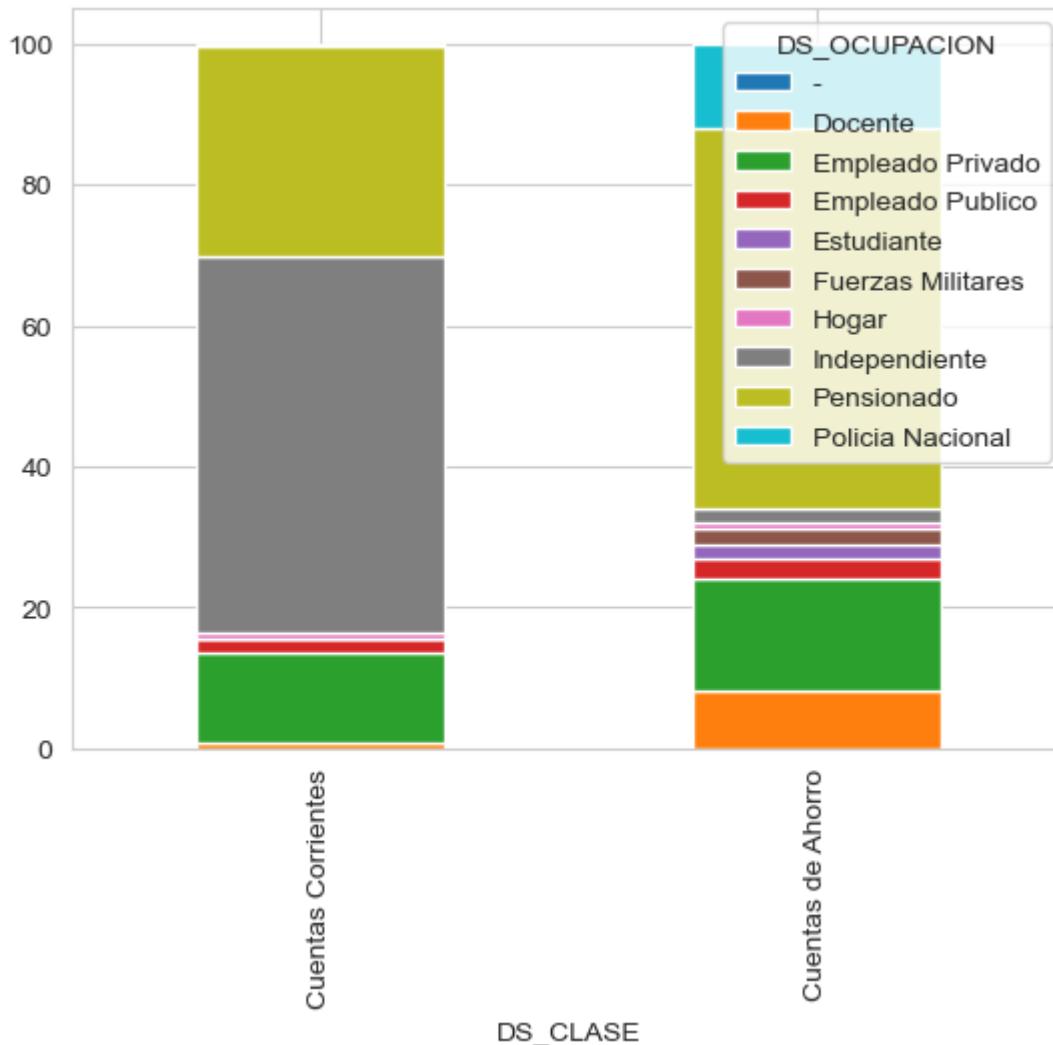


In [93]:

```
# Visualizacion DS_CLASE / DS_OCUPACION (Porcentaje)

pd.crosstab(index=data['DS_CLASE'],
             columns=data['DS_OCUPACION']).apply(lambda r: r/r.sum() *100, axis=1)
```

Out[93]: <Axes: xlabel='DS_CLASE'>

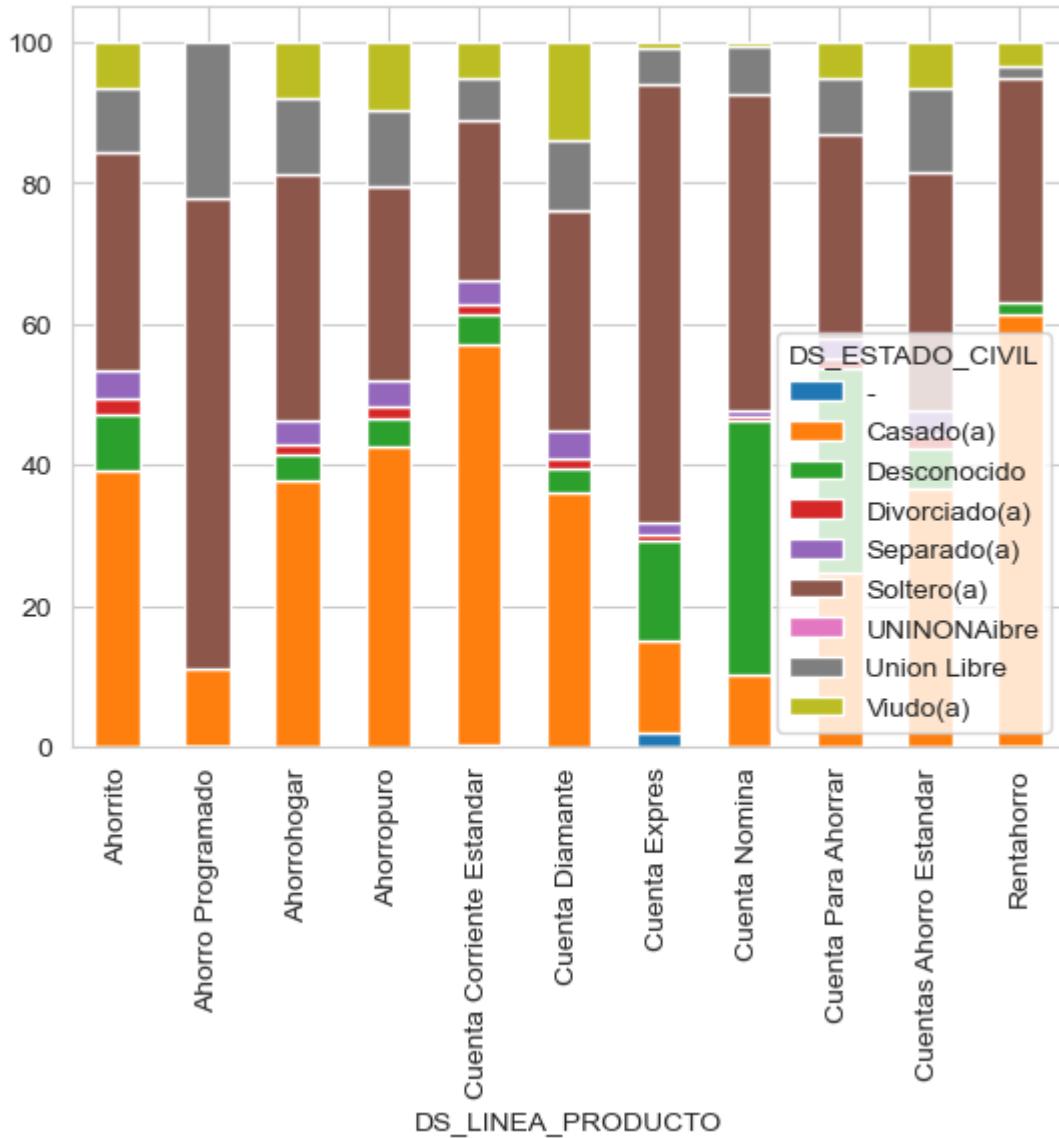


In [94]:

```
# Visualizacion DS_LINEA_PRODUCTO / DS_ESTADO_CIVIL (Porcentaje)

pd.crosstab(index=data['DS_LINEA_PRODUCTO'],
             columns=data['DS_ESTADO_CIVIL']).apply(lambda r: r/r.sum() *100, axis=1)
```

Out[94]: <Axes: xlabel='DS_LINEA_PRODUCTO'>



In [95]: *#identificamos las nuevas variables*

```
data.columns
```

Out[95]: Index(['SK_CLIENTE', 'NO_PRODUCTO', 'SK_TRANSACCION', 'SK_FE_TRANSACCION',
 'SK_PRODUCTO_SERVICIO', 'DS_LINEA_PRODUCTO', 'DS_CLASE', 'VALOR_MVTO',
 'CANTIDAD_TRANSACCION', 'DS_SECTOR_PIB', 'DS_TIPO_EMPRESA',
 'DS_TIPO_PERSONA', 'DS_GENERO', 'FE_VINCULACION_CLIENTE',
 'DS_ESTADO_CIVIL', 'FE_NACIMIENTO', 'DS_PAIS_NACIMIENTO',
 'DS_NIVEL_ESTUDIOS', 'DS_PROFESION', 'DS_OCUPACION', 'DS_TIPO_VIVIENDA',
 'ID_CLIENTE', 'NO_PERSONAS_A_CARGO', 'FE_APERTURA', 'DK_PERSONA',
 'SK_RC', 'EDAD_ANIOS', 'ANTIGUEDAD_ANIOS', 'APERTURA_ANIOS',
 'Mes_Transact', 'PROM', 'fecha_dato_yearmonth', 'Edad_cat'],
 dtype='object')

RECOMENDACIÓN

El objetivo de este primer segmento es implementar un sistema de recomendación más eficaz, permitiendo la satisfacción de necesidades individuales de todos los clientes. Para ello, se construye una matriz usuario-ítem (SK_CLIENTE, PRODUCTO) que contiene el identificador del cliente y los productos que poseían a partir del 01 de Marzo.

Los siguientes modelos son técnicas relativamente simples pero poderosas utilizadas por muchas empresas comerciales minoristas

Se calcula la recomendación con tres modelos diferentes: Modelo basado en el producto MAS POPULAR, FILTADO COLABORATIVO Basado en memoria y FILTRADO COLABORATIVO basado en modelos. Finalmente se integran los tres modelos en una recomendación hibrida ponderada, utilizando métricas de precisión promedial.

SISTEMA DE RECOMENDACION CON EL PRODUCTO MAS POPULAR:

Modelo basado en la popularidad para recomendar al cliente el producto bancario más vendido. La razón de ser de este método es que los clientes, especialmente los nuevos, tienden a obtener los productos que más compran otros clientes. El modelo basado en la popularidad se aplica ampliamente en la industria minorista. En la industria bancaria, aunque la situación es un poco diferente debido a una menor selección de productos, un modelo de negocios más simple y más regulación, este modelo sigue siendo atractivo, especialmente un banco que tiene diversos productos de tarjetas de crédito, préstamos y depósitos que apuntan a diferentes segmentos de clientes.

In [29]: #Se extraen las columnas de identificación y Línea producto para realizar el Modelo

```
name_col = ['SK_CLIENTE', 'DS_LINEA_PRODUCTO']

data_pr=pd.read_csv(file_path, sep=";", index_col=False, encoding='latin1', usecols=name_col)

# Elimina Duplicados
data_pr=data_pr.drop_duplicates()

missing_values= data_pr.isnull().sum()
total_cells = np.product(data_pr.shape)
total_missing= missing_values.sum()
# percent of data that is missing
(total_missing/total_cells) * 100
```

Out[29]: 0.0

In [30]: data_pr.info

```
Out[30]: <bound method DataFrame.info of
          SK_CLIENTE           DS_LINEA_PRODUCTO
0      2582480  Cuenta Corriente Estandar
12     1563136                  Ahorrohogar
13     3483262                  Ahorrohogar
15     1802756                  Ahorrohogar
16     2988470                  Ahorrohogar
...
892389    3162352                  Ahorrohogar
892391    1352148  Cuenta Corriente Estandar
892392    4064478                  Cuenta Nomina
892399    3537381                  Ahorrohogar
892400    1528359                  Ahorrohogar
[380581 rows x 2 columns]>
```

In [31]: #Pivotea la extracción para obtener una matriz de Identificador con cada uno de los productos

```
productos=pd.pivot_table(data_pr,
                         index=['SK_CLIENTE'],
                         columns=['DS_LINEA_PRODUCTO'],
                         aggfunc=['size'],
                         fill_value=0)
```

In [32]: # Como la data nos da información detallada por día, se eliminan duplicados

```
productos.columns = productos.columns.droplevel()
productos=productos.drop(columns = '-')
productos=productos.reset_index()
productos.columns.name = None
```

In [33]: productos

Out[33]:

	SK_CLIENTE	Ahorrito	Ahorro Programado	Ahorrohogar	Ahorropuro	Cuenta Corriente Estandar	Cuenta Diamante	Cuent Expre
0	-1	0	0	1	1	1	1	1
1	61862	0	0	1	0	0	0	0
2	63954	0	0	0	1	0	0	0
3	68937	0	0	1	0	0	0	1
4	69976	0	0	0	1	0	0	0
...
326283	4559998	0	0	0	0	0	0	1
326284	4560003	0	0	0	1	0	0	0
326285	4560008	0	0	0	0	0	0	1
326286	4560012	0	0	0	0	0	0	1
326287	4560024	0	0	0	0	0	0	1

326288 rows × 12 columns

In [34]: #Se extraen las columnas de identificación y Línea producto para solo Octubre
#(Ejercicio de recomendación Popular con ponderación)

```

name_col_oct = ['SK_CLIENTE', 'DS_LINEA_PRODUCTO', 'SK_FE_TRANSACCION']

data_pr_oct=pd.read_csv(file_path, sep=";", index_col=False, encoding='latin1', u
data_pr_oct['SK_FE_TRANSACCION']= pd.to_datetime(data_pr_oct.SK_FE_TRANSACCION, t
data_pr_oct['PERIODO'] = data_pr_oct['SK_FE_TRANSACCION'].apply(lambda x: (100*x
data_pr_oct = data_pr_oct[data_pr_oct['PERIODO'] == 202112]

data_pr_oct.drop(['PERIODO'], axis = 1, inplace = True)

data_pr_oct=data_pr_oct.drop_duplicates()

data_pr_oct=pd.pivot_table(data_pr_oct,
                           index=['SK_CLIENTE', 'SK_FE_TRANSACCION'],
                           columns=['DS_LINEA_PRODUCTO'],
                           aggfunc=['size'],
                           fill_value=0)

data_pr_oct.columns = data_pr_oct.columns.droplevel()
data_pr_oct=data_pr_oct.reset_index()
data_pr_oct.columns.name = None

data_pr_oct.drop(['SK_CLIENTE'], axis = 1, inplace = True)

```

In [35]:

```
"""
    Define una funcion para el calculo de probabilidad de ocurrencia de un producto

    Se realiza un recorrido por cada producto, contando la cantidad de clientes que lo compraron
    Se agrega el valor a un diccionario y se recorre para calcular el porcentaje de compra
    Se organiza por el producto con mayor cantidad de adquisiciones
"""

def mas_popular(data):

    top_pr = {}
    for c in data.columns[1:]:
        top_pr[c] = data[c].value_counts()[1]

    # Ordenado por el más popular
    top_pr = dict(sorted(top_pr.items(), key=lambda it: it[1], reverse=True))

    for k, v in top_pr.items():
        top_pr[k] = np.around(v / data.shape[0], decimals=4)

    return top_pr
```

In [36]: """ A partir de esta recomendación, se puede concluir que los 3 productos más vendidos son Ahorrohogar, Cuenta Diamante y Ahorropuro. Teniendo en cuenta que el análisis se realizó para la fecha de hoy, es difícil decir si estos son los más vendidos actualmente. Como el historial está dado para una fecha de 4 meses del año 2021, nos resulta difícil decir si estos son los más vendidos en ese año, sin embargo, es posible que otros factores como cambios en la situación financiera u otras razones hayan influido en los resultados. Un método para resolver esto y que nos resulte útil es seleccionar los 3 productos más recientes como base de recomendación, eso sí perdiendo muchos registros de compra. Otra forma es utilizar la función Tiempo de caída para disminuir arbitrariamente la influencia de los productos en el momento en que ocurrieron. El ajuste de caída del tiempo es una técnica utilizada en los sistemas de recomendación para ponderar las interacciones recientes y reducir la influencia de las interacciones más antiguas. La popularidad de los elementos puede cambiar con el tiempo y, por lo tanto, las interacciones más recientes suelen ser más indicativas de los intereses actuales de un usuario. Este método se aplica comúnmente en industrias como las noticias y los medios, donde la gente siempre desea ver las noticias más populares y las más recientes.

```
"""
mas_popular(productos)
```

Out[36]: {
 'Ahorrohogar': 0.4851,
 'Cuenta Diamante': 0.3671,
 'Ahorropuro': 0.191,
 'Cuentas Ahorro Estandar': 0.0387,
 'Cuenta Nomina': 0.0379,
 'Cuenta Corriente Estandar': 0.0175,
 'Cuenta Para Ahorrar': 0.0146,
 'Rentahorro': 0.0066,
 'Ahorrito': 0.0047,
 'Cuenta Expres': 0.0032,
 'Ahorro Programado': 0.0}

In [37]: # Implementación 2 Más Popular

```
# Define la popularidad con la cantidad de unidades adquiridas por Producto (Venta)
pop_df_2 = pd.DataFrame(columns = ['producto','Volumen_Ventas','Frecuencia_de_Venta'])

for i in productos.columns[1:]:
    pop_df_2 = pop_df_2.append({'producto':i, 'Volumen_Ventas':productos[i].value_counts().sum(), 'Frecuencia_de_Venta':len(productos[i])}, ignore_index=True)

pop_df_2.sort_values('Frecuencia_de_Venta', inplace = True, ascending = False)
pop_df_2.reset_index(inplace = True)
```

In [38]: *#Imprime la Lista de recomendaciones de productos*

```
pop_df_2[['producto','Frecuencia_de_Venta']]
```

Out[38]:

	producto	Frecuencia_de_Venta
0	Ahorrohogar	0.49
1	Cuenta Diamante	0.37
2	Ahorropuro	0.19
3	Cuenta Nomina	0.04
4	Cuentas Ahorro Estandar	0.04
5	Cuenta Corriente Estandar	0.02
6	Cuenta Para Ahorrar	0.01
7	Rentahorro	0.01
8	Ahorrito	0.00
9	Ahorro Programado	0.00
10	Cuenta Expres	0.00

Recomendacion más popular con tiempo de caida

In [39]:

```

from math import exp

columna = ['SK_FE_TRANSACCION']

data_pr_oct = data_pr_oct.groupby(['SK_FE_TRANSACCION'], as_index = False).sum()

#Unpivot del dataframe - para obtener en una columna Los productos y el cálculo de los días transcurridos entre la fecha de transaccion y la fecha mas reciente.

data_pr_oct = pd.melt(data_pr_oct, id_vars=columna, var_name='producto', value_name='cantidad_producto')

#Dias transcurridos entre la fecha de transaccion y la fecha mas reciente.

data_pr_oct['tm_transcurrido'] = (data_pr_oct['SK_FE_TRANSACCION'].max() - data_pr_oct['FECHA']).dt.days

# Funcion para calcular el tiempo de caida

def tiempo_caida(t, rate):
    return exp(-rate * t)

#Se estable la tasa de caída y se generan recomendaciones basadas en popularidad
rate = 0.001

#Calcula el peso de caida de tiempo para cada Registro

data_pr_oct['peso'] = data_pr_oct['tm_transcurrido'].apply(lambda x: tiempo_caida(x, rate))

# Se adiciona variable con el valor ponderado de cada fila calculado con el peso

data_pr_oct['valor_Ponderado'] = data_pr_oct['cantidad_producto'] * data_pr_oct['peso']

# Calcular por producto la suma de valores ponderados

Puntajes_ponderados = data_pr_oct.groupby('producto',as_index = False)[['valor_Ponderado']].sum()

#Se organizan por valor ponderado de forma descendente
recomendacion = Puntajes_ponderados.sort_values('valor_Ponderado',ascending=False)
recomendacion

```

Out[39]:

	producto	valor_Ponderado
1	Ahorrohogar	79045.048128
2	Ahorropuro	21776.000000
3	Cuenta Corriente Estandar	2670.997002
5	Rentahorro	1865.824979
4	Cuenta Expres	148.000000
0	Ahorro Programado	2.000000

De los resultados de las recomendaciones anteriores: Ahorrohogar,Ahorropuro,Cuenta Corriente Estandar. Tiene una variación en uno de los productos lo que explica la razón por la cual los

análisis deben ser más detallados en razón del tiempo.

En resumen, el recomendador basado en la popularidad es bastante útil en los escenarios generales de recomendación de productos, ya que los clientes, pueden saber qué productos bancarios son las mejores opciones para los demás y hacer sus selecciones.

Sin embargo, para los clientes que ya participan en transacciones con el banco y tienen sus propias preferencias bancarias en la compra de productos, no basta con recomendarles los productos y servicios más vendidos en general. Por lo tanto, sería necesaria una recomendación personalizada para satisfacer las demandas personalizadas de los clientes. Los modelos que se aplicarán a continuación utilizan algoritmos para conocer las preferencias de los clientes individuales y recomendarles los productos que tienen mayores posibilidades de comprar.

Filtro Colaborativo - Basado en Memoria

Es una técnica que ayuda a filtrar elementos para un usuario de forma colaborativa en función de las preferencias de usuarios similares.

- Filtrado colaborativo basado en el usuario: técnica de recomendación que aprovecha las similitudes entre los usuarios para realizar recomendaciones personalizadas. Se basa en la idea de que los usuarios que tienen preferencias similares en el pasado probablemente tendrán preferencias similares en el futuro.

La similitud entre usuarios se calcula mediante diversas técnicas como la similitud del coseno, la correlación de Pearson o el coeficiente de Jaccard.

Este modelo se basa en las calificaciones dadas por los usuarios. Para los datos trabajados, esta calificación está dada sobre la propiedad de cada producto (entre 0 y 1).

La recomendación se basa en la similitud entre los usuarios, calculada entre la función de medida de similitud, utilizando la distancia del coseno para crear la matriz de similitud entre usuario y elemento.

In [40]:

```
"""
Se limita a 10 mil por capacidad de la memoria.
En el caso real, se espera que se utilicen más recursos computacionales (por ejem-
La matriz de similitud del modelo basado en usuarios tiene los 10000 clientes en
denotan sus similitudes en función de sus comportamientos de tenencia de producto
los 4 meses de estudio del 2021.
El modelo calculará el volumen estimado de compra de productos, para cada cliente
recomendarles los productos con los volúmenes de compra estimados más altos (es m

data_fm = productos.copy()

data_fm = data_fm[:10000]
data_fm = data_fm.set_index('SK_CLIENTE')
data_fm.shape
```

Out[40]: (10000, 11)

In [41]:

```
"""
    La distancia por pares proporciona la distancia entre dos vectores/matrices.
    cuanto mayor sea la distancia por pares, menor será la similitud, mientras que
    coseno_similaridad = (1 - pares_distancia), por lo que cuanto mayor sea la similitud
    entre dos vectores/matrices.

"""

from sklearn.metrics.pairwise import pairwise_distances

# crea la matriz de similitud Ussuario Elemento - Quita los nombres de indice

Cos_Similitud = 1 - pairwise_distances(data_fm, metric="cosine")
```

In [42]:

```

"""
Función que permite calcular las recomendaciones para un usuario.
Utiliza la similitud del coseno para calcular los usuarios más similares.
Devuelve la probabilidad de los productos para un usuario determinado en función
El rango de probabilidad es <0, 1>.

"""

def user_reco(sk_cliente, data, matriz_s = Cos_Similitud):

    # Calcula el índice en la matriz de similitud usuario-elemento para el sk_cliente
    cal_ind = list(data.index).index(sk_cliente)

    # Usuario Similares
    k = 0
    sim_min = 0.79
    user_sim_k = {}

    while k < 20:
        # Crea un Diccionario {'Usuario Similar': 'Similitud'}
        for user in range(len(data)):

            # 0.99 ya que no se desea traer el mismo user ID
            if sim_min < matriz_s[cal_ind, user] < 0.99:
                user_sim_k[user] = matriz_s[cal_ind, user]
                k+=1

        sim_min -= 0.025

        # Si no hay usuarios con al menos 0,65 de similitud, La probabilidad de recomendar es 0
        if sim_min < 0.65:
            break

    # Se ordenan los usuarios mas similares
    user_sim_k = dict(sorted(user_sim_k.items(), key=lambda item: item[1], reverse=True))
    user_id_k = list(user_sim_k.keys())

    # dataframe con los usuarios mas similares (Transpone el indice a los productos)
    df_user_k = data.iloc[user_id_k]
    df_user_k_T = df_user_k.T

    # cambia el indice del usuario al indice del coseno
    df_user_k_T.columns = user_id_k

    # promedio de los usuarios similares
    ownership = []
    usuarios = {}

    for nombre, reg in df_user_k_T.iterrows():

        for indx, own in reg.items():

            ownership.append(own)

```

```

    usuarios[nombre] = np.mean(ownership)
    ownership = []

    # De no tener usuarios con similitud de al menos 0,65, la probabilidad de recomendar es 0
    if pd.isna(list(usuarios.values())[0]) == True:
        usuarios = {key : 0 for (key, value) in usuarios.items()}

    return usuarios

```

In [43]: *#Validacion*

```
user_reco(69976, data_fm)
```

Out[43]:

```
{'Ahorrito': 0.09375,
 'Ahorro Programado': 0.0,
 'Ahorrohogar': 0.46875,
 'Ahorropuro': 1.0,
 'Cuenta Corriente Estandar': 0.0625,
 'Cuenta Diamante': 0.34375,
 'Cuenta Expres': 0.0,
 'Cuenta Nomina': 0.0,
 'Cuenta Para Ahorrar': 0.03125,
 'Cuentas Ahorro Estandar': 0.0,
 'Rentahorro': 0.0}
```

In [44]: *# Implementación 2 - Modelo recomendacion por similitud de Usuarios*

```

# Matriz de relacion de clientes x productos

from sklearn.metrics.pairwise import cosine_similarity

df_cf = productos.groupby("SK_CLIENTE").sum()
df_cf = df_cf.fillna(0)
df_cf.index = df_cf.index.astype('string')

df_cf_user = df_cf.copy()
df_cf_user = df_cf_user[:10000]

df_cf_user = pd.DataFrame(cosine_similarity(df_cf_user), index = df_cf_user.index)
df_cf_user.shape

```

Out[44]: (10000, 10000)

In [45]: df_cf_user

Out[45]:	SK_CLIENTE	-1	61862	63954	68937	69976	73971	80196	80399	8047
	SK_CLIENTE									
	-1	1.000000	0.377964	0.377964	0.534522	0.377964	0.377964	0.377964	0.377964	0.37
	61862	0.377964	1.000000	0.000000	0.707107	0.000000	1.000000	1.000000	1.000000	0.00
	63954	0.377964	0.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.00
	68937	0.534522	0.707107	0.000000	1.000000	0.000000	0.707107	0.707107	0.707107	0.00
	69976	0.377964	0.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.00

	337748	0.534522	0.707107	0.000000	0.500000	0.000000	0.707107	0.707107	0.707107	0.00
	337763	0.377964	0.000000	0.000000	0.707107	0.000000	0.000000	0.000000	0.000000	0.00
	337785	0.534522	0.707107	0.000000	1.000000	0.000000	0.707107	0.707107	0.707107	0.00
	337788	0.377964	1.000000	0.000000	0.707107	0.000000	1.000000	1.000000	1.000000	0.00
	337792	0.377964	0.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	1.00

10000 rows × 10000 columns



In [46]:

```
"""
id es la identificación del cliente para el cual recomendamos el producto, rate es
para que se incluyan otros clientes para calcular el volumen de compra previsto y
los productos que se deben recomendar al cliente.

"""

def sim_usuarios(id, rate, num):

    id = str(id)

    # obtiene la lista de clientes con el Score mínimo de Similitud
    sim_clientes = df_cf_user.loc[(df_cf_user[id] >= rate)&(df_cf_user[id] < 1),id]
    print("Hay {} Clientes con similitud al cliente consultado: ".format(len(sim_clientes)))

    # Productos comprados del cliente objetivo y se obtenga la matriz de usuarios
    productos = df_cf.loc[id,:]
    productos_NO = [i for i in productos.index if productos[i] == 0]

    matriz_nueva = df_cf.loc[sim_clientes.index, productos_NO]
    matriz_nueva['puntaje'] = sim_clientes

    # Calcula el monto de compra previsto del producto bancario en función del peso
    for i in matriz_nueva.columns:
        if i != 'puntaje':
            matriz_nueva[i] = matriz_nueva[i] * matriz_nueva['puntaje']
        else:
            break

    # Agrega las puntuaciones previstas y calcula el promedio en función de la similitud
    # Lista los productos recomendados según los volúmenes de compra promedio por cliente
    Productos_reco = (matriz_nueva.sum()/matriz_nueva.sum()['puntaje'])[:-1].sort_values(ascending=False)

    # Se filtran los productos con una ponderación mayor a 0
    Productos_reco = Productos_reco[Productos_reco > 0]

    # De no haber ningún elemento en la lista no hay recomendación alguna
    if len(Productos_reco) == 0:
        print('No hay recomendaciones para el Cliente')

    return Productos_reco
```

In [47]:

```
""" VALIDACION MODELO SIMILITUD USUARIOS - 2"""

sim_usuarios(id= 63954, rate = 0.70, num = 5)
#sim_usuarios(id= 80488, rate = 0.70, num = 5)
#sim_usuarios(id= 337725, rate = 0.70, num = 5)
#sim_usuarios(id= 337700, rate = 0.70, num = 5)

#Para validar que productos tiene y comparar con respecto a la recomendacion
#prod = df_cf.loc['63954',:]
#prod
```

Hay 32 Clientes con similitud al cliente consultado:

Out[47]:

Ahorrohogar	0.46875
Cuenta Diamante	0.34375
Ahorrito	0.09375
Cuenta Corriente Estandar	0.06250
Cuenta Para Ahorrar	0.03125

dtype: float64

Podemos encontrar que la mayoría de los productos recomendados a los clientes tienen un volumen de compra muy bajo (volumen promedio ponderado), faltan muchos valores en la matriz de elementos de usuario debido a que los usuarios no han comprado ni interactuado con ciertos elementos, lo que dificulta la predicción precisa de puntuaciones o recomendaciones para esos elementos. Aunque la técnica nos arroja recomendaciones de acuerdo a las compras realizadas, resulta costoso mantener el almacenamiento y el cálculo de la matriz de elementos de usuario y de la matriz de similitud de usuarios.

Filtrado colaborativo basado en elementos

El filtrado colaborativo basado en elementos es otro enfoque utilizado en los sistemas de recomendación que se centra en las similitudes entre elementos, las recomendaciones se hacen en función de las similitudes entre los elementos y las preferencias de los usuarios por los elementos se utilizan para determinar las recomendaciones.

La matriz de similitud de artículos se calcula en función de los comportamientos de compra del cliente. Dos productos son similares si fueron comprados y mantenidos en cantidades similares por clientes similares.

In [48]: # Se copian y obtienen 10000 registros para calcular la matriz por elementos (Linea 1)

```

df_sim_elementos = df_cf.copy()
df_sim_elementos = df_sim_elementos[:10000].T

df_sim_elementos = pd.DataFrame(cosine_similarity(df_sim_elementos), index = df_s

```

def reco_elementos(id, rate, num):

```

id = str(id)

# Seleccionar el producto que el cliente 'compró principalmente'
data_tmp = df_cf.T
num_productos = data_tmp[id]
num_productos = num_productos[num_productos > 0].sort_values(ascending = False)

while len(num_productos) == 0:
    print("El Cliente no ha Comprado productos en el periodo Consultado")
    break

# obtiene el subconjunto de productos de la matriz de similitud de productos
mt_sim = df_sim_elementos[num_productos]

# Se excluye el producto que compró el cliente y se ordenan el numero de recomendaciones
lista_recomendacion = mt_sim[(mt_sim < 0.999)&(mt_sim > 0)].sort_values(ascending = True)

while len(lista_recomendacion) == 0:
    print("No hay productos para recomendar")
    break

return lista_recomendacion

```

In [49]: """ VALIDACION MODELO SIMILITUD ELEMENTOS """

```

reco_elementos(id= 68937, rate = 0.75, num = 3)
#sim_usuarios(id= 80488, rate = 0.70, num = 5)
#sim_usuarios(id= 337725, rate = 0.70, num = 5)
#sim_usuarios(id= 337700, rate = 0.70, num = 5)

#Para validar que productos tiene y comparar con respecto a la recomendacion
#prod = df_cf.loc['68937',:]
#prod

```

Out[49]: Cuenta Diamante 0.346125
Cuentas Ahorro Estandar 0.125108
Ahorritto 0.034746
Name: Ahorrohogar, dtype: float64

En comparación con el recomendador basado en el usuario, el modelo basado en artículos proporciona recomendaciones más precisas, ya que se basa en productos similares que el cliente compró antes, en lugar de en las elecciones similares de los clientes, que no pueden reflejar el

cambio rápido de preferencias del cliente objetivo. La recomendación de producto cambiará según los diferentes productos que compró el cliente.

Además, el filtrado colaborativo basado en elementos normalmente funciona mejor en conjuntos de datos más dispersos donde los usuarios han calificado solo una pequeña fracción de los elementos. Dado que los elementos suelen tener más calificaciones que los usuarios, los cálculos de similitud en el filtrado basado en elementos tienden a ser más confiables y precisos.

Filtrado colaborativo Basado en modelos

El filtrado colaborativo basado en modelos es un enfoque para los sistemas de recomendación que implica la creación de un modelo predictivo basado en interacciones usuario-elemento. En lugar de depender únicamente de las similitudes entre usuarios o elementos, los métodos basados en modelos utilizan algoritmos de aprendizaje automático para aprender patrones y hacer predicciones.

Los métodos basados en modelos pueden manejar los problemas de escasez de manera más efectiva. Pueden aprovechar técnicas como la factorización matricial, la reducción de dimensionalidad o modelos de factores latentes para capturar patrones y dependencias subyacentes en los datos. Esto ayuda a superar el desafío de las interacciones limitadas entre el usuario y el elemento, proporcionando mejores recomendaciones incluso cuando los datos son escasos.

La descomposición implica la creación de dos matrices: una matriz de usuario y una matriz de elementos. La matriz de usuarios representa la relación entre los usuarios y los factores latentes, mientras que la matriz de ítems representa la relación entre los ítems y los factores latentes. Los factores latentes son comunes a ambas matrices y capturan la información compartida entre usuarios y elementos.

La idea clave detrás de la factorización matricial es encontrar los valores óptimos para las matrices de usuario y elemento que, cuando se multiplican, se aproximan lo más posible a la matriz original de interacción usuario-elemento. Esto generalmente se logra mediante técnicas de optimización, como el descenso de gradiente o la alternancia de mínimos cuadrados, que tienen como objetivo minimizar el error de reconstrucción entre la matriz original y el producto de las matrices de usuario y de elementos.

```
In [50]: data_modelos = productos.copy()
data_modelos = data_modelos.set_index('SK_CLIENTE')
```

```
In [51]: from collections import defaultdict
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings('ignore')
```

In [52]:

```
"""
Función para calcular las recomendaciones de un usuario determinado.
Utiliza un modelo de aprendizaje automático para calcular la probabilidad de los
"""

def modelos_pr(sk_cliente, data, modelo=DecisionTreeClassifier(max_depth=9)):
    res_mod={}

    for c in data.columns:
        y_train = data[c].astype('int')
        x_train = data.drop([c], axis = 1)
        modelo.fit(x_train, y_train)
        prob_train = modelo.predict_proba(x_train[x_train.index == sk_cliente])[0]

        res_mod[c] = prob_train[0]

    return res_mod
```

In [53]: modelos_pr(68937, data_modelos)

```
Out[53]: {'Ahorrito': 0.0007102598176236081,
'Ahorro Programado': 0.0,
'Ahorrohogar': 0.36772083062836713,
'Ahorropuro': 0.0035184719778838906,
'Cuenta Corriente Estandar': 0.0007331546268930281,
'Cuenta Diamante': 0.2891321064913025,
'Cuenta Expres': 0.0004812540104500871,
'Cuenta Nomina': 0.0002521432173474534,
'Cuenta Para Ahorrar': 0.0028121999176917096,
'Cuentas Ahorro Estandar': 0.00157952568446113,
'Rentahorro': 0.0}
```

RECOMENDACION HIBRIDA PONDERADA:

combina los sistemas de recomendacion individuales, utilizando ponderaciones específicas.

In [36]:

```
"""
    Funcion que calcula la recomendacion Ponderada para un usuario. se calcula en
"""

def Hibrido_Pond(sk_cliente, data_p, data_u, data_m, f1, f2, f3):

    popular_h = mas_popular(data_p)
    memoria_p = user_reco(sk_cliente, data_u)
    modelos_p = modelos_pr(sk_cliente, data_m)

    hibrido = {}
    for k, v in popular_h.items():
        hibrido[k] = (v * f1) + (memoria_p[k] * f2) + (modelos_p[k] * f3)

    return hibrido
```

```
In [55]: Reco_Hibrida = Hibrido_Pond(68937, data_p = productos, data_u = data_fm, data_m =
```

```
Out[55]: {'Ahorrohogar': 0.5844802076570917,
'Cuenta Diamante': 0.5058330266228256,
'Ahorropuro': 0.13804628466113764,
'Cuentas Ahorro Estandar': 0.061411548087781945,
'Cuenta Nomina': 0.0398463691376702,
'Cuenta Corriente Estandar': 0.05059995532338992,
'Cuenta Para Ahorrar': 0.09133638331275626,
'Rentahorro': 0.0033,
'Ahorrito': 0.023360898287739237,
'Cuenta Expres': 0.001720313502612522,
'Ahorro Programado': 0.0}
```

```
In [56]: """
    Función que filtra productos recomendados excluyendo los que ya tiene
"""

def Recom_final(sk_cliente, data, hybrid_outcome):

    # Productos propios del cliente
    usuario_reg = data[data.index == sk_cliente]
    user_products = list(filter(lambda product: usuario_reg[product].to_numpy()[0] > 0, data))

    # elimina productos que son propios
    recomendar = { key : hybrid_outcome[key] for key in hybrid_outcome if key not in user_products }

    recom_ordenado = dict(sorted(recomendar.items(), key=lambda item: item[1], reverse=True))

    return list(recom_ordenado.keys())
```

```
In [57]: recomendacion = Recom_final(68937, data_modelos, Reco_Hibrida)
```

```
In [58]: data_modelos[data_modelos.index == 68937]
```

```
Out[58]:
```

	Ahorrito	Ahorro Programado	Ahorrohogar	Ahorropuro	Cuenta Corriente Estandar	Cuenta Diamante	Cuenta Expres	Cuenta Nomina
--	----------	-------------------	-------------	------------	---------------------------	-----------------	---------------	---------------

```
SK_CLIENTE
```

68937	0	0	1	0	0	1	0
-------	---	---	---	---	---	---	---

In [59]: recomendacion

Out[59]:

```
['Ahorropuro',
 'Cuenta Para Ahorrar',
 'Cuentas Ahorro Estandar',
 'Cuenta Corriente Estandar',
 'Cuenta Nomina',
 'Ahorrito',
 'Rentahorro',
 'Cuenta Expres',
 'Ahorro Programado']
```

SEGMENTACIÓN CLIENTES

En este parte del estudio se realizará una agrupación no supervisada de datos en los registros del cliente de la entidad. La segmentación de clientes es la práctica de separar a los clientes en grupos que reflejen similitudes entre ellos. Se dividirán en segmentos para optimizar la importancia de cada cliente, caracterizarlos y personalizar los productos de acuerdo con las distintas necesidades y comportamientos.

In [18]:

```
""" Data frame para obtener la caracterizacion de los clientes """
df = pd.DataFrame()
```

In [19]:

```
#Cantidad de transacciones registradas por el cliente
#df['transacciones'] = data.groupby("SK_CLIENTE").size()
df['transacciones'] = data.groupby('SK_CLIENTE')[ 'CANTIDAD_TRANSACCION'].sum()
```

In [20]:

```
#Edad del cliente
df['edad'] = data.groupby('SK_CLIENTE')[ 'EDAD_ANIOS'].max()
```

In [21]:

```
#Antiguedad del cliente en la entidad
df['antiguedad'] = data.groupby('SK_CLIENTE')[ 'ANTIGUEDAD_ANIOS'].max()
```

In [22]:

```
#Tiempo transcurrido en la ultima apertura realizada
df['apertura'] = data.groupby('SK_CLIENTE')[ 'APERTURA_ANIOS'].min()
```

In [23]:

```
#Pais de residencia del cliente
df['pais'] = data.groupby('SK_CLIENTE')[ 'DS_PAIS_NACIMIENTO'].max()
```

In [24]:

```
#Ocupacion
df['ocupacion'] = data.groupby('SK_CLIENTE')[ 'DS_OCUACION'].max()
```

In [25]:

```
# Tipo de Vivienda dle cliente
df['tipo_vivienda'] = data.groupby('SK_CLIENTE')[ 'DS_TIPO_VIVIENDA'].max()
```

In [26]:

```
# Estado Civil del Cliente
df['estado_civil'] = data.groupby('SK_CLIENTE')[ 'DS_ESTADO_CIVIL'].max()
```

```
In [27]: #Nivel de estudio del Cliente  
df['nivel_estudios'] = data.groupby('SK_CLIENTE')['DS_NIVEL_ESTUDIOS'].max()
```

```
In [28]: #Genero del Cliente  
df['genero'] = data.groupby('SK_CLIENTE')['DS_GENERO'].max()
```

```
In [29]: # Mapea el Genero con valores numéricos  
df['genero'] = df['genero'].map({'M': 1, 'F': 2, '-' : 0})
```

```
In [30]: # Numero de personas a cargo del titular  
df['no_personas_a_cargo'] = data.groupby('SK_CLIENTE')['NO_PERSONAS_A_CARGO'].ma
```

```
In [31]: # Valor del monto promedio en Compras  
df['valor_mvto'] = data.groupby('SK_CLIENTE')['VALOR_MVTO'].mean().round(0)
```

```
In [32]: # Valor del saldo promedio del cliente  
df['valor_saldo'] = data.groupby('SK_CLIENTE')['PROM'].mean().round(0)
```

```
In [33]: #La Cantidad de productos que tiene el cliente  
df['cant_productos'] = data.groupby('SK_CLIENTE').NO_PRODUCTO.nunique()
```

```
In [34]: #Categoriza el promedio de compras
```

```
grupo_valor_compras = [0, np.nanpercentile(df['valor_mvto'], 25), np.nanpercentile(df['valor_mvto'], 75), np.nanpercentile(df['valor_mvto'], 95), np.nanpercentile(df['valor_mvto'], 99), df['valor_mvto'].max()]
vl_compras_labels = ['Bajo', 'Ordinario', 'Mediana-Alta', 'Alta']
df['vl_compra_cat'] = pd.cut(df['valor_mvto'], grupo_valor_compras, labels = vl_compras_labels)
```

In [35]: # validar la distribucion para las categorias nivel de estudio y Ocupacion

```
for i in [df['nivel_estudios'], df['ocupacion']]:
    print(i.value_counts())
```

Primaria	157298
Profesional	72994
Tecnico Profesional	35960
Especializacion	28527
Sin estudio	24786
-	1110
Tecnologico	327
Maestria	128
Doctorado	21
Secundaria	15
Name: nivel_estudios, dtype: int64	
Pensionado	180200
Empleado Privado	46539
Policia Nacional	39186
Docente	22845
Empleado Publico	8028
Fuerzas Militares	7943
Independiente	7279
Estudiante	6570
Hogar	2565
-	11
Name: ocupacion, dtype: int64	

In [36]: #Se ajustan valores sobre los registros que no tienen una categoria definida
#(Nivel de estudios - Estado Civil - Tipo Vivienda - Ocupacion)

```
df.nivel_estudios = df.nivel_estudios.replace({"-":'Sin estudio'})
df.estado_civil = df.estado_civil.replace({"-":'Desconocido'})
df.tipo_vivienda = df.tipo_vivienda.replace({"-":'Desconocido'})
df.ocupacion = df.ocupacion.replace({"-":'Hogar'})
df.estado_civil = df.estado_civil.replace({"UNINONAibre":'Union Libre'})
```

In [37]: #Variable que muestra si estado_civil se encuentran en pareja o no, disminuyendo
df["Con_Pareja"] = df["estado_civil"].replace({"Casado(a)":1, "Soltero(a)":0, "Unic"})

In [38]: #Se categoriza el nivel de estudios

```
df["Educacion"] = df["nivel_estudios"].replace({
    "Doctorado": "Postgrado", "Maestria": "Postgrado",
    "Primaria": "Academico", "Secundaria": "Academico",
    "Tecnico Profesional": "Tecnico", "Especializacion": "Tecnico"})
```

In [39]: #Reduce la Categorizacion de la ocupacion del cliente

```
df.ocupacion = df.ocupacion.replace({
    "Policia Nacional": 'Fuerzas Militares', "Empleado Publico": 'Fuerzas Militares',
    "Pensionado": 'Fuerzas Militares', "Empleado Privado": 'Fuerzas Militares'})
```

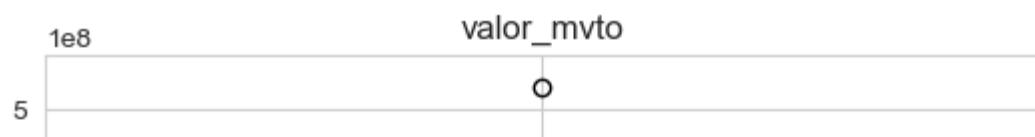
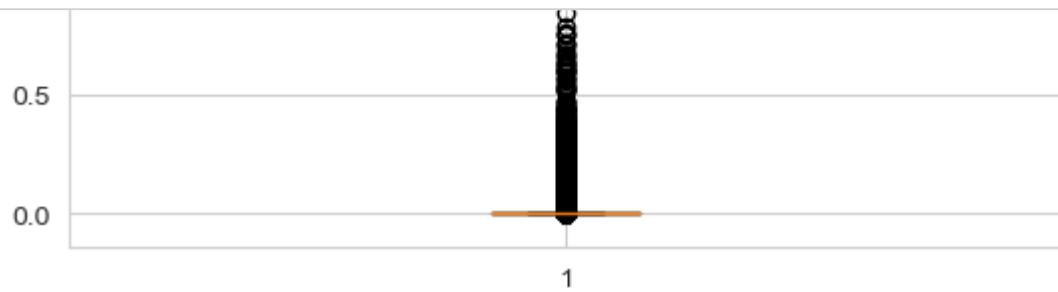
In [121]: df

Out[121]:

	transacciones	edad	antiguedad	apertura	pais	ocupacion	tipo_vivienda	estad
SK_CLIENTE								
61862	2	45.0	24.0	15.0	Colombia	Fuerzas Militares	Propia	Ca
63954	2	87.0	24.0	21.0	Colombia	Pensionado	Familiar	Ca
68937	2	80.0	26.0	26.0	Colombia	Pensionado	Familiar	Ca
69976	1	71.0	27.0	24.0	Colombia	Pensionado	Propia	Se
73971	3	67.0	23.0	23.0	Colombia	Pensionado	Propia	Ca
...
4559998	5	57.0	4.0	4.0	Colombia	Pensionado	Propia	Se
4560003	3	61.0	26.0	25.0	Colombia	Empleado	Propia	Ca
4560008	1	29.0	9.0	9.0	Colombia	Estudiante	Familiar	Se
4560012	3	77.0	21.0	21.0	Colombia	Pensionado	Propia	Ca
4560024	2	46.0	3.0	3.0	Colombia	Empleado	Propia	Se

321166 rows × 17 columns

In [40]: #Visualizamos la caracterizacion de los datos de 'edad', 'valor_saldo', 'valor_mvto', 'antiguedad', 'no_personas_a_cargo'
for i in ['edad', 'valor_saldo', 'valor_mvto', 'antiguedad', 'no_personas_a_cargo']:
 plt.boxplot(df[i])
 plt.title(i)
 plt.show()



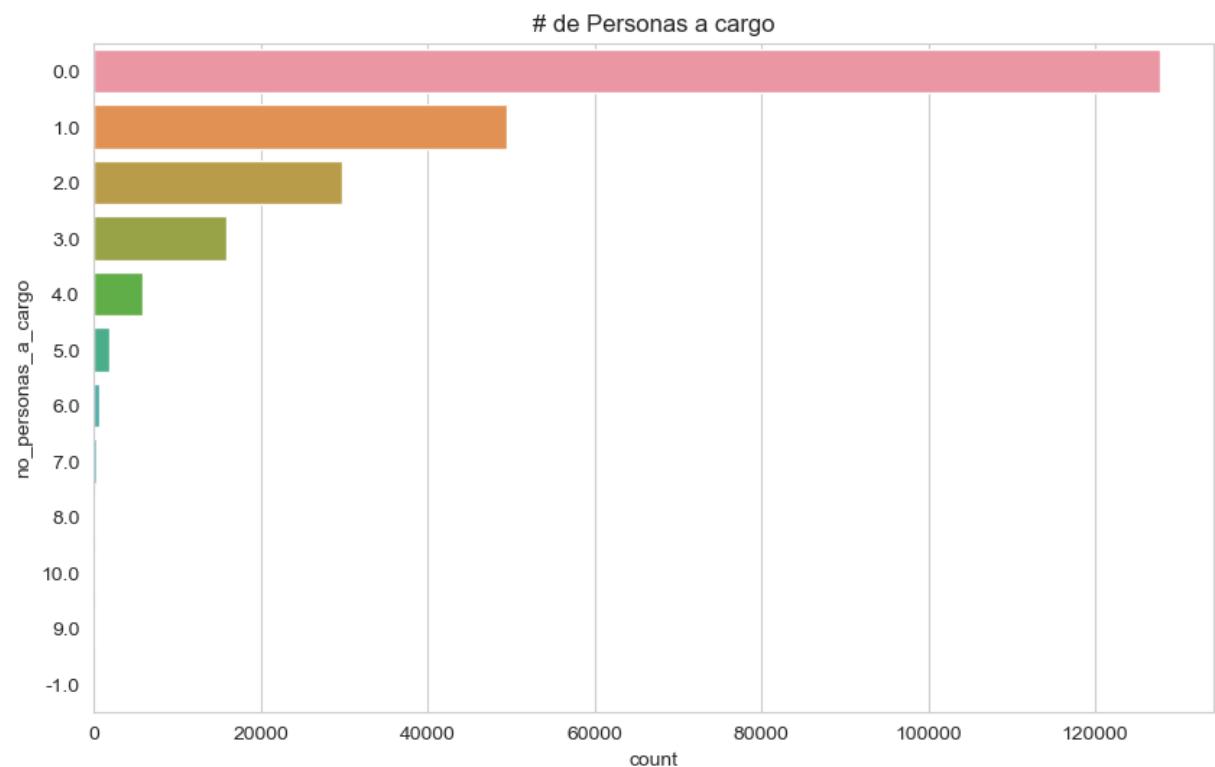
In [41]: #Depuración Outliers

```
#final_df['valor_mvto'].mean()
#final_df['valor_saldo'].mean()

df = df[df['edad'] <=120]
df = df[df['antiguedad'] <=46]
df = df[df['valor_mvto'] <= 857423]
df = df[df['valor_saldo'] <= 2960964]
df = df[df['no_personas_a_cargo'] <= 10]
```

In [42]: plt.figure(figsize=(10,6))

```
sns.countplot(y="no_personas_a_cargo", data=df, order=df["no_personas_a_cargo"].value_counts().index)
plt.title("# de Personas a cargo");
```



In [43]:

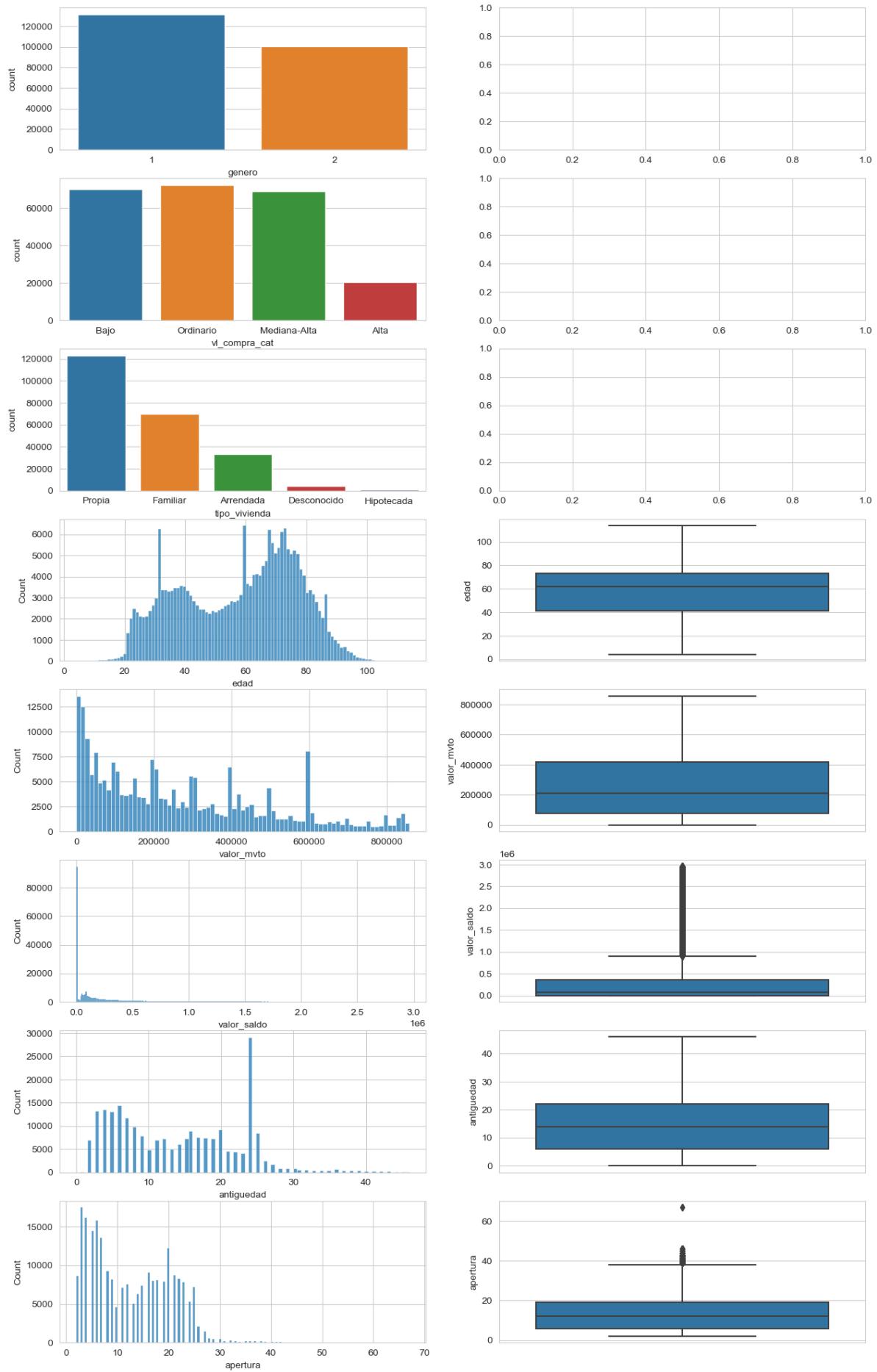
```
"""Nuevo
Visualizacion variables para el dataset de clientes

Se puede evidenciar el Sesgo de la data con una participación mucho mayor del genero
Algunas variables no tienen distribución normal. Lo que genera problemático si queremos
"""

fig, ax = plt.subplots(8, 2, figsize=(15, 25))
columnas_S = ['genero','vl_compra_cat','tipo_vivienda', 'edad', 'valor_mvto', 'valores_mvto']

for idx, column in enumerate(columnas_S):
    if column == 'vl_compra_cat' or column == 'tipo_vivienda' or column == 'genero':
        sns.countplot(data=df, x=column, ax=ax[idx][0])
    else:
        sns.histplot(data=df, x=column, ax=ax[idx][0])

    if column == 'vl_compra_cat' or column == 'tipo_vivienda' or column == 'genero':
        pass
    else:
        sns.boxplot(data=df, y=column, ax=ax[idx][1])
```



In [62]: df

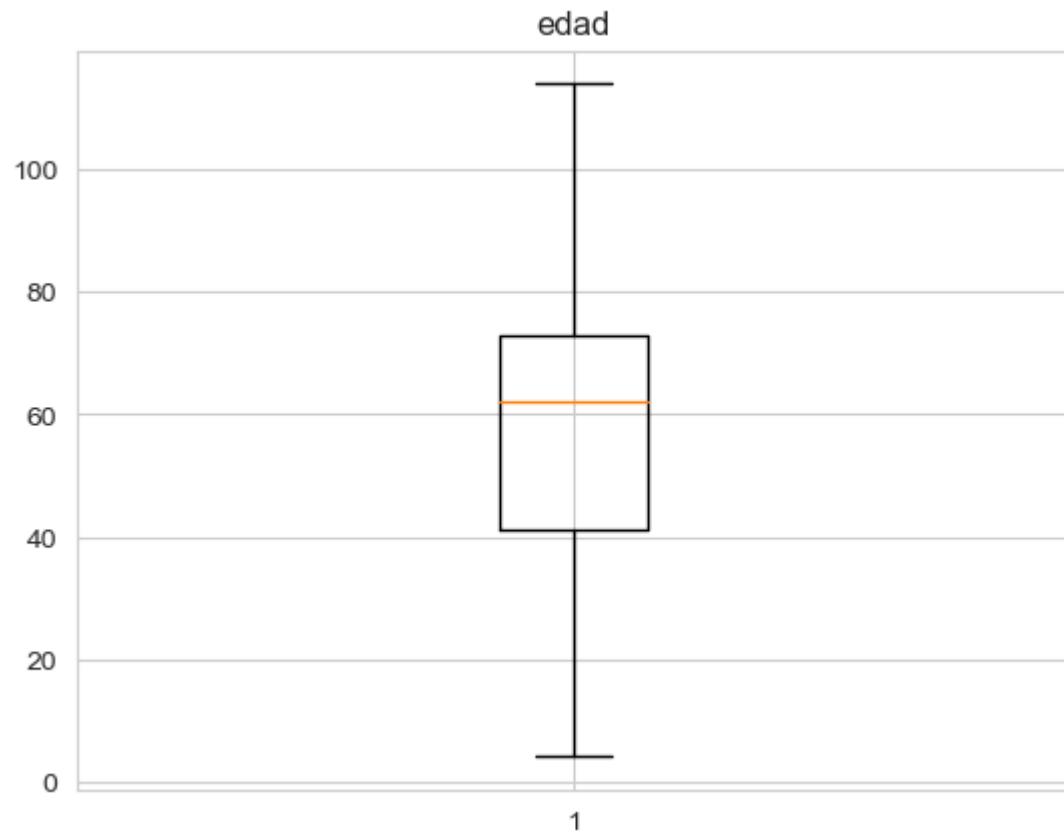
Out[62]:

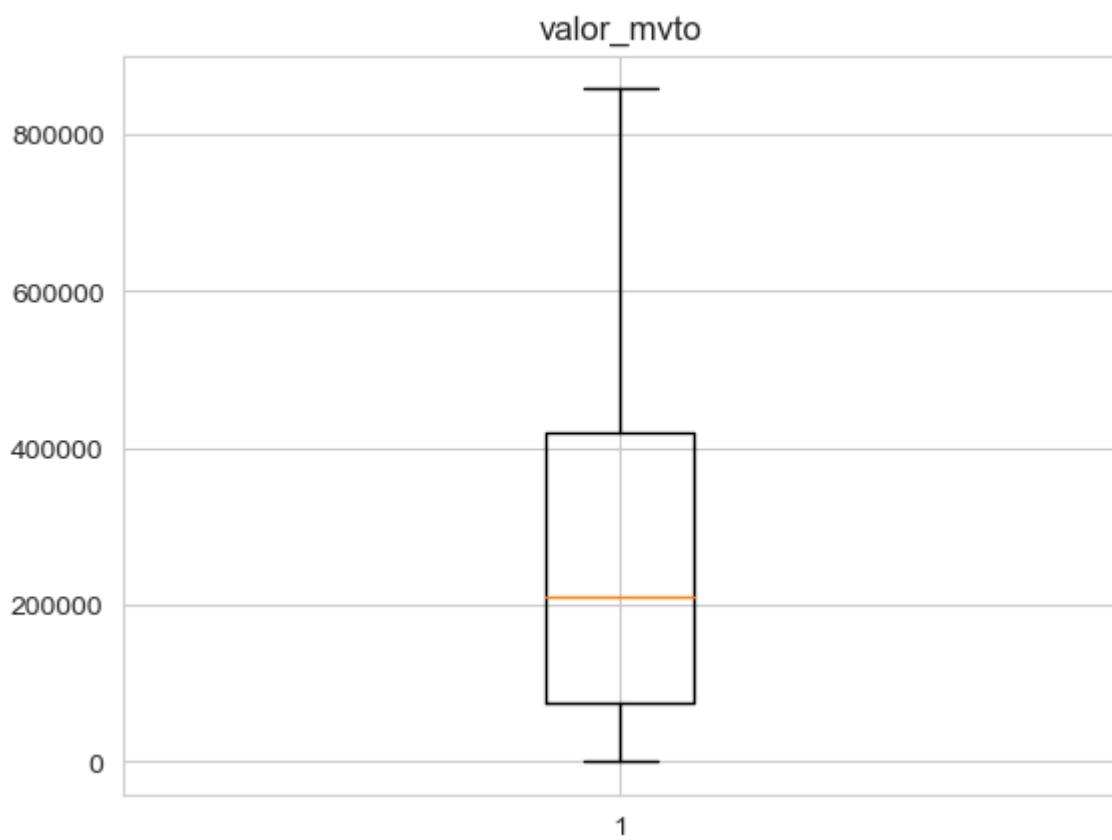
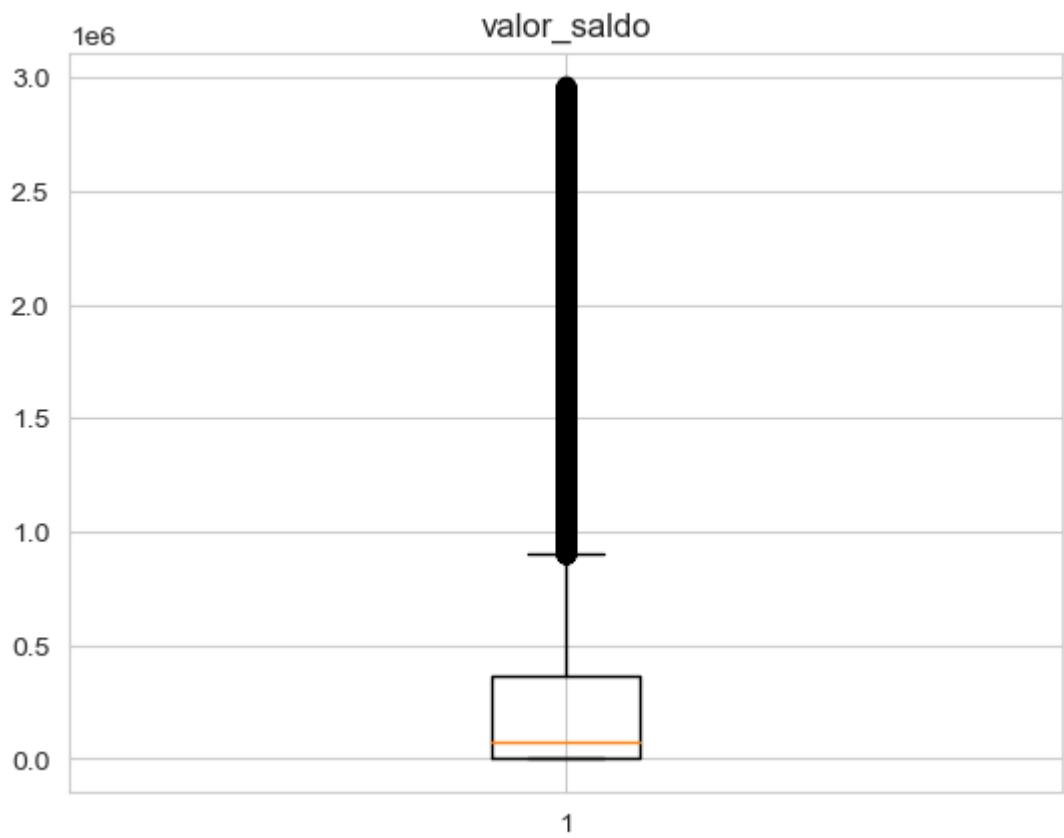
	transacciones	edad	antiguedad	apertura	pais	ocupacion	tipo_vivienda	estad
SK_CLIENTE								
61862	2	45.0	24.0	15.0	Colombia	Fuerzas Militares	Propia	Ca
63954	2	87.0	24.0	21.0	Colombia	Pensionado	Familiar	Ca
68937	2	80.0	26.0	26.0	Colombia	Pensionado	Familiar	Ca
80196	2	38.0	12.0	12.0	Colombia	Fuerzas Militares	Arrendada	Uni
80399	3	43.0	17.0	17.0	Colombia	Docente	Familiar	Ca
...
4559998	5	57.0	4.0	4.0	Colombia	Pensionado	Propia	Sa
4560003	3	61.0	26.0	25.0	Colombia	Empleado	Propia	Ca
4560008	1	29.0	9.0	9.0	Colombia	Estudiante	Familiar	Sa
4560012	3	77.0	21.0	21.0	Colombia	Pensionado	Propia	Ca
4560024	2	46.0	3.0	3.0	Colombia	Empleado	Propia	Sa

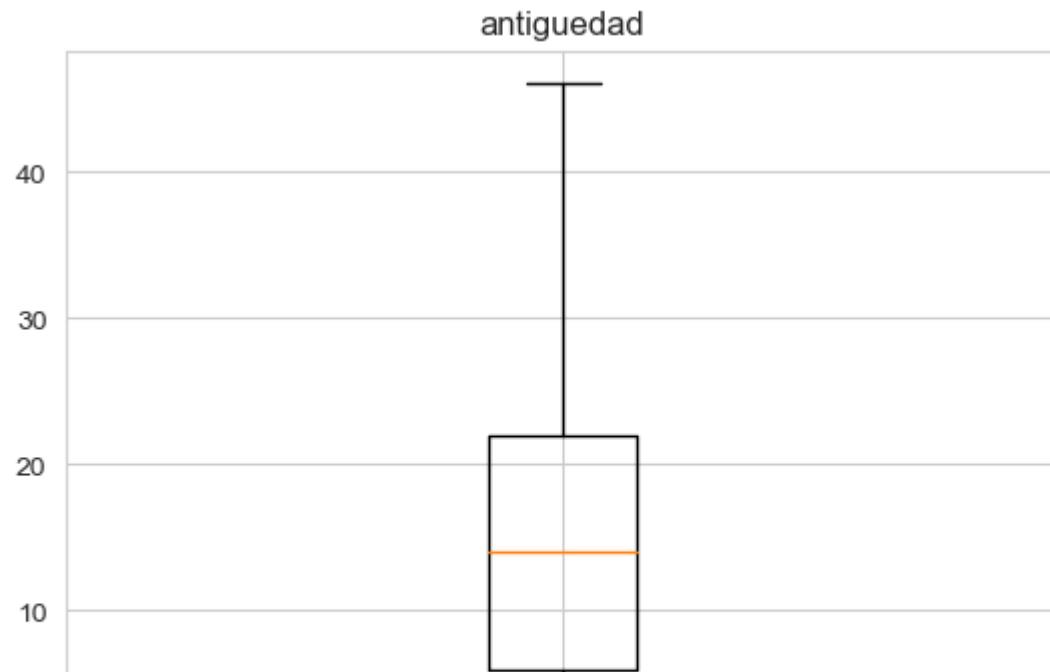
231607 rows × 17 columns



```
In [44]: #Visualizamos la caracterizacion de la 'edad', 'valor_saldo', 'valor_mvto', 'antiguedad'  
for i in ['edad', 'valor_saldo', 'valor_mvto', 'antiguedad']:  
    plt.boxplot(df[i])  
    plt.title(i)  
    plt.show()
```



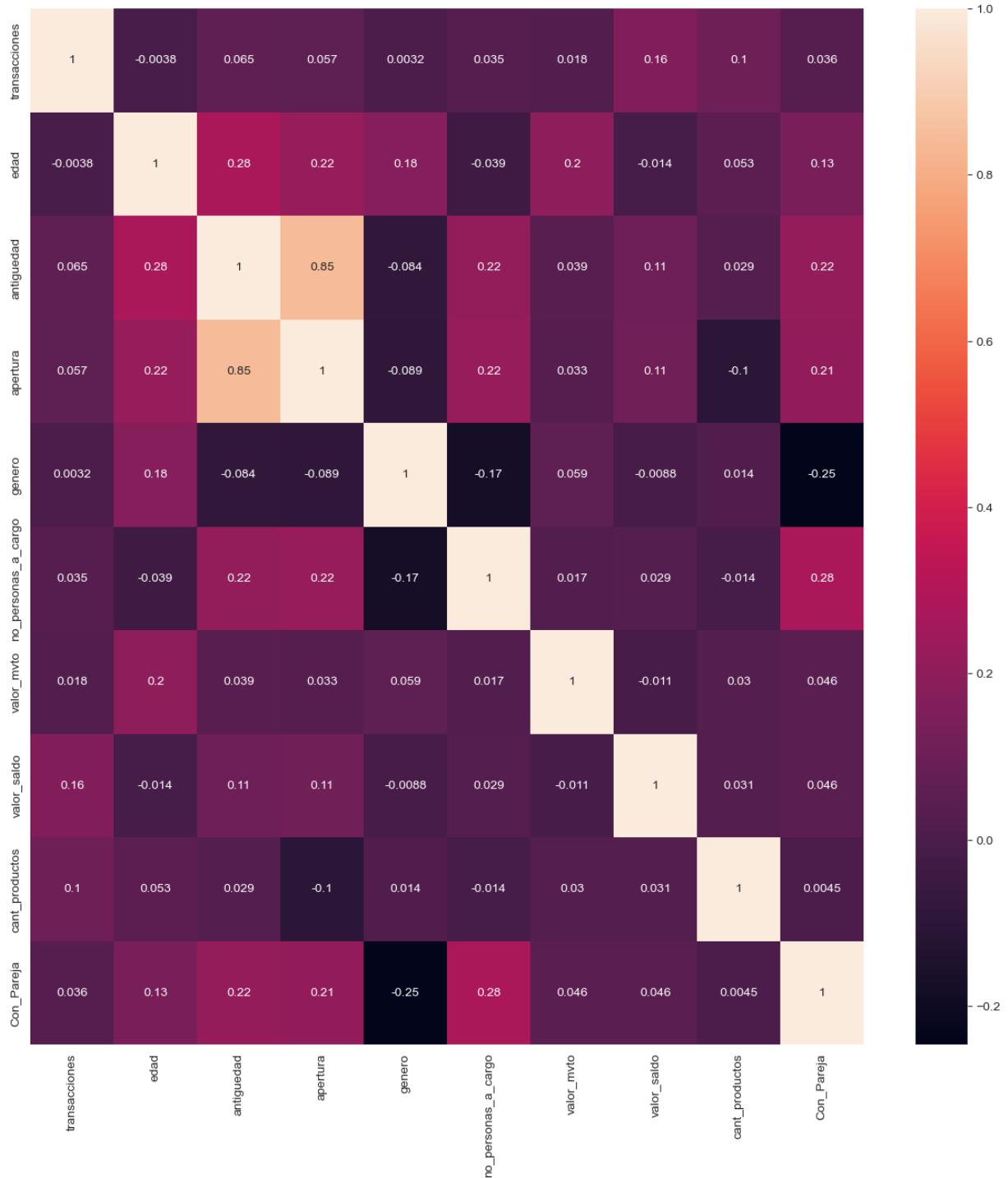




In [45]: #Correlacion de variables seleccionadas

"""Es evidente la correspondencia entre variables de antiguedad, edad y tiempo de I por el cliente. las variables de Personas a cargo y la tenencia de pareja son est entendiendo que tener pareja implica tener mas de una persona a cargo
"""

```
correlations = df.corr()
f, ax = plt.subplots(figsize = (15,15))
sns.heatmap(correlations, annot=True)
correlations.round(2);
```



```
In [100]: #Plot1 = ['antiguedad', 'transacciones', 'Con_Pareja', 'genero', 'no_personas_a_cargo',  
#sns.pairplot(df[Plot1], hue = 'valor_mvto')
```

```
In [46]: """Creación de una copia del dataset para segmentacion aplicando PCA teniendo en  
de análisis y poder reducir la dimensionalidad"""  
final_df = df.copy()  
final_aglo = df.copy()  
final_aglo = final_aglo[final_aglo['valor_saldo'] > 0]  
final_aglo = final_aglo[final_aglo['valor_saldo'] < 500000]  
final_aglo=final_aglo[:10000]
```

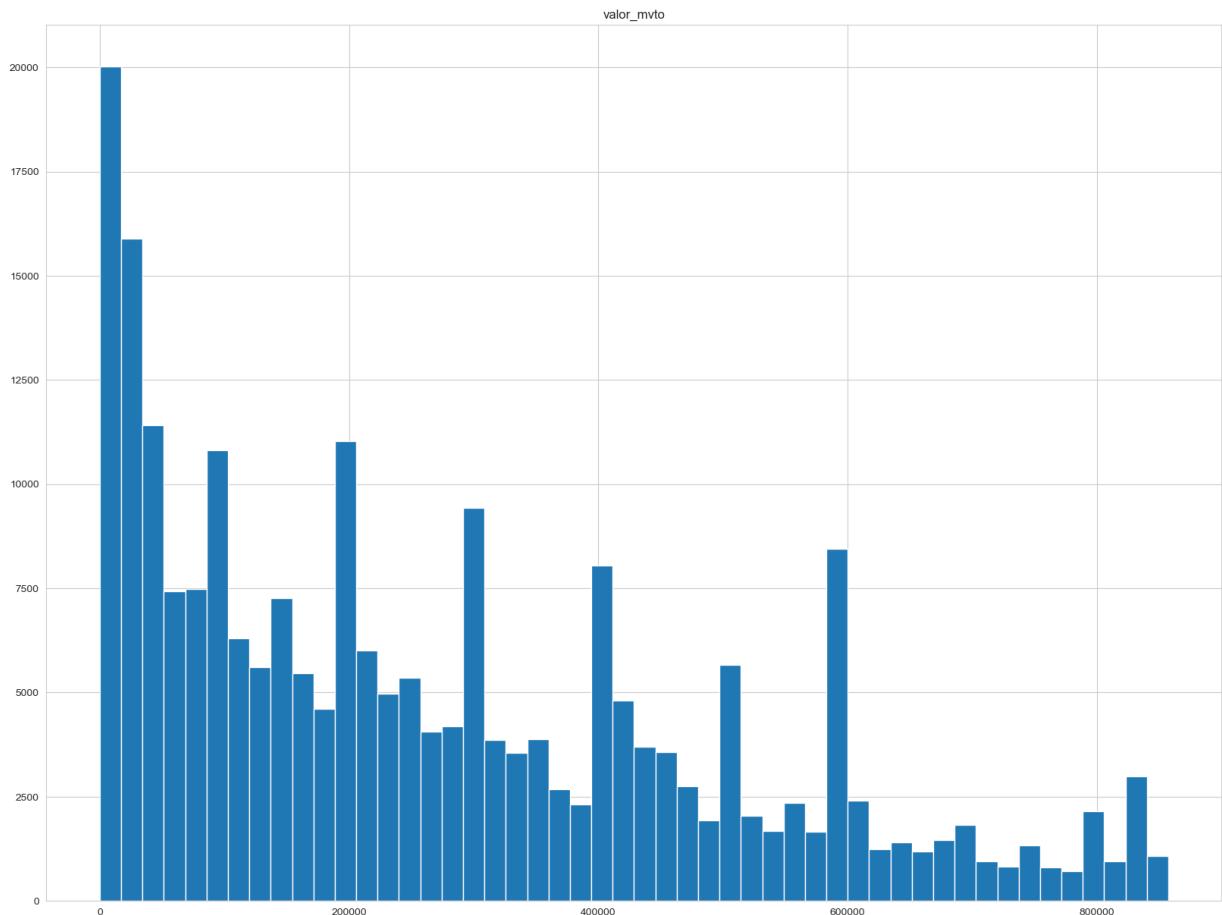
```
In [47]: # Teniendo en cuenta que se agregaron variables que disminuyen la categorización  
#y nivel de estudios. Se Eliminan del dataset al igual que el pais en donde el 96%  
final_df= final_df.drop(columns=['nivel_estudios', 'estado_civil', 'pais'], errors='ignore')  
final_aglo= final_aglo.drop(columns=['nivel_estudios', 'estado_civil', 'pais'], errors='ignore')
```

In [48]: # Adicionamos una categorización a los montos de compra por cada cliente

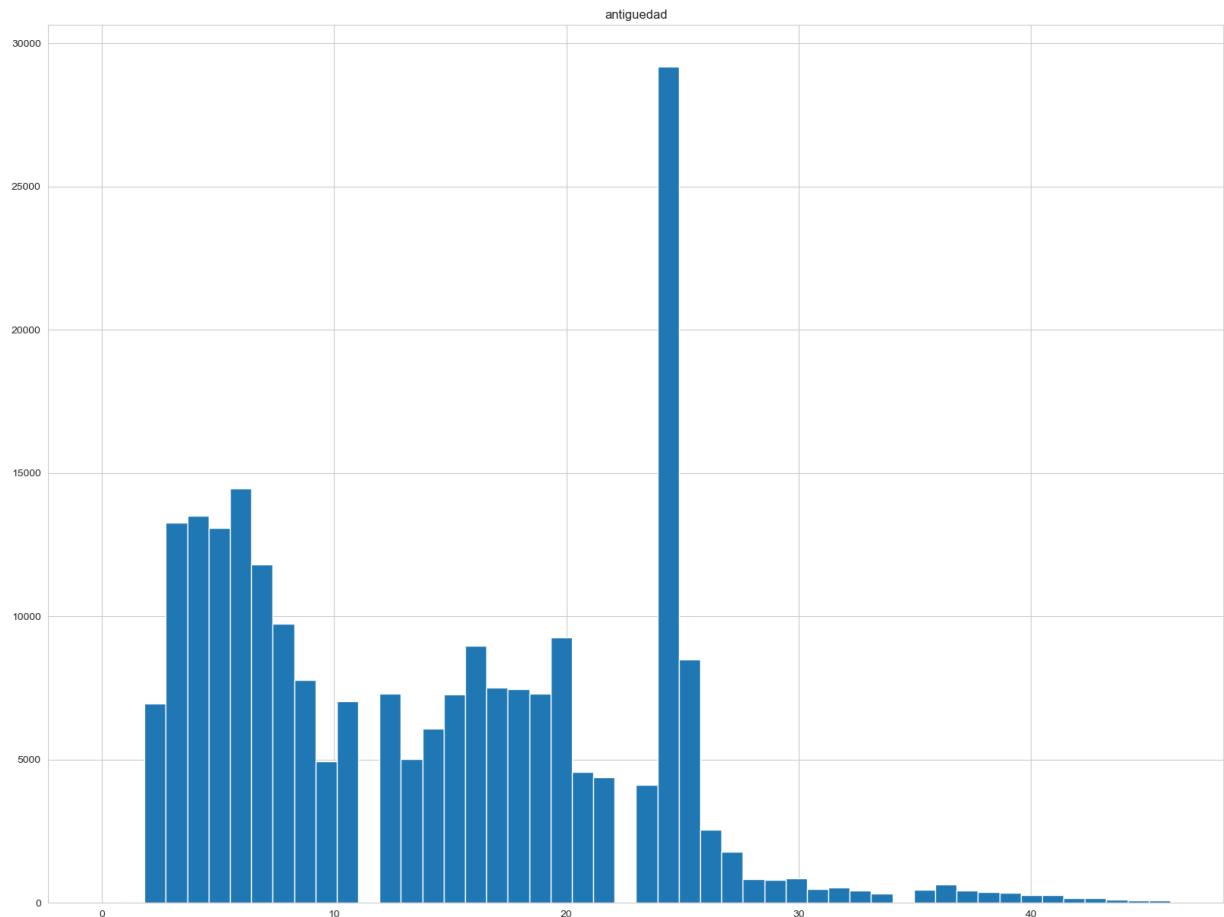
```
final_df["vl_compra_cat"] = final_df["vl_compra_cat"].replace({"Bajo":1, "Ordinario":2, "Alto":3})  
final_aglo["vl_compra_cat"] = final_aglo["vl_compra_cat"].replace({"Bajo":1, "Ordinario":2, "Alto":3})
```

In [49]: # Distribución del valor de monto en compras

```
hist_valor = final_df[['valor_mvto']]  
  
hist_valor.hist(bins=50, figsize=(20,15))  
plt.show()
```



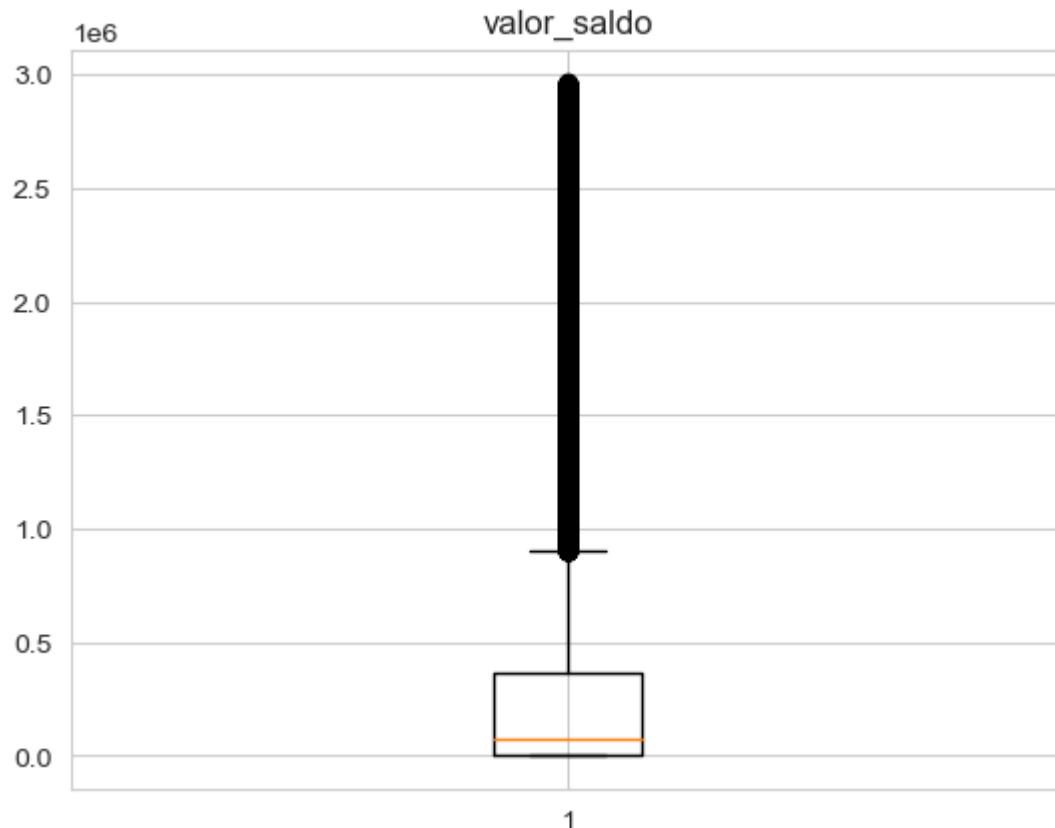
```
In [50]: #Distribucion de la antiguedad  
hist_antig=final_df[['antiguedad']]  
  
hist_antig.hist(bins=50, figsize=(20,15))  
plt.show()
```

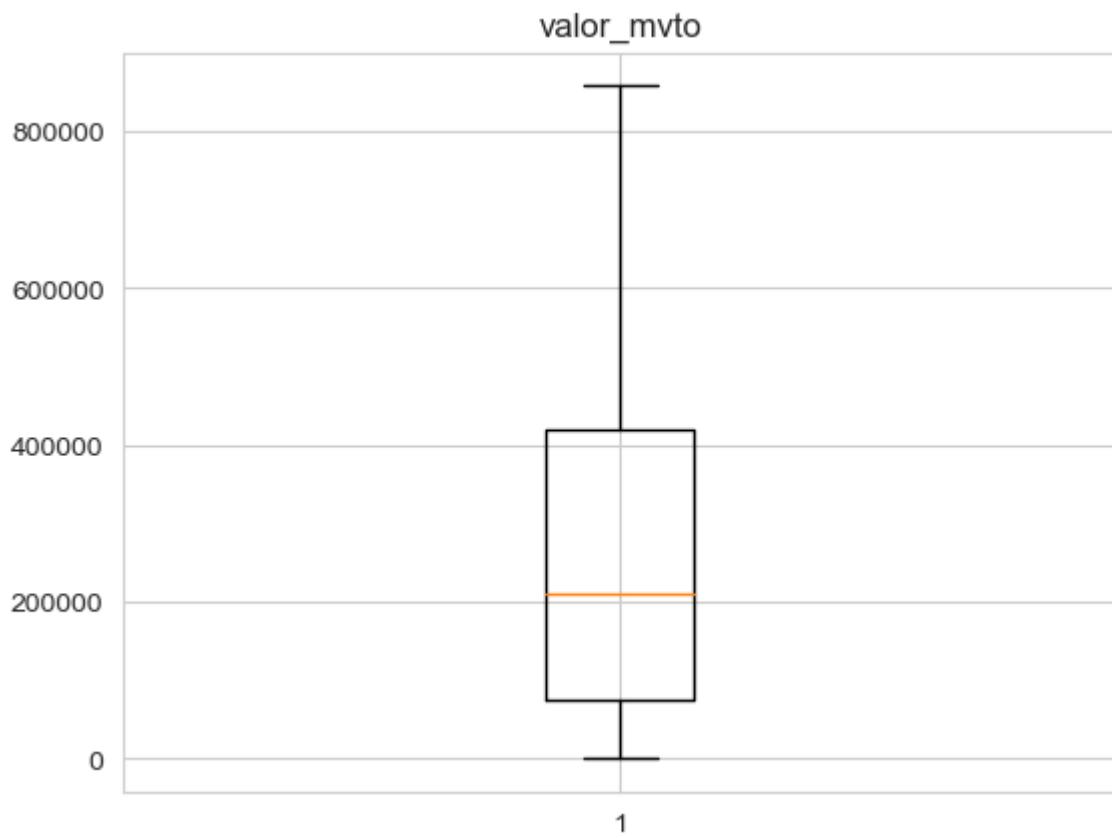


```
In [51]: #Se eliminan valores faltantes  
  
final_df=final_df.dropna()  
final_aglo=final_aglo.dropna()
```

In [52]: #Visualizamos los valores de compra y saldos por cuenta

```
for i in ['valor_saldo', 'valor_mvto']:
    plt.boxplot(final_df[i])
    plt.title(i)
    plt.show()
# plt.boxplot(final_df[i], showfliers = False)
```





```
In [53]: """codificacion de las etiquetas sobre cada una de las variables Categóricas"""

# Lista con variables categoricas no numericas
Colum_Categoricas = [col for col in final_df.columns if final_df[col].dtype=="O"]

# Se aplica el Label encoding sobre el final_df para las columnas categoricas
LE=LabelEncoder()
for i in Colum_Categoricas:
    final_df[i]=final_df[[i]].apply(LE.fit_transform)
```

```
In [54]: # Lista con variables categoricas no numericas
Colum_Categoricas = [col for col in final_aglo.columns if final_aglo[col].dtype=="O"]

# Se aplica el Label encoding sobre el final_aglo para las columnas categoricas
LE=LabelEncoder()
for i in Colum_Categoricas:
    final_aglo[i]=final_aglo[[i]].apply(LE.fit_transform)
```

```
In [55]: # Escalado de Los variables con StandardScaler

Escalado=StandardScaler()

Escalado.fit(final_df)
final_df_escalado=pd.DataFrame(Escalado.transform(final_df), columns = final_df.c
```

```
In [56]: # aplico el mismo escalado sobre el dataset seleccionado para el modelo aglomerante
Escalado1=StandardScaler()

Escalado1.fit(final_aglo)
final_aglo_escalado=pd.DataFrame(Escalado1.transform(final_aglo), columns = final_aglo.columns)
```

```
In [57]: """El siguiente dataset es el utilizado para aplicar la segmentación con la biblioteca scikit-learn. Se ha seguido un análisis de componentes principales con PCA, reduciendo la dimensionalidad del dataset y logrando una mejor interpretación y reduciendo la pérdida de información"""
final_df_escalado
```

Out[57]:

	transacciones	edad	antiguedad	apertura	ocupacion	tipo_vivienda	genero	no_p
0	-0.285643	-0.674126	1.162311	0.304624	-0.579625	0.845584	-0.873547	
1	-0.285643	1.517903	1.162311	1.072321	0.796108	-0.517648	-0.873547	
2	-0.285643	1.152565	1.397665	1.712069	0.796108	-0.517648	-0.873547	
3	-0.285643	-1.039464	-0.249808	-0.079225	-0.579625	-1.880880	-0.873547	
4	0.024786	-0.778508	0.338575	0.560523	-1.955358	-0.517648	1.144758	
...
231532	0.645644	-0.047832	-1.191220	-1.102822	0.796108	0.845584	-0.873547	
231533	0.024786	0.160933	1.397665	1.584120	-1.496780	0.845584	1.144758	
231534	-0.596071	-1.509184	-0.602837	-0.463074	-1.038202	-0.517648	-0.873547	
231535	0.024786	0.995992	0.809282	1.072321	0.796108	0.845584	-0.873547	
231536	-0.285643	-0.621934	-1.308897	-1.230771	-1.496780	0.845584	-0.873547	

231537 rows × 14 columns

Kmeans - Método del Codo

La inercia mide qué tan bien se agrupó un conjunto de datos mediante K-Means. Se calcula midiendo la distancia entre cada punto de datos y su centroide, elevando esta distancia al cuadrado y sumando estos cuadrados en un clúster.

Un buen modelo tiene baja inercia y un bajo número de clusters (K). Se compensan, ya que a mayor numero de clusters menor inercia.

El método del codo es una heurística utilizada para determinar el número de clústeres en un conjunto de datos. El método consiste en trazar la variación explicada en función del número de conglomerados y elegir el codo de la curva como el número de conglomerados a utilizar.

El método del codo es una representación gráfica de la búsqueda de la 'K' óptima en un agrupamiento de K-medias. Funciona encontrando WCSS (Within-Cluster Sum of Square), es decir, la suma de la distancia cuadrada entre los puntos de un clúster y el centroide del clúster.

- Calculamos el algoritmo de agrupación en clústeres de k-medias para diferentes valores de k, variando k de 1 a 10 clústeres.

- Para cada k, se calcula la suma total dentro del clúster del cuadrado (WSS).
- Se Grafica la curva de wss de acuerdo con el número de clústeres k. La ubicación de una flexión (rodilla) en la grafica se considera como un indicador del número apropiado de clústeres.

In [58]: *# Aplicación del Kmeans sobre la data con 14 Variables con valores por variable e*

```
import time
inicio = time.time()

wcss_1 = []
range_values = range(1, 10)
for i in range_values:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(final_df_escalado)
    wcss_1.append(kmeans.inertia_)

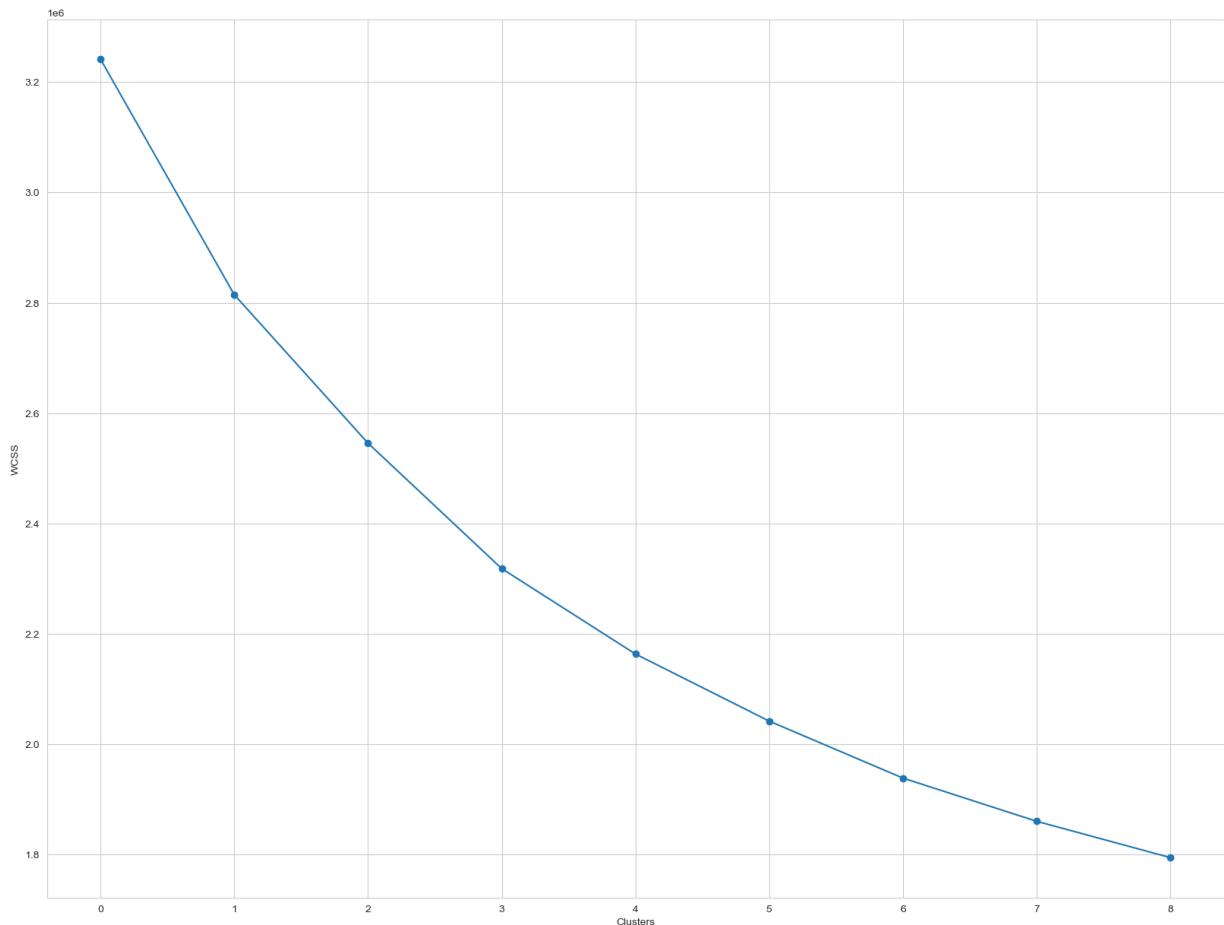
fin = time.time()
print("Tiempo de Ejecución KMeans: ", fin-inicio, " Segundos")
```

Tiempo de Ejecución KMeans: 39.701998710632324 Segundos

In [59]: `print(wcss_1)`

```
[3241517.99999999, 2814732.4554509744, 2545636.8009703117, 2318251.117794379,
2163407.1611324586, 2041798.4245503102, 1938515.147219959, 1860207.8375765686,
1794384.733241558]
```

```
In [60]: #Inercia - # Clusters
plt.figure(figsize=(20,15))
plt.plot(wcss_1, '-o')
plt.xlabel('Clusters')
plt.ylabel('WCSS');
```



```
In [61]: # Definimos La segmentación con 4 Cluster y obtenemos el Label de cada grupo del
kmeans = KMeans(n_clusters=4)
kmeans.fit(final_df_escalado)
labels = kmeans.labels_
```

```
In [62]: # Centroide para Los 4 Cluster definidos en cada una de Las variables
cluster_centers = pd.DataFrame(data = kmeans.cluster_centers_, columns = [final_d
cluster_centers
```

Out[62]:

	transacciones	edad	antiguedad	apertura	ocupacion	tipo_vivienda	genero	no_person
0	0.095499	0.260112	0.991359	1.010215	0.040135	0.319511	-0.248683	
1	-0.030352	-1.178679	-0.615278	-0.539440	-0.943980	-0.668390	-0.188884	
2	-0.111161	0.724483	-0.503964	-0.541582	0.751905	0.231327	0.413538	
3	0.905222	0.436117	0.232903	-0.862055	0.388514	0.298598	0.113525	

```
In [63]: final_df.reset_index(inplace=True)
```

In [64]: # Se agrega el label del cluster a cada registro del dataset

```
final_df_cl = pd.concat([final_df, pd.DataFrame({'cluster': labels})], axis = 1)
final_df_cl.head()
```

Out[64]:

	SK_CLIENTE	transacciones	edad	antiguedad	apertura	ocupacion	tipo_vivienda	genero	no_
0	61862	2	45.0	24.0	15.0	3		4	1
1	63954	2	87.0	24.0	21.0	6		2	1
2	68937	2	80.0	26.0	26.0	6		2	1
3	80196	2	38.0	12.0	12.0	3		0	1
4	80399	3	43.0	17.0	17.0	0		2	2

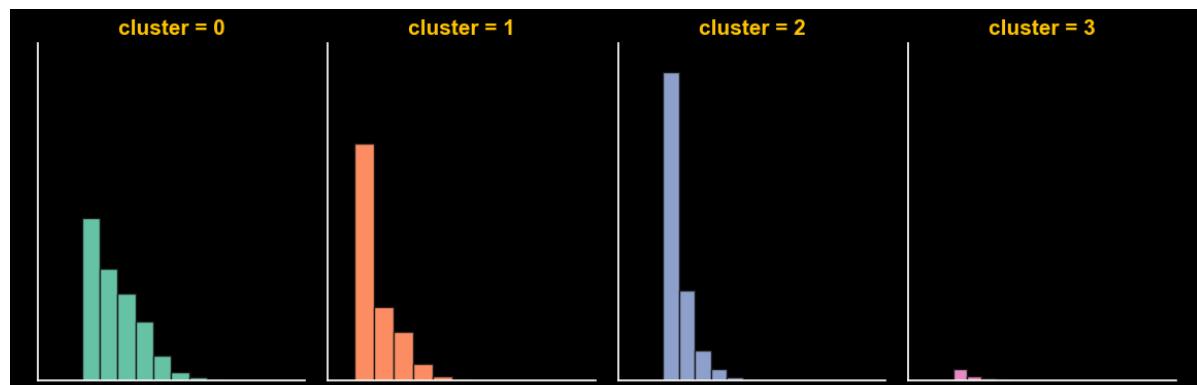
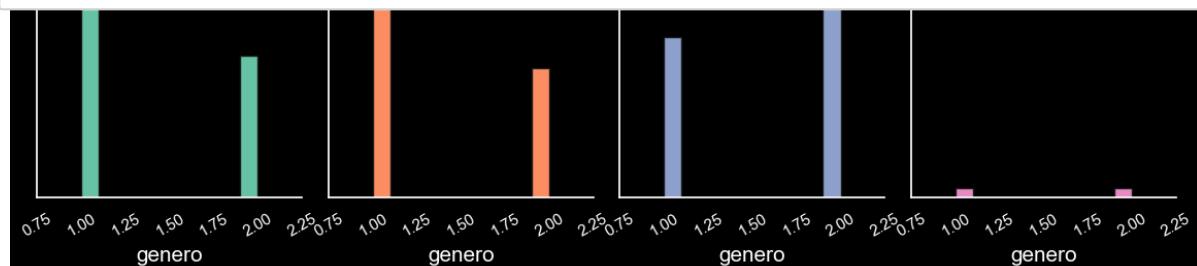
In [65]:

```
#New
#Visualizamos la caracterizacion de cada una de las variables por Cluster
```

```
"""
    Los clientes son segmentados
"""

sns.set(rc={'axes.facecolor':'black', 'figure.facecolor':'black', 'axes.grid' : False})
```

```
for i in final_df_cl:
    g = sns.FacetGrid(final_df_cl, col = "cluster", hue = "cluster", palette = "Spectral")
    g.map(plt.hist, i, bins=10, ec="k")
    g.set_xticklabels(rotation=30, color = 'white')
    g.set_yticklabels(color = 'white')
    g.set_xlabels(size=15, color = 'white')
    g.set_titles(size=15, color = '#FFC300', fontweight="bold")
    g.fig.set_figheight(5);
```



In [66]:

```
#New
#Visualizamos en un diagrama de pastel la distribucion por Cluster

Grupo_conteo = final_df_cl['cluster'].value_counts()
Grupo_conteo = Grupo_conteo.to_frame().reset_index()
Grupo_conteo.columns = ['clusters', 'count']
Grupo_conteo = Grupo_conteo.sort_values('clusters', ascending = True)

labels = [
    "0",
    "1",
    "2",
    "3"
]

plt.figure(figsize=(12,8))

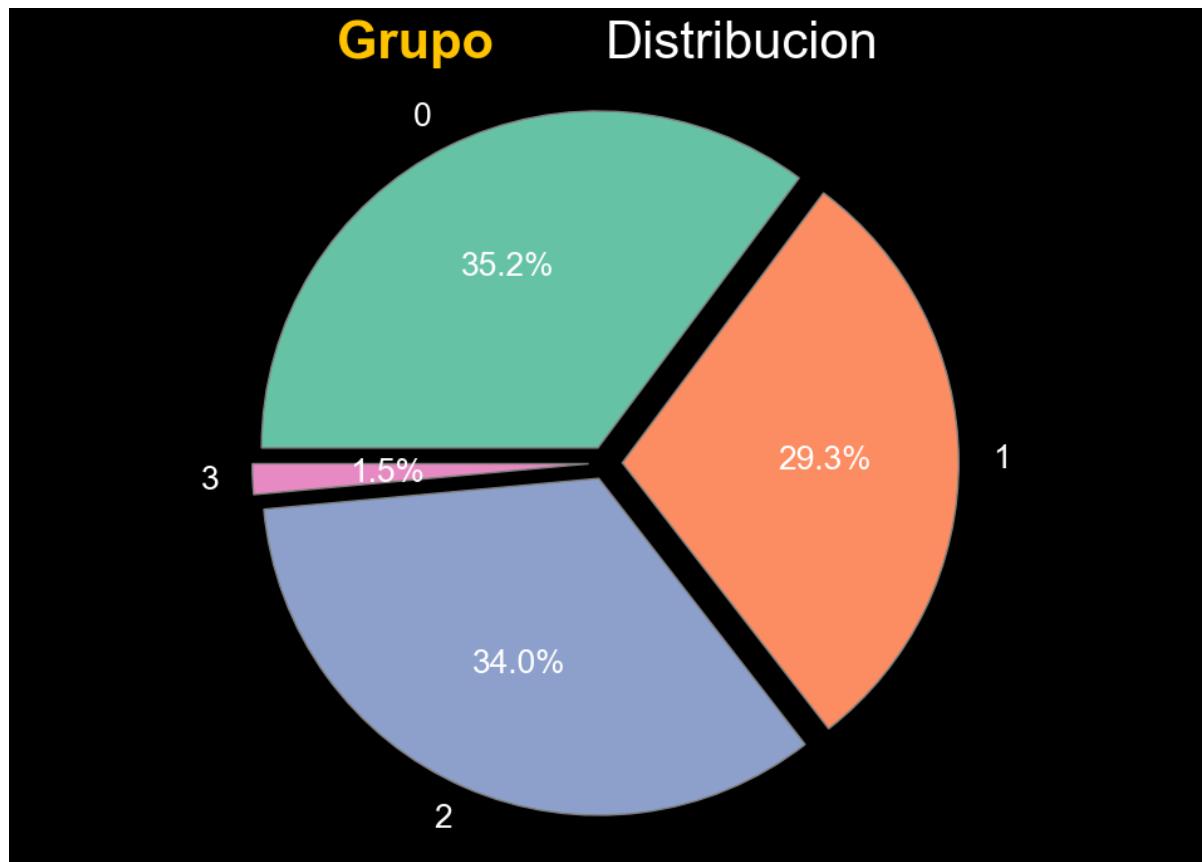
mpl.rcParams['font.size'] = 17
colors = sns.color_palette('Set2')[0:5]

plt.pie(Grupo_conteo['count'],
        explode=(0.05, 0.05, 0.05, 0.05),
        labels = labels,
        colors= colors,
        autopct='%1.1f%%',
        textprops = dict(color ="white", fontsize=19),
        counterclock = False,
        startangle=180,
        wedgeprops={"edgecolor":"gray",'linewidth':1}
    )

plt.axis('equal')

plt.text(-0.8, 1.2, "Grupo", size=30, color="#FFC300", fontweight="bold")
plt.text(-0.0, 1.2, "Distribucion", size=30, color="white")

plt.text(1.1, -1.25, "", fontsize=12, ha="right", color='lightgray', fontweight='bold')
plt.show();
```



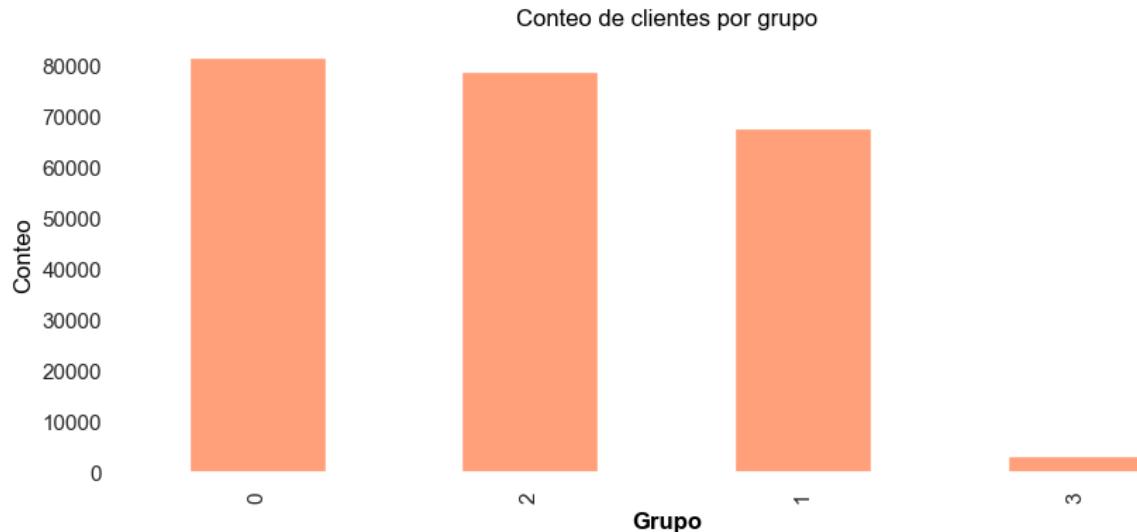
In [67]: #Cantidad de registros por cada grupo
Grupo_conteo

Out[67]:

	clusters	count
0	0	81515
2	1	67802
1	2	78838
3	3	3382

In [68]: sns.set(rc={'axes.facecolor':'white', 'figure.facecolor':'white', 'axes.grid' : F})

```
In [69]: final_df_cl.cluster.value_counts().plot(kind='bar', figsize=(10,4), color= "lightcoral")
plt.title('Conteo de clientes por grupo').set_color('black')
plt.xlabel('Grupo', fontweight="bold").set_color('black')
plt.text(1.1, -1.25, "", fontsize=70, ha="right", color='white', fontweight="bold")
_ = plt.ylabel('Conteo').set_color('black')
```



```
In [71]: k_df_1= final_df_escalado.drop(columns=['apertura'], errors='ignore')
k_df_2= final_df_escalado.drop(columns=['apertura', 'Educacion'], errors='ignore')
k_df_3= final_df_escalado.drop(columns=['apertura', 'Educacion', 'ocupacion'], errors='ignore')
k_df_4= final_df_escalado.drop(columns=['valor_saldo', 'Educacion', 'ocupacion'], errors='ignore')
k_df_5= pd.DataFrame(final_df_escalado, columns = ['edad', 'antiguedad','genero','tipodeconsumo'])

k_df_1_ini= final_df.drop(columns=['apertura'], errors='ignore')
k_df_2_ini= final_df.drop(columns=['apertura', 'Educacion'], errors='ignore')
k_df_3_ini= final_df.drop(columns=['apertura', 'Educacion', 'ocupacion'], errors='ignore')
k_df_4_ini= final_df.drop(columns=['valor_saldo', 'Educacion', 'ocupacion', 'apertura'], errors='ignore')
k_df_5_ini= pd.DataFrame(final_df, columns = ['edad', 'antiguedad','genero','tipodeconsumo'])

kdf_2v=pd.DataFrame(final_df, columns = ['valor_mvto','valor_saldo'])
```

In [72]: #Validación de Kmeans para la data con 12 variables, sin tener en cuenta la varia

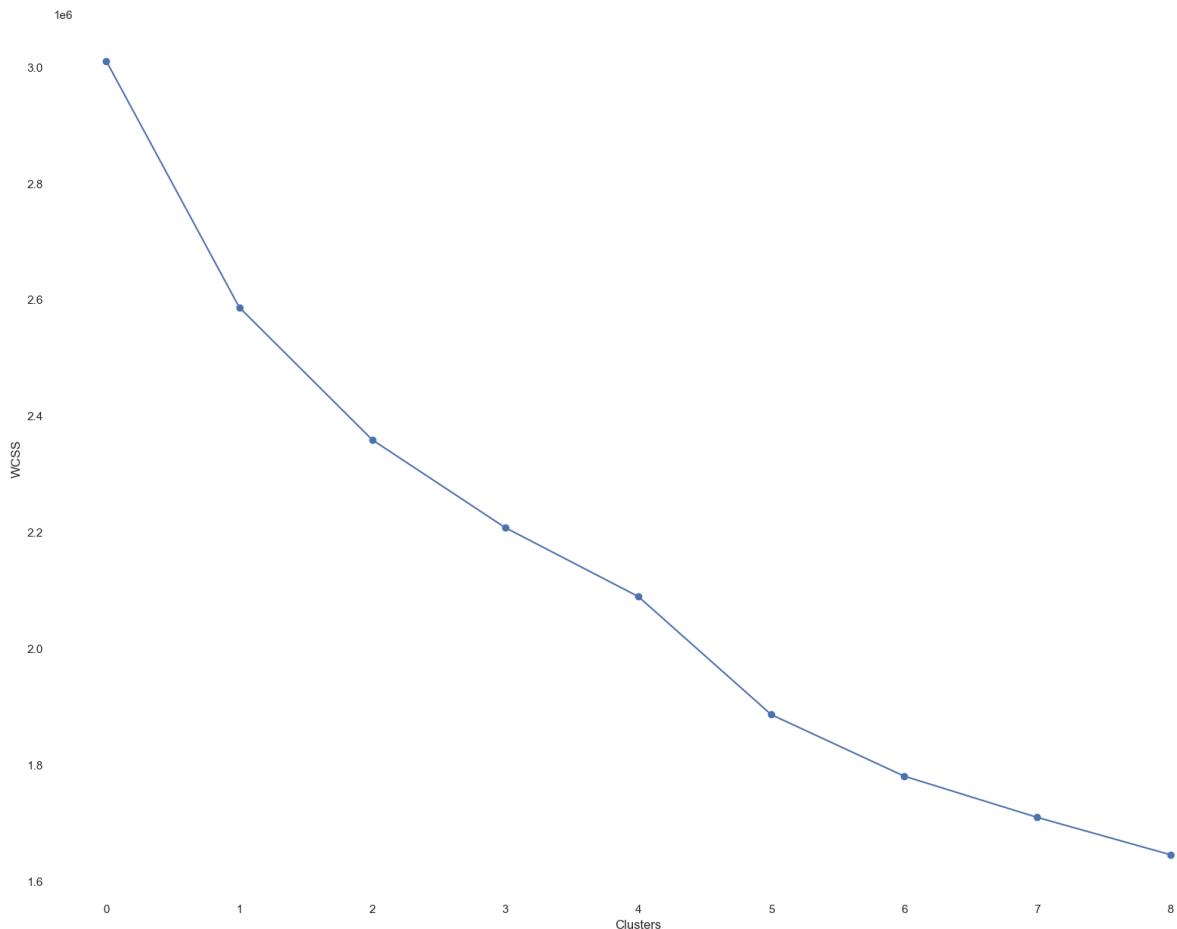
```
import time
inicio = time.time()

wcss_1 = []
range_values = range(1, 10)
for i in range_values:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(k_df_1)
    wcss_1.append(kmeans.inertia_)

fin = time.time()
print("Tiempo de Ejecución KMeans: ", fin-inicio, " Segundos")

#Inercia - # Clusters
plt.figure(figsize=(20,15))
plt.plot(wcss_1, '-o')
plt.xlabel('Clusters')
plt.ylabel('WCSS');
```

Tiempo de Ejecución KMeans: 37.52399206161499 Segundos



```
In [73]: wcss_1
```

```
Out[73]: [3009980.99999995,  
 2586238.878625911,  
 2358343.531017599,  
 2207232.306256655,  
 2088787.7042177843,  
 1885568.1990926818,  
 1779722.2615341144,  
 1708818.1448409685,  
 1644672.7652097067]
```

In [74]: #Validación de Kmeans para la data con 11 variables, sin tener en cuenta la varia

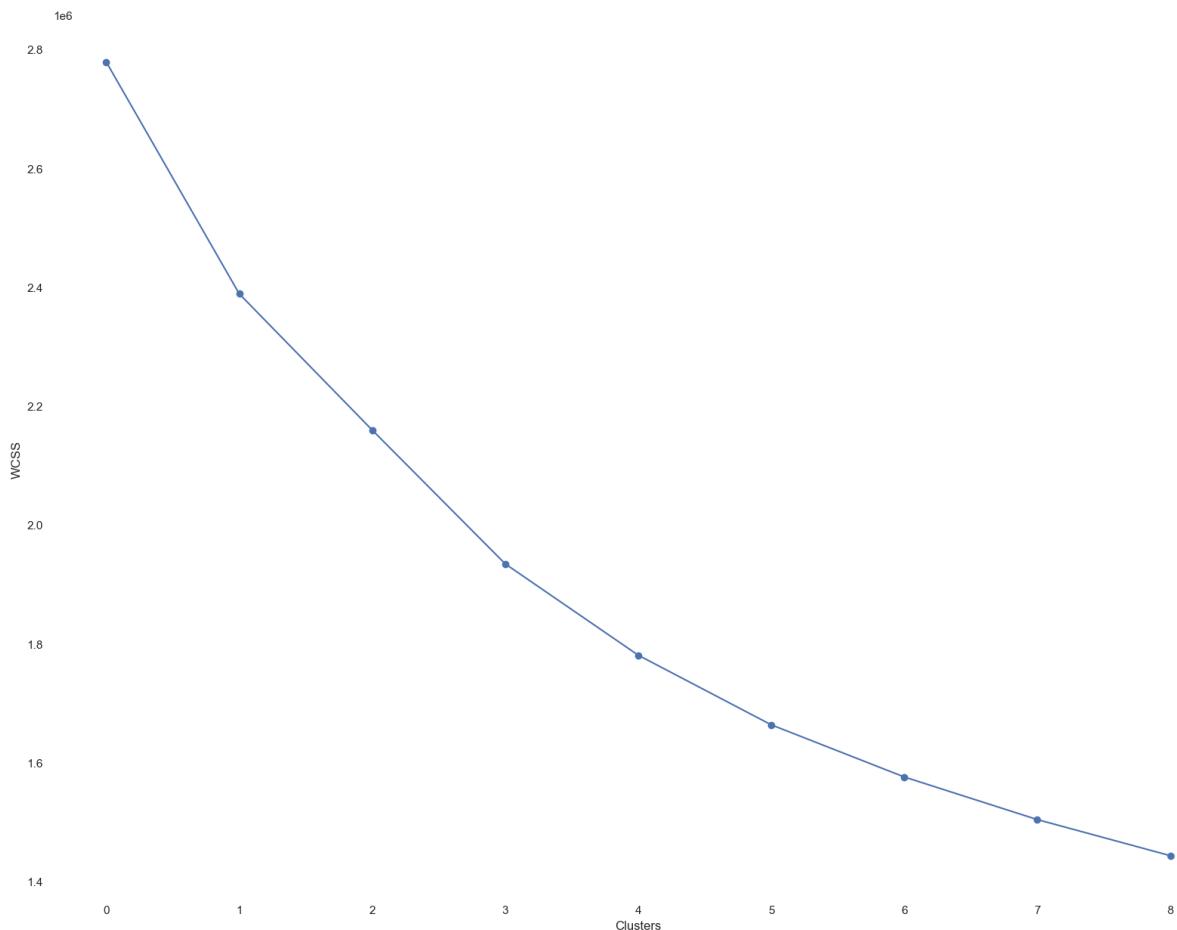
```
import time
inicio = time.time()

wcss_1 = []
range_values = range(1, 10)
for i in range_values:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(k_df_2)
    wcss_1.append(kmeans.inertia_)

fin = time.time()
print("Tiempo de Ejecución KMeans: ", fin-inicio, " Segundos")

#Inercia - # Clusters
plt.figure(figsize=(20,15))
plt.plot(wcss_1, '-o')
plt.xlabel('Clusters')
plt.ylabel('WCSS');
```

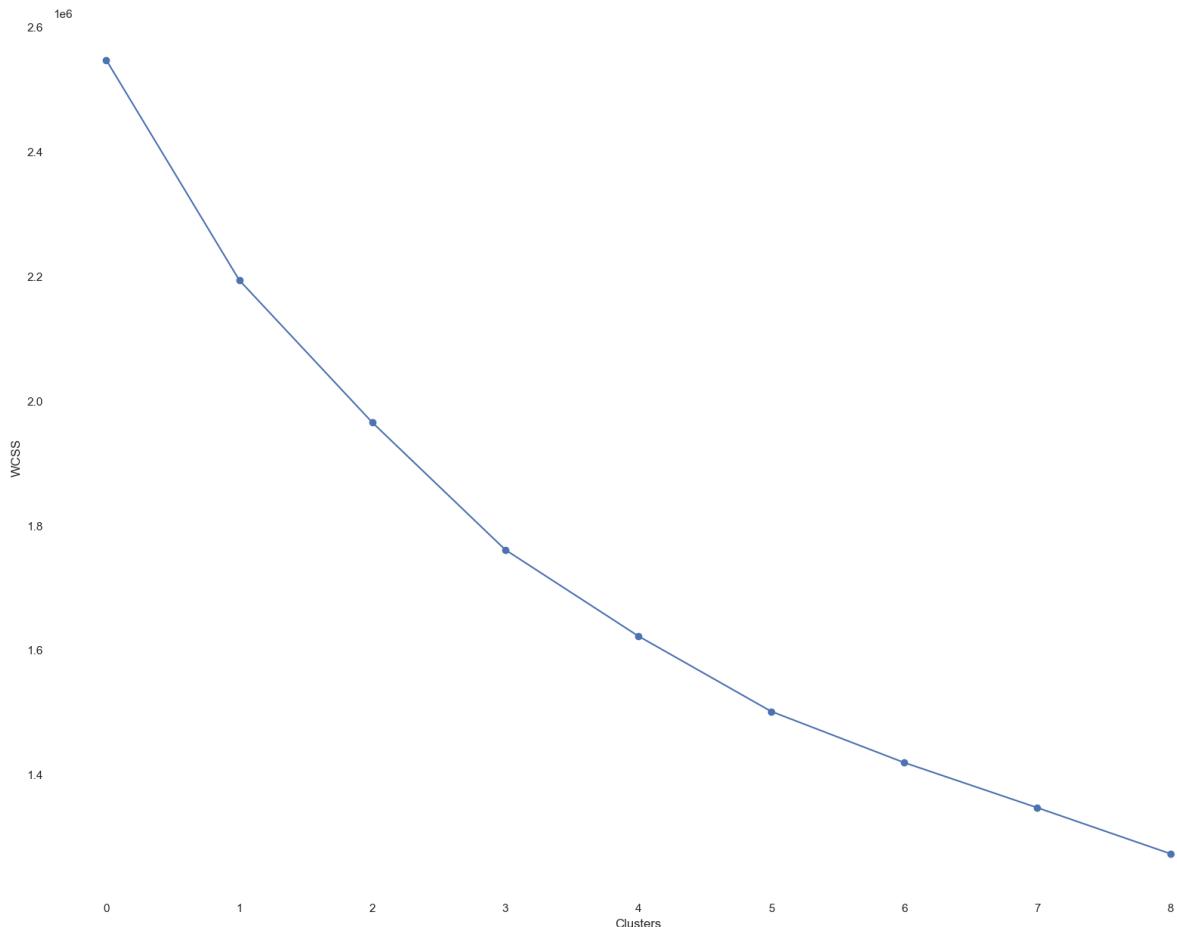
Tiempo de Ejecución KMeans: 36.59701490402222 Segundos



In [75]:

```
#Validación de Kmeans para la data con 10 variables, sin tener en cuenta la varia  
  
import time  
inicio = time.time()  
  
wcss_1 = []  
range_values = range(1, 10)  
for i in range_values:  
    kmeans = KMeans(n_clusters=i)  
    kmeans.fit(k_df_3)  
    wcss_1.append(kmeans.inertia_)  
  
fin = time.time()  
print("Tiempo de Ejecución KMeans: ", fin-inicio, " Segundos")  
  
#Inercia - # Clusters  
plt.figure(figsize=(20,15))  
plt.plot(wcss_1, '-o',)  
plt.xlabel('Clusters',)  
plt.ylabel('WCSS');
```

Tiempo de Ejecución KMeans: 37.035255432128906 Segundos



In [76]:

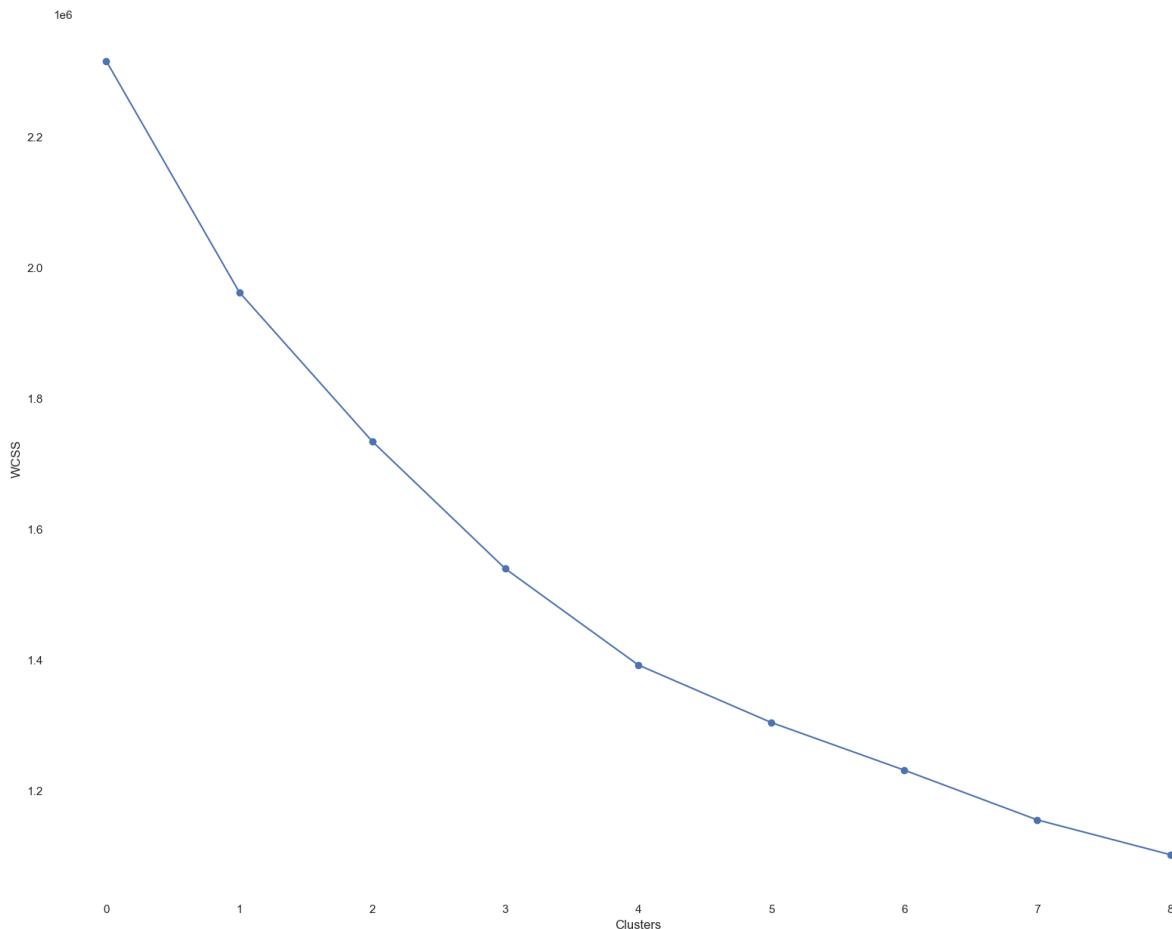
```
#Validación de Kmeans para la data con 9 variables, sin tener en cuenta valor_sal
import time
inicio = time.time()

wcss_1 = []
range_values = range(1, 10)
for i in range_values:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(k_df_4)
    wcss_1.append(kmeans.inertia_)

fin = time.time()
print("Tiempo de Ejecución KMeans: ", fin-inicio, " Segundos")

#Inercia - # Clusters
plt.figure(figsize=(20,15))
plt.plot(wcss_1, '-o')
plt.xlabel('Clusters')
plt.ylabel('WCSS');
```

Tiempo de Ejecución KMeans: 36.183603048324585 Segundos



```
In [77]: # Data agrupada con las variables edad, antiguedad,genero,tipo_vivienda,valor_mvi

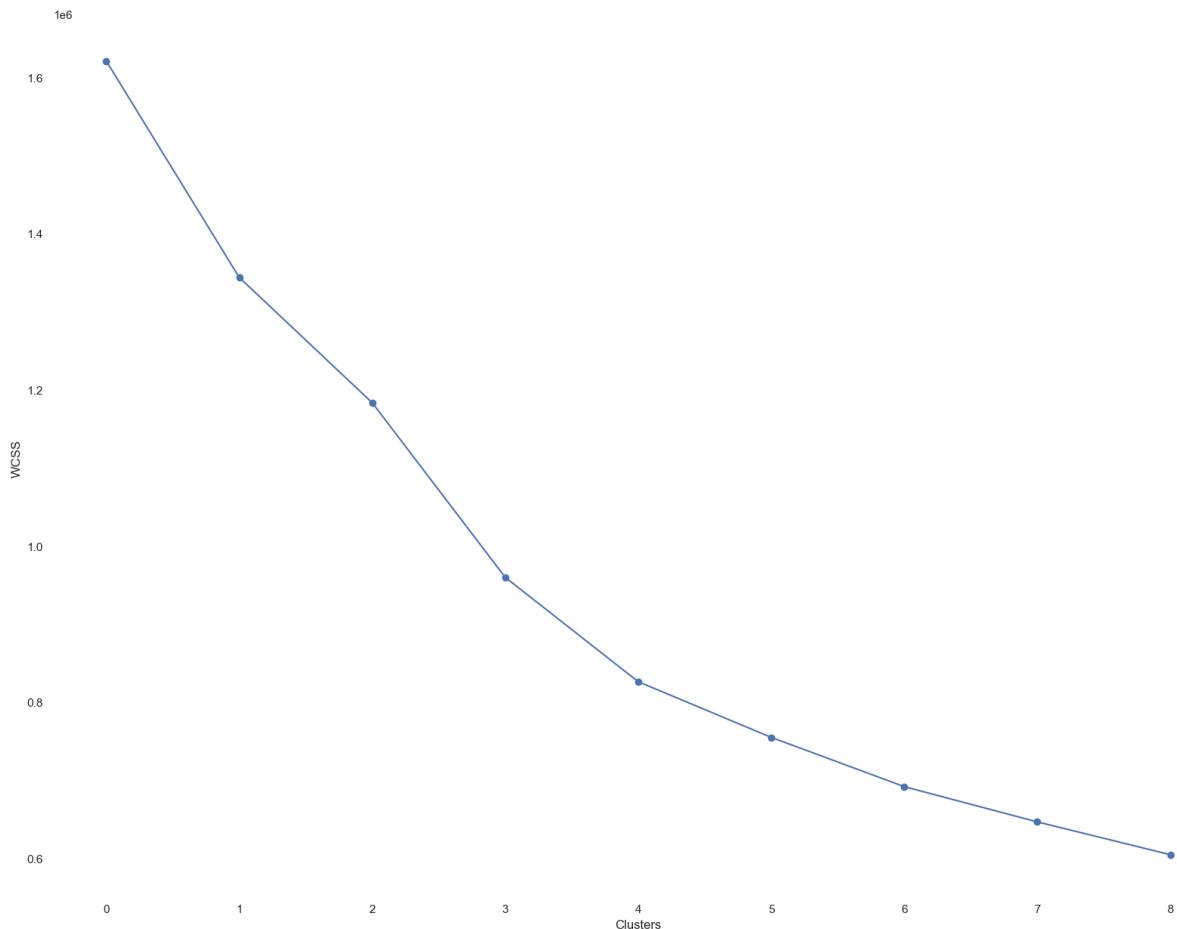
import time
inicio = time.time()

wcss_1 = []
range_values = range(1, 10)
for i in range_values:
    kmeans = KMeans(n_clusters=i)
    kmeans.fit(k_df_5)
    wcss_1.append(kmeans.inertia_)

fin = time.time()
print("Tiempo de Ejecución KMeans: ", fin-inicio, " Segundos")

#Inercia - # Clusters
plt.figure(figsize=(20,15))
plt.plot(wcss_1, '-o')
plt.xlabel('Clusters')
plt.ylabel('WCSS');
```

Tiempo de Ejecución KMeans: 33.42035937309265 Segundos



In [78]: #Nuevo

```
#aplicando diferentes variables al kmeans y calculo de la media del siluthe

from sklearn.metrics import silhouette_score

kmeans_1 = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
kmeans_2 = KMeans(n_clusters = 3, init = 'k-means++', random_state = 42)
kmeans_3 = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
kmeans_4 = KMeans(n_clusters = 4, init = 'k-means++', random_state = 42)
kmeans_5 = KMeans(n_clusters = 4, init = 'k-means++', random_state = 42)

y_kmeans_1 = kmeans_1.fit_predict(k_df_1)
y_kmeans_2 = kmeans_2.fit_predict(k_df_2)
y_kmeans_3 = kmeans_3.fit_predict(k_df_3)
y_kmeans_4 = kmeans_4.fit_predict(k_df_4)
y_kmeans_5 = kmeans_5.fit_predict(k_df_5)

k_df_1_ini['Cluster'] = y_kmeans_1
k_df_2_ini['Cluster'] = y_kmeans_2
k_df_3_ini['Cluster'] = y_kmeans_3
k_df_4_ini['Cluster'] = y_kmeans_4
k_df_5_ini['Cluster'] = y_kmeans_5
```

In [111]: #Nuevo

```
# Cálculo Silhouette Score 1
score = silhouette_score(k_df_1, kmeans_1.labels_, metric='euclidean')
# Score Silhouette para Data 1
print('Silhouetter Average Score Para el data set 1: %.3f' % score)

# Cálculo Silhouette Score 2
score = silhouette_score(k_df_2, kmeans_2.labels_, metric='euclidean')
# Score Silhouette para Data 2
print('Silhouetter Average Score Para el data set 2: %.3f' % score)

# Cálculo Silhouette Score 3
score = silhouette_score(k_df_3, kmeans_3.labels_, metric='euclidean')
# Score Silhouette para Data 3
print('Silhouetter Average Score Para el data set 3: %.3f' % score)

# Cálculo Silhouette Score 4
score = silhouette_score(k_df_4, kmeans_4.labels_, metric='euclidean')
# Score Silhouette para Data 4
print('Silhouetter Average Score Para el data set 4: %.3f' % score)

# Cálculo Silhouette Score 5
score = silhouette_score(k_df_5, kmeans_5.labels_, metric='euclidean')
# Score Silhouette para Data 5
print('Silhouetter Average Score Para el data set 5: %.3f' % score)
```

Silhouetter Average Score Para el data set 1: 0.148
Silhouetter Average Score Para el data set 2: 0.144
Silhouetter Average Score Para el data set 3: 0.152
Silhouetter Average Score Para el data set 4: 0.169
Silhouetter Average Score Para el data set 5: 0.215

In [88]: #Nuevo

```
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(10,30))
fig.suptitle('K-Means', size = 18)

axes[0,0].scatter(k_df_1_ini['antiguedad'], k_df_1_ini['vl_compra_cat'], c=k_df_1_ini['Cluster'])
axes[0,0].set_title("13_Var Antiguedad-Compras");
axes[0,1].scatter(k_df_1_ini['edad'], k_df_1_ini['transacciones'], c=k_df_1_ini['Cluster'])
axes[0,1].set_title("13_Var Edad-Transacciones");

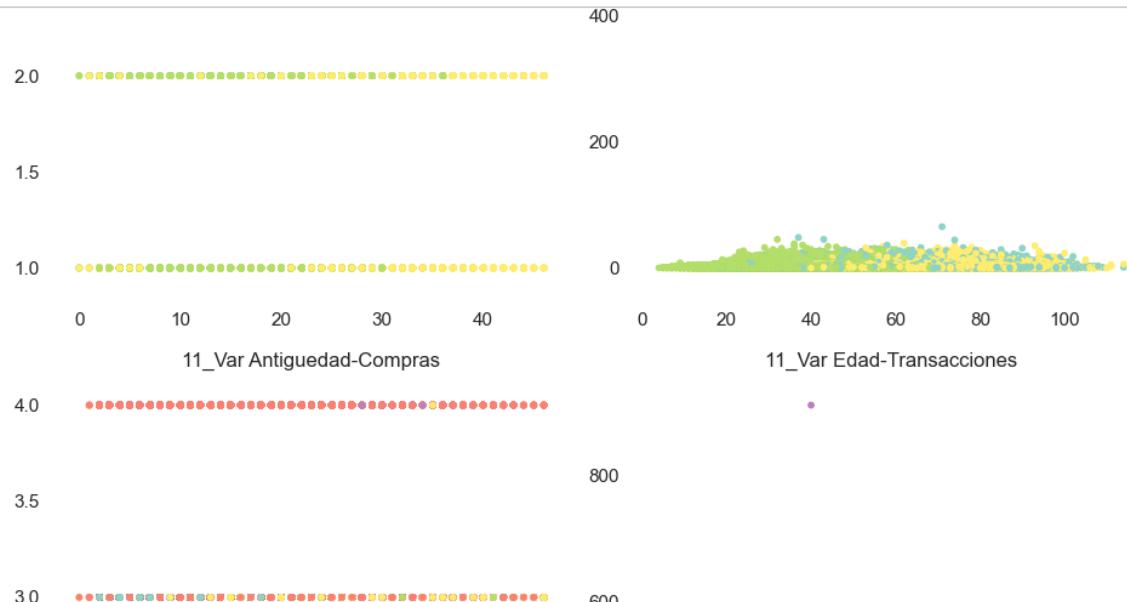
axes[1,0].scatter(k_df_2_ini['antiguedad'], k_df_2_ini['vl_compra_cat'], c=k_df_2_ini['Cluster'])
axes[1,0].set_title("12_Var Antiguedad-Compras");
axes[1,1].scatter(k_df_2_ini['edad'], k_df_2_ini['transacciones'], c=k_df_2_ini['Cluster'])
axes[1,1].set_title("12_Var Edad-Transacciones");

axes[2,0].scatter(k_df_3_ini['antiguedad'], k_df_3_ini['vl_compra_cat'], c=k_df_3_ini['Cluster'])
axes[2,0].set_title("11_Var Antiguedad-Compras");
axes[2,1].scatter(k_df_3_ini['edad'], k_df_3_ini['transacciones'], c=k_df_3_ini['Cluster'])
axes[2,1].set_title("11_Var Edad-Transacciones");

axes[3,0].scatter(k_df_4_ini['antiguedad'], k_df_4_ini['vl_compra_cat'], c=k_df_4_ini['Cluster'])
axes[3,0].set_title("10_Var Antiguedad-Compras");
axes[3,1].scatter(k_df_4_ini['edad'], k_df_4_ini['transacciones'], c=k_df_4_ini['Cluster'])
axes[3,1].set_title("10_Var Edad-Transacciones");

axes[4,0].scatter(k_df_5_ini['antiguedad'], k_df_5_ini['valor_mvto'], c=k_df_5_ini['Cluster'])
axes[4,0].set_title("7_Var Antiguedad-Compras");
axes[4,1].scatter(k_df_5_ini['edad'], k_df_5_ini['valor_saldo'], c=k_df_5_ini['Cluster'])
axes[4,1].set_title("7_Var edad-saldo");

plt.tight_layout()
```



In [91]: #Nuevo

```
kmeans_df= k_df_5_ini.drop(columns=['Cluster'], errors='ignore')
kmeans_df_2= k_df_4_ini.drop(columns=['Cluster'], errors='ignore')
```

#Nuevo

Métricas de precisión A diferencia de la clasificación, es difícil evaluar la calidad de los resultados de la agrupación. Aquí, una métrica no puede depender de las etiquetas, sino solo de la bondad de la división. En segundo lugar, no solemos tener etiquetas verdaderas de las observaciones cuando usamos clustering.

Las métricas externas utilizan la información sobre la división verdadera conocida.

Todas las métricas que se describen a continuación se implementan en `sklearn.metrics`.

ARI: Esta métrica es simétrica y no depende de la permutación de la etiqueta. Este índice es una medida de las distancias entre diferentes divisiones de muestra. ARI toma valores en el rango de [-1,1]. Los valores negativos indican la independencia de las divisiones y los valores positivos indican que estas divisiones son consistentes (coinciden con ARI=1).

AMI: Esta métrica es similar a ARI . También es simétrico y no depende de los valores y la permutación de las etiquetas. Se define mediante la función de entropía e interpreta una división de muestra como una distribución discreta (la probabilidad de asignar a un clúster es igual al porcentaje de objetos que contiene).

El AMI se encuentra en el rango [0,1]. Los valores cercanos a cero significan que las divisiones son independientes, y los cercanos a 1 significan que son similares (con coincidencia completa en AMI = 1).

Homogeneidad, completitud, medida V

Formalmente, estas métricas también se definen en función de la función de entropía.

Por lo tanto, se evalúa si cada clúster está compuesto por los mismos objetos de clase y se mide qué tan bien se ajustan los mismos objetos de clase a los clústeres. Los valores de estas métricas no se escalan como las métricas ARI o AMI y, por lo tanto, dependen del número de clústeres.

Un resultado de agrupación aleatoria en clústeres no tendrá valores de métricas más cercanos a cero cuando el número de clústeres sea lo suficientemente grande y el número de objetos sea pequeño. En tal caso, sería más razonable utilizar ARI . Sin embargo, con un gran número de observaciones (más de 100) y un número de conglomerados inferior a 10, este problema es menos crítico y puede ignorarse.

La medida V es una combinación es una media armónica, es simétrico y mide la consistencia de los resultados de dos agrupamientos.

Silhouette: este coeficiente no implica el conocimiento sobre las etiquetas verdaderas de los objetos. Nos permite estimar la calidad de la agrupación utilizando solo la muestra inicial sin etiquetar y el resultado de la agrupación. Para empezar, para cada observación, se calcula el coeficiente de silueta.

Sea "a" la media de la distancia entre un objeto y otros objetos dentro de un grupo y "b" la distancia media de un objeto a un objeto del grupo más cercano (diferente de aquel al que pertenece el objeto). La silueta de una muestra es un valor medio de los valores de silueta de esta muestra. Por lo tanto, la distancia de silueta, muestra hasta qué punto la distancia entre los objetos de la misma clase difiere de la distancia media entre los objetos de diferentes clústeres.

Este coeficiente toma valores en el rango $[-1,1]$. Los valores cercanos a -1 corresponden a malos resultados de agrupación en clústeres, mientras que los valores más cercanos a 1 corresponden a clústeres densos y bien definidos. Por lo tanto, cuanto mayor sea el valor de la silueta, mejores serán los resultados de la agrupación.

In [172]:

```
"""
Nuevo

Función para dar lectura de las metricas sobre la tecnica de segmentacion; metricas
marcadas en al segmentacion con kmeans, técnica de inicializacion kmeans++, random
"""

from time import time

from sklearn import metrics
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler


def ref_k_means(kmeans, name, data, labels):
    """Punto de referencia para evaluar los metodos de inicializacion del kmeans

    Parametros
    -----
    kmeans : técnica usada
    name : Muestra resultados
    data: Los datos para el Cluster
    labels: Etiquetas utilizadas para calcular las métricas de agrupación en clústeres
    """

    #Evalua el tiempo de escalado
    t0 = time()
    estimator = make_pipeline(StandardScaler(), kmeans).fit(data)
    fit_time = time() - t0
    results = [name, fit_time, estimator[-1].inertia_]

    # Define las Metricas que solo requieren las etiquetas y el estimador

    clustering_metrics = [
        metrics.homogeneity_score,
        metrics.completeness_score,
        metrics.v_measure_score,
        metrics.adjusted_rand_score,
        metrics.adjusted_mutual_info_score,
    ]
    results += [m(labels, estimator[-1].labels_) for m in clustering_metrics]

    # EL silhouette
    results += [
        metrics.silhouette_score(
            data,
            estimator[-1].labels_,
            metric="euclidean",
            sample_size=300,
        )
    ]

    # Muestra los resultados
    formatter_result = (
        "{:9s}\t{:.3f}s\t{:.0f}\t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}\t{:.3f}"
    )
```

```
)  
print(formatter_result.format(*results))
```

In [107]: #Nuevo

```
print(82 * "_")
print("init\ttime\tinertia\thomo\tcompl\tv-meas\tARI\tAMI\tsilhouette")

kmeans = KMeans(init="k-means++", n_clusters=4, n_init=4, random_state=0)
ref_k_means(kmeans=kmeans, name="k-means+_d1", data=kmeans_df, labels=y_kmeans_5)

kmeans = KMeans(init="random", n_clusters=4, n_init=4, random_state=0)
ref_k_means(kmeans=kmeans, name="random_d1", data=kmeans_df, labels=y_kmeans_5)

pca = PCA(n_components=4).fit(kmeans_df)
kmeans = KMeans(init=pca.components_, n_clusters=4, n_init=1)
ref_k_means(kmeans=kmeans, name="PCA-based_d1", data=kmeans_df, labels=y_kmeans_5)

kmeans = KMeans(init="k-means++", n_clusters=4, n_init=4, random_state=0)
ref_k_means(kmeans=kmeans, name="k-means+_d2", data=kmeans_df_2, labels=y_kmeans_4)

kmeans = KMeans(init="random", n_clusters=4, n_init=4, random_state=0)
ref_k_means(kmeans=kmeans, name="random_d2", data=kmeans_df_2, labels=y_kmeans_4)

pca = PCA(n_components=4).fit(kmeans_df_2)
kmeans = KMeans(init=pca.components_, n_clusters=4, n_init=1)
ref_k_means(kmeans=kmeans, name="PCA-based_d2", data=kmeans_df_2, labels=y_kmeans_4)

print(82 * "_")
```

init	time	inertia	homo	compl	v-meas	ARI	AMI	silhouette
tte								
k-means+_d1	1.513s	959735	1.000	1.000	1.000	1.000	1.000	-0.099
random_d1	1.266s	1046706	0.826	0.736	0.778	0.830	0.778	0.004
PCA-based_d1	0.369s	1104129	0.736	0.620	0.673	0.678	0.673	-0.015
k-means+_d2	1.591s	1748372	0.819	0.819	0.819	0.858	0.819	-0.189
random_d2	1.458s	1806138	0.534	0.535	0.534	0.490	0.534	-0.112
PCA-based_d2	0.442s	1833906	0.615	0.513	0.559	0.558	0.559	-0.030

In [175]:

```
"""
Nuevo
```

El siguiente algoritmo es el más simple de entender entre todos los algoritmos de sin un número fijo de clústeres.

Comienza asignando cada observación a su propio clúster, ordena las distancias po en orden descendente, se Toman los dos clústeres vecinos más cercanos, se combina Estos pasos son repetidos hasta que todos los datos se combinen en un clúster.

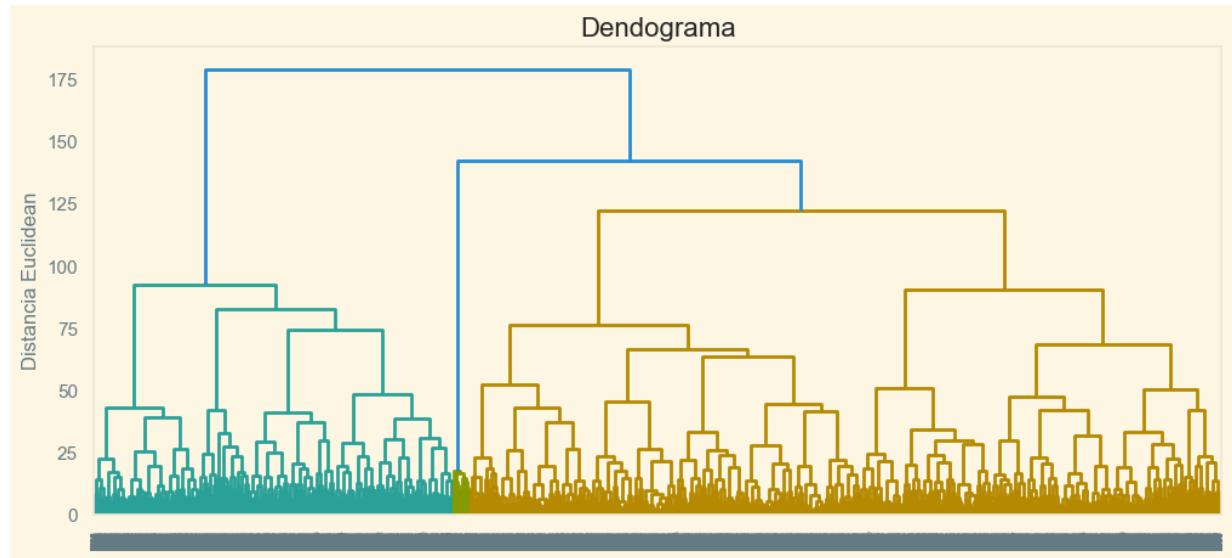
Generalmente al utilizar un dendrograma, solemos tomar la altura máxima entre bif la cantidad de clusters. El problema de este proceso es que la distancia entre cl sesgada a los datos iniciales, y reflejan poco los datos agregados. Por esto, una correctos es con un gráfico de Siluetas. Si no fuera por esto, el mejor número de está sobresimplificando el problema.

```
"""
```

```
import scipy.cluster.hierarchy as sch
```

```
from matplotlib import pyplot
pyplot.figure(figsize=(12, 5))
dendrogram = sch.dendrogram(sch.linkage(final_aglo_escalado, method = 'ward'))

plt.title('Dendograma')
plt.ylabel('Distancia Euclidean')
plt.show()
```



```
In [108]: from sklearn.metrics import silhouette_score

# Instantiate the KMeans for 5 clusters
km = KMeans(n_clusters=5, random_state=42)
# Fit the KMeans model
km.fit_predict(final_aglo)
# Calculate Silhouette Score
score = silhouette_score(final_aglo, km.labels_, metric='euclidean')
# Print the score
print('Silhouetter Average Score: %.3f' % score)
```

Silhouetter Average Score: 0.408

PCA

El análisis de componentes principales es uno de los métodos más fáciles, intuitivos y utilizados con mayor frecuencia para la reducción de dimensionalidad, ya que proyecta datos en su subespacio de características ortogonales.

La reducción de dimensionalidad es el proceso de reducir el número de variables aleatorias consideradas, mediante la obtención de un conjunto de variables principales. Es una técnica para reducir la dimensionalidad de dichos conjuntos de datos, aumentando la interpretabilidad pero al mismo tiempo minimizando la pérdida de información.

Cada componente generado por el PCA es una combinación lineal de las variables originales, siendo además independientes y no correlacionadas entre si.

El proceso de pca identifica las direcciones con mayor varianza. Se estandarizan los datos evitando que alguna de las variables sea dominante sobre el resto.

```
In [113]: #Inicialmente definimos 6 componentes y el porcentaje de Varianza explicada
pca = PCA(n_components=6)
pca.fit(final_df_escalado)
```

Out[113]:

▼	PCA
PCA(n_components=6)	

In [114]:

```
# =====
print('-----')
print('Porcentaje de varianza explicada por cada componente')
print('-----')
print(pca.explained_variance_ratio_)

plt.style.use('Solarize_Light2')

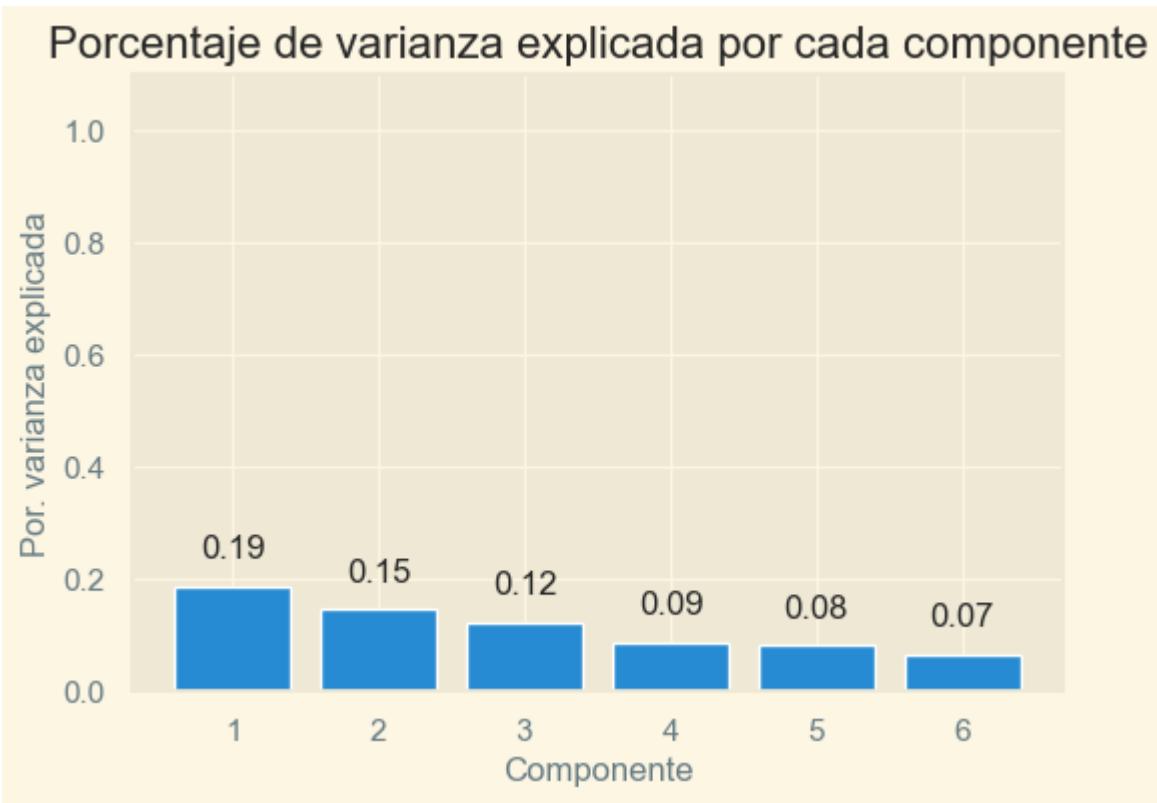
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(6, 4))
ax.bar(
    x      = np.arange(pca.n_components_) + 1,
    height = pca.explained_variance_ratio_
)

for x, y in zip(np.arange(len(final_df_escalado.columns)) + 1, pca.explained_variance_ratio_):
    label = round(y, 2)
    ax.annotate(
        label,
        (x,y),
        textcoords="offset points",
        xytext=(0,10),
        ha='center'
    )

ax.set_xticks(np.arange(pca.n_components_) + 1)
ax.set_xlim(0, 1.1)
ax.set_title('Porcentaje de varianza explicada por cada componente')
ax.set_xlabel('Componente')
ax.set_ylabel('Por. varianza explicada');
```

Porcentaje de varianza explicada por cada componente

[0.1864673 0.14654661 0.1234843 0.08781423 0.08118899 0.0665487]



Reducidos los atributos a tres dimensiones, se realiza la agrupación en clústeres aglomerativos. La agrupación en clústeres aglomerada es un método de agrupación jerárquica. Implica la combinación de ejemplos hasta alcanzar el número deseado de clústeres.

Pasos

1. Método del codo para determinar el número de clústeres que se formarán
2. Agrupación en clústeres a través de la agrupación en clústeres aglomerados
3. Examen de los clústeres formados a través de un diagrama de dispersión

In [127]: #Nuevo

```

from sklearn.metrics import silhouette_samples, silhouette_score
from sklearn.cluster import AgglomerativeClustering
import matplotlib.cm as cm
from sklearn.decomposition import PCA

#aplicando PCA solo con dos componentes principales que seran loso visualizados en 2 dimensiones
pca = PCA(n_components=2)
PCA_2d = pca.fit_transform(final_aglo_escalado)

for n_clusters in range(2, 7):
    # Creando subgrafico de Una fila y Dos columnas
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # La primer caja es la grafica de la silueta
    # El coeficiente de silueta puede oscilar entre -1, 1, pero en este ejemplo tiene sentido
    ax1.set_xlim([-0.1, 1])
    # Los (n_clusters+1)*10 es para insertar un espacio en blanco entre cada silueta
    # Grafica los cluster individuales para demarcar individualmente
    ax1.set_ylim([0, len(final_aglo_escalado) + (n_clusters) * 10])

    # Inicializa el clúster con el valor del iterativo de n_clusters y un generador de semilla de 10 para la reproducibilidad.
    clusterer = AgglomerativeClustering(n_clusters=n_clusters, affinity='euclidean')
    cluster_labels = clusterer.fit_predict(final_aglo_escalado)

    # El silhouette_score da el valor medio de todas las muestras.
    # Esto da una perspectiva de la densidad y separación de los cúmulos formados
    silhouette_avg = silhouette_score(final_aglo_escalado, cluster_labels)
    print("Para ", n_clusters,
          " grupos la media del silhouette_score es :", silhouette_avg)

    # Calcular las puntuaciones de silueta para cada muestra
    sample_silhouette_values = silhouette_samples(final_aglo_escalado, cluster_labels)

    y_lower = 10

    for i in range(n_clusters):
        # Se Agregan las puntuaciones de silueta de las muestras que pertenecen a ese cluster
        ith_cluster_silhouette_values = \
            sample_silhouette_values[cluster_labels == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i+1) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

```

```

# Etiqueta Los gráficos de silueta con sus números de clúster en el centro
ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

# Calcular el nuevo y_lower para el siguiente trazado
y_lower = y_upper + 10 # 10 para las 0 muestras

ax1.set_title("Gráfico de silueta para los distintos clústeres.")
ax1.set_xlabel("coeficiente de silueta")
ax1.set_ylabel("Etiqueta de clúster")

# La Línea vertical para La puntuación media de la silueta de todos los valores
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([]) # Borrar las etiquetas / marcas yaxis
ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

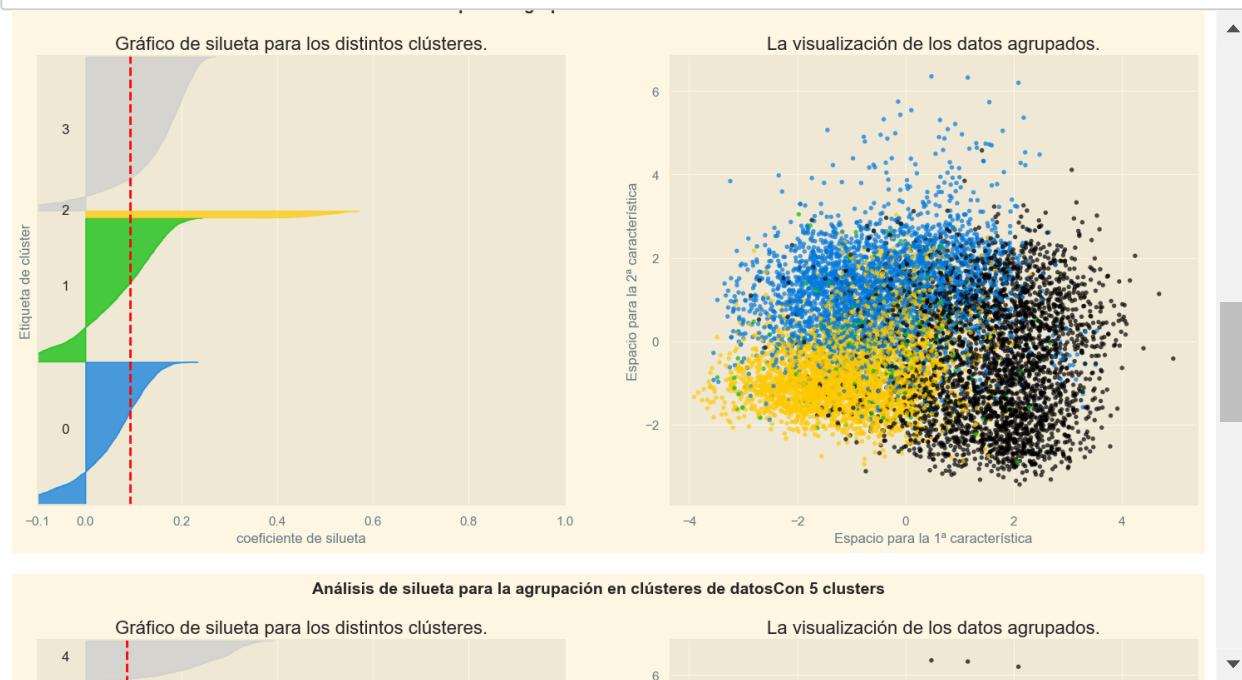
# 2º Gráfico que muestra los grupos reales formados
colors = cm.nipy_spectral((cluster_labels.astype(float)) / n_clusters)
ax2.scatter(PCA_2d[:, 0], PCA_2d[:, 1], marker='.', s=60, lw=0, alpha=0.7,
            c=colors, edgecolor='k')

ax2.set_title("La visualización de los datos agrupados.")
ax2.set_xlabel("Espacio para la 1ª característica")
ax2.set_ylabel("Espacio para la 2ª característica")

plt.suptitle(("Análisis de silueta para la agrupación en clústeres de datos"
              "Con %d clusters" % n_clusters),
             fontsize=14, fontweight='bold')

plt.show()

```



```
In [ ]: #Definimos 3 componentes para los datos completos
pca = PCA(n_components=3, random_state = 42)
pca.fit(final_df_escalado)

PCA_dataset = pd.DataFrame(pca.transform(final_df_escalado), columns=[["Com1", "Co
```

```
In [131]: #Definimos 3 componentes para el dataset para aplicar tecnica de agrupación aglo
pca_aglo = PCA(n_components=3, random_state = 42)
pca_aglo.fit(final_aglo_escalado)

PCA_dataset_2 = pd.DataFrame(pca_aglo.transform(final_aglo_escalado), columns=[['
```

```
In [132]: # Se convierte el array a dataframe para añadir nombres a los ejes.
pca_components = pd.DataFrame(
    data = pca_aglo.components_,
    columns = final_df_escalado.columns,
    index = ['Com1', 'Com2', 'Com3']
).T
pca_components
```

Out[132]:

	Com1	Com2	Com3
transacciones	0.185701	0.054242	0.059959
edad	-0.440175	0.284573	-0.224314
antiguedad	0.110974	0.603188	0.002292
apertura	0.128277	0.591547	0.012517
ocupacion	-0.408321	0.125506	-0.350331
tipo_vivienda	-0.170413	0.228903	-0.087091
genero	-0.181165	-0.145862	-0.148224
no_personas_a_cargo	0.207762	0.173699	0.249758
valor_mvto	-0.404690	0.014097	0.564099
valor_saldo	0.210947	0.075214	0.118142
cant_productos	-0.028306	-0.015342	-0.012805
vl_compra_cat	-0.405602	0.012686	0.562323
Con_Pareja	0.085819	0.260690	0.180873
Educacion	0.300478	-0.092227	0.223625

In [133]:

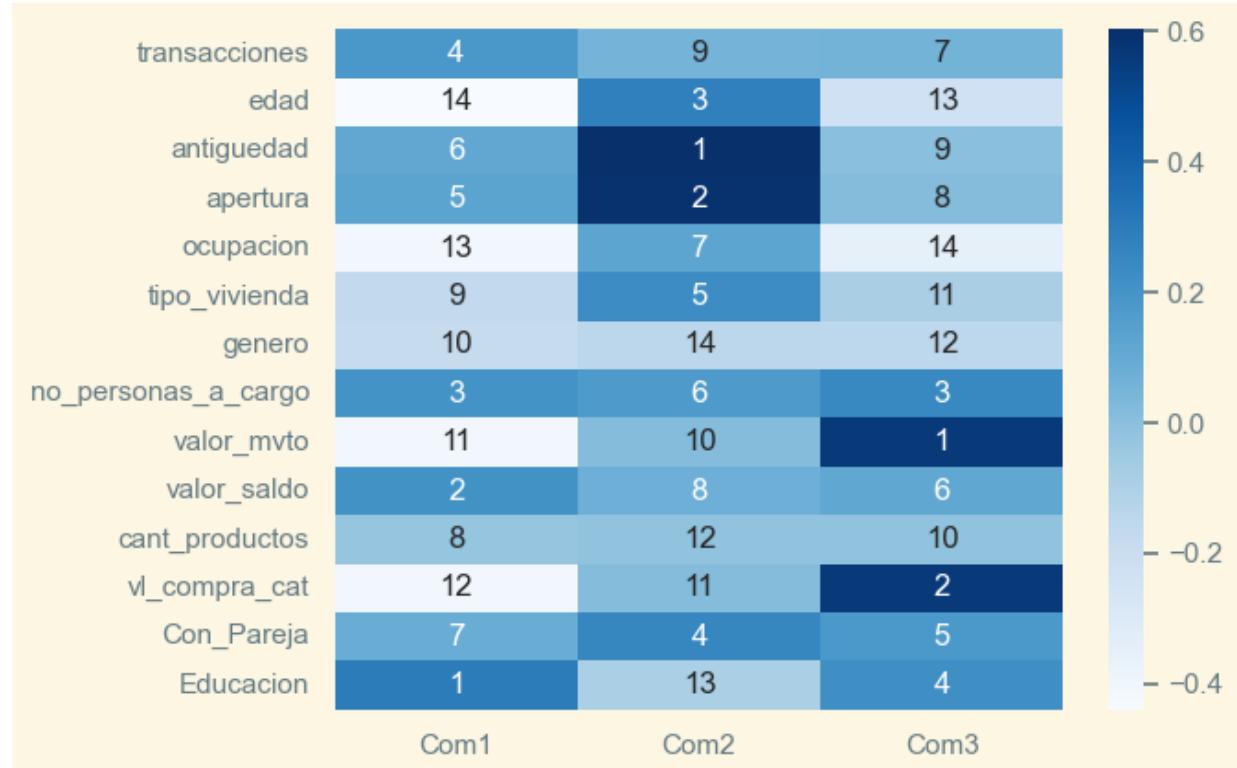
"""

Los pesos asignados en la primer componente a las variables Edad, antiguedad y ap
mayoritariamente información correspondiente al tiempo de antiguedad en la entic
En el segundo componente refleja mayor peso con la informacion relacionada a la c
si tiene pareja el cliente. Finalmente el componente 3 corresponde principalmente

"""

```
sns.heatmap(pca_components, annot=pca_components.rank(axis="rows", ascending = False), center=0)
```

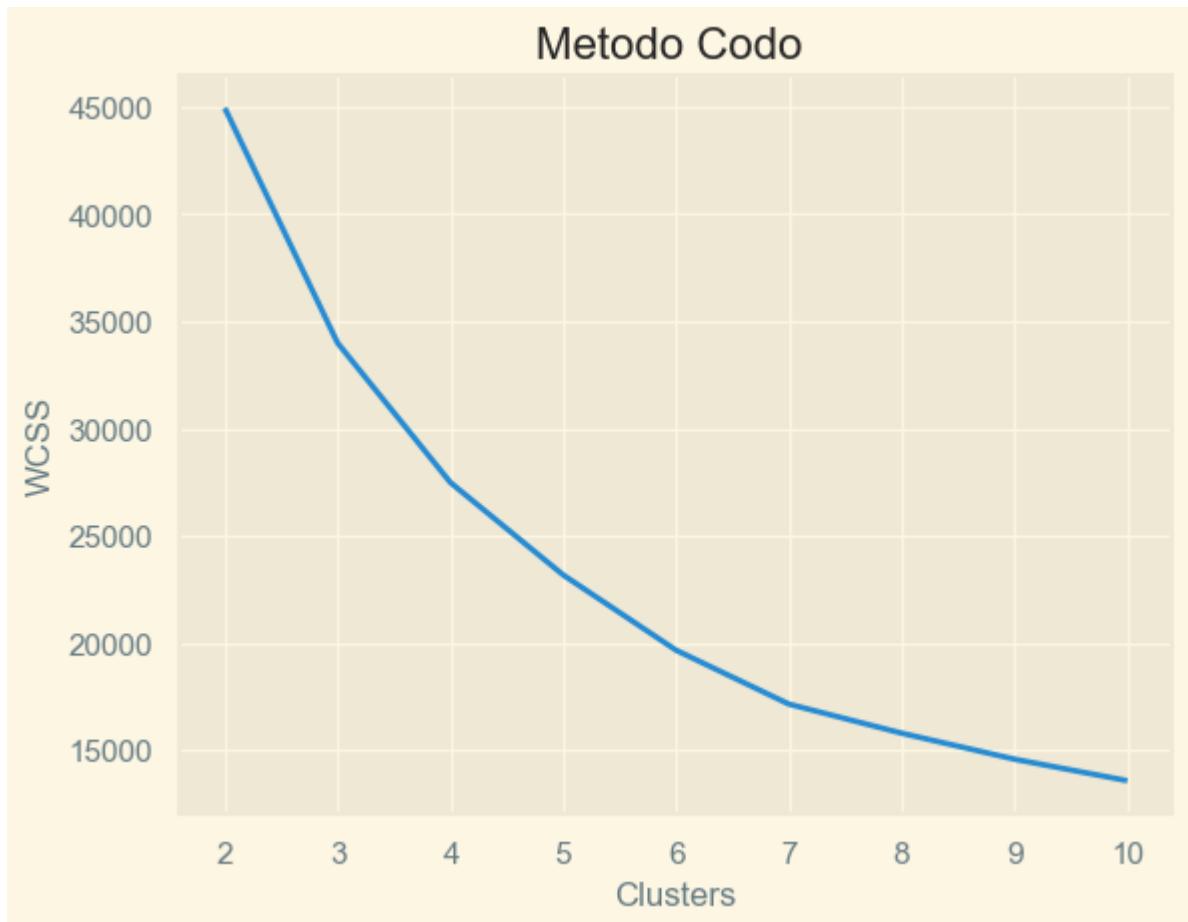
Out[133]: <Axes: >



In [134]: # Encontrar el número de clusters

```
from sklearn.cluster import KMeans
wcss = []
for i in range(2, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 12)
    kmeans.fit(PCA_dataset_2)
    wcss.append(kmeans.inertia_)
```

```
In [135]: import matplotlib.pyplot as plt
plt.plot(range(2, 11), wcss)
plt.title('Metodo Codo ')
plt.xlabel('Clusters')
plt.ylabel('WCSS')
plt.show()
```



La celda anterior indica que cuatro será un número óptimo de clústeres para estos datos. A continuación, ajustaremos el modelo de agrupación en clústeres aglomerados para obtener los clústeres finales.

```
In [136]: #Se inicializa El modelo de Clusterización Aglomerativa
C_Agl = AgglomerativeClustering(n_clusters=4)

# Se ajusta el modelo y Se predicen los grupos
Aglo = C_Agl.fit_predict(PCA_dataset_2)
PCA_dataset_2["Clusters"] = Aglo

#Adiciona el Cluster al DATaset Original
final_aglo["Clusters"] = Aglo
```

In [138]: final_aglo

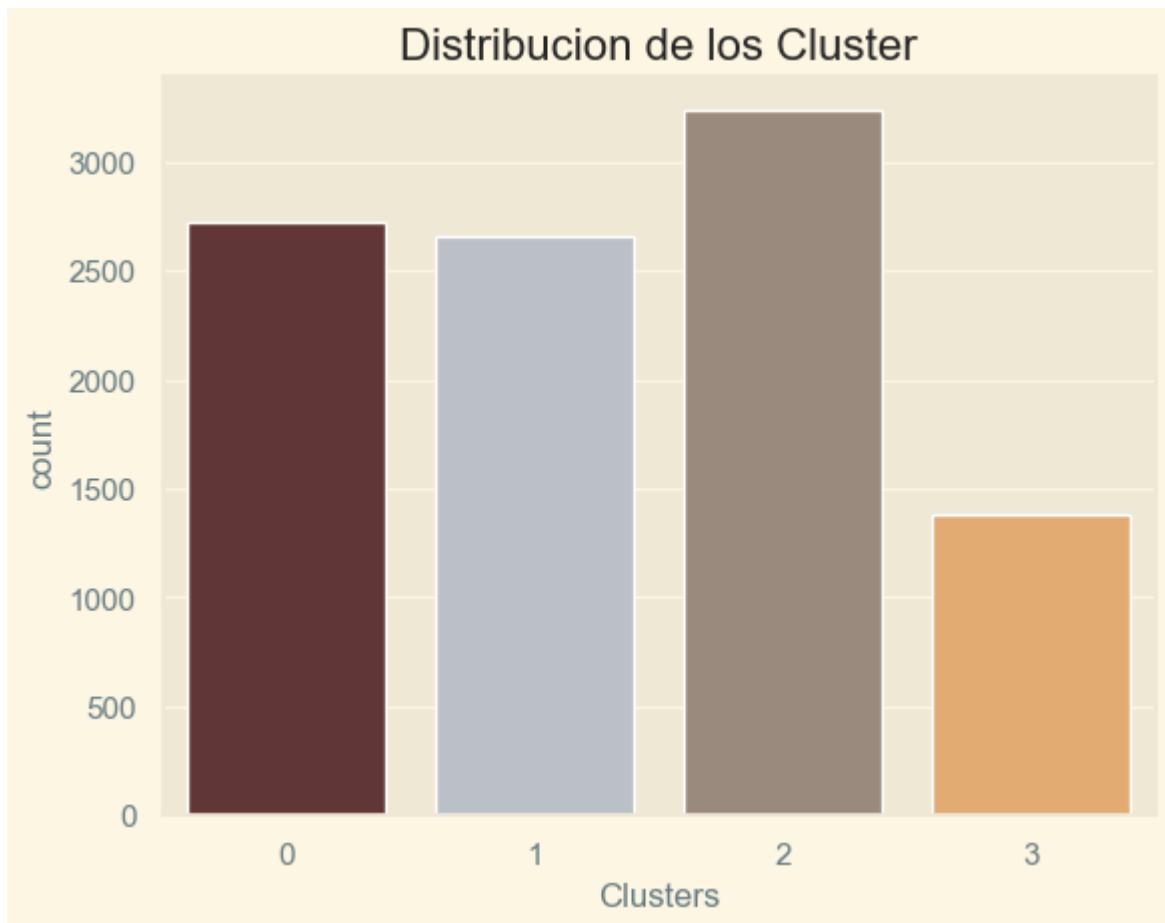
Out[138]:

lad	apertura	ocupacion	tipo_vivienda	genero	no_personas_a_cargo	valor_mvto	valor_saldo	c...
6.0	26.0	6	2	1		2.0	309950.0	184304.0
2.0	12.0	3	0	1		1.0	27500.0	307678.0
7.0	17.0	0	2	2		1.0	32333.0	209667.0
5.0	22.0	6	4	2		0.0	258667.0	154333.0
4.0	24.0	0	4	2		0.0	450000.0	194878.0
...
6.0	16.0	6	4	2		0.0	510000.0	55292.0
9.0	19.0	6	4	1		2.0	100000.0	304509.0
7.0	7.0	6	0	2		1.0	215000.0	49538.0
5.0	5.0	6	4	2		0.0	286667.0	89516.0
7.0	7.0	6	0	2		0.0	15137.0	89636.0

El propósito de esta sección es estudiar los patrones en los conglomerados formados y determinar la naturaleza de los patrones de los conglomerados.

In [139]: #Grafico con Distribucion de Los Cluster

```
pal = ["#682F2F", "#B9C0C9", "#9F8A78", "#F3AB60"]
pl = sns.countplot(x=final_aglo["Clusters"], palette= pal)
pl.set_title("Distribucion de los Cluster")
plt.show()
```



In [176]: `final_aglo.head()`

Out[176]:

SK_CLIENTE	transacciones	edad	antiguedad	apertura	ocupacion	tipo_vivienda	genero	no_per
68937	2	80.0	26.0	26.0	6	2	1	
80196	2	38.0	12.0	12.0	3	0	1	
80399	3	43.0	17.0	17.0	0	2	2	
80476	3	82.0	25.0	22.0	6	4	2	
80488	2	65.0	24.0	24.0	0	4	2	



In [191]: #Nuevo

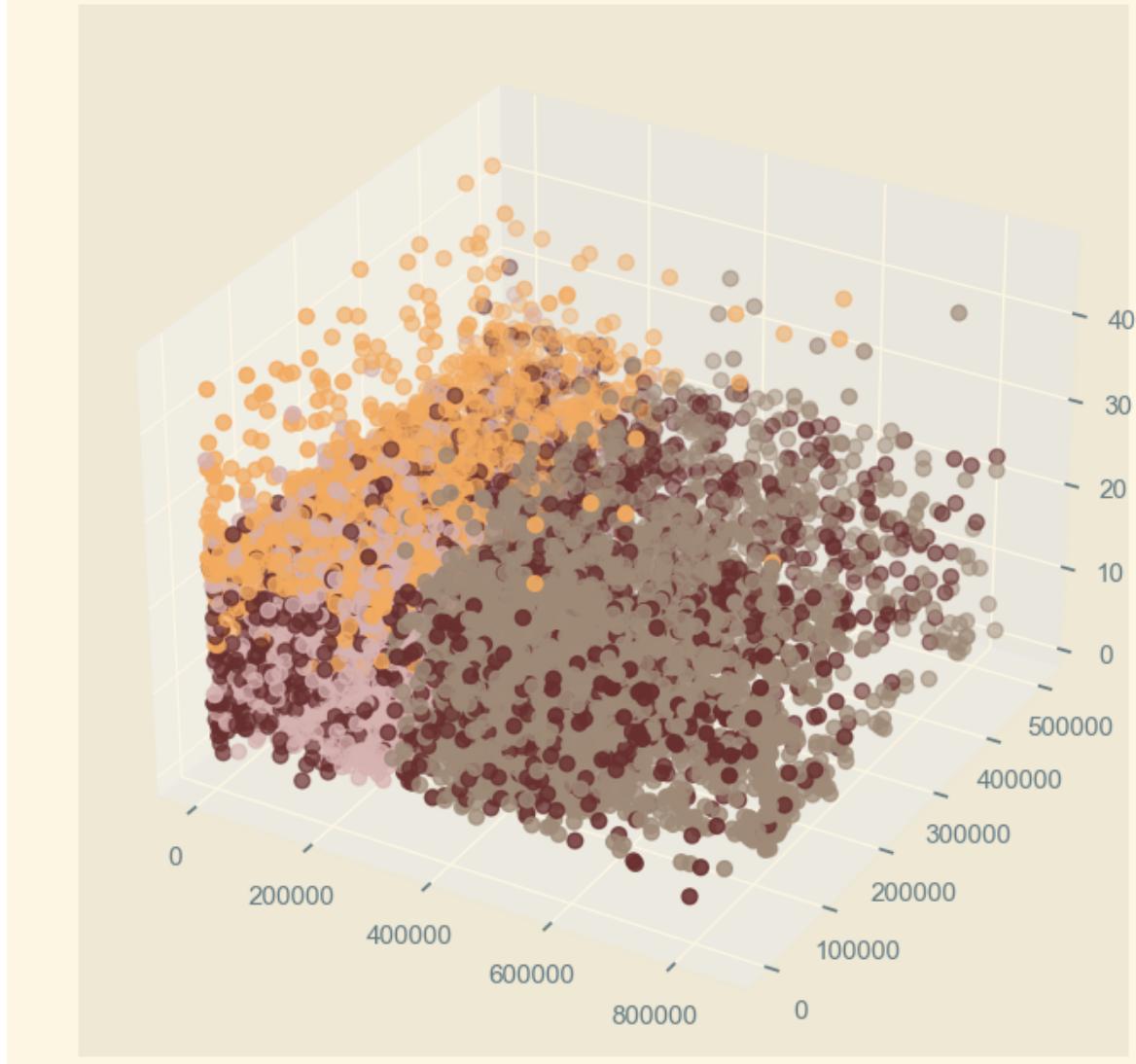
```
from matplotlib import colors

x = final_aglo["valor_mvto"]
y = final_aglo["valor_saldo"]
z = final_aglo["antiguedad"]

cmap = colors.ListedColormap(["#682F2F", "#9E726F", "#D6B2B1", "#B9C0C9", "#9F8A7B"])

fig = plt.figure(figsize=(10,8))
ax = plt.subplot(111, projection='3d', label="bla")
ax.scatter(x, y, z, s=40, c=final_aglo["Clusters"], marker='o', cmap = cmap)
ax.set_title("Grupos definidos para X=valor_mvto - Y=valor_saldo - Z=antiguedad")
plt.show()
```

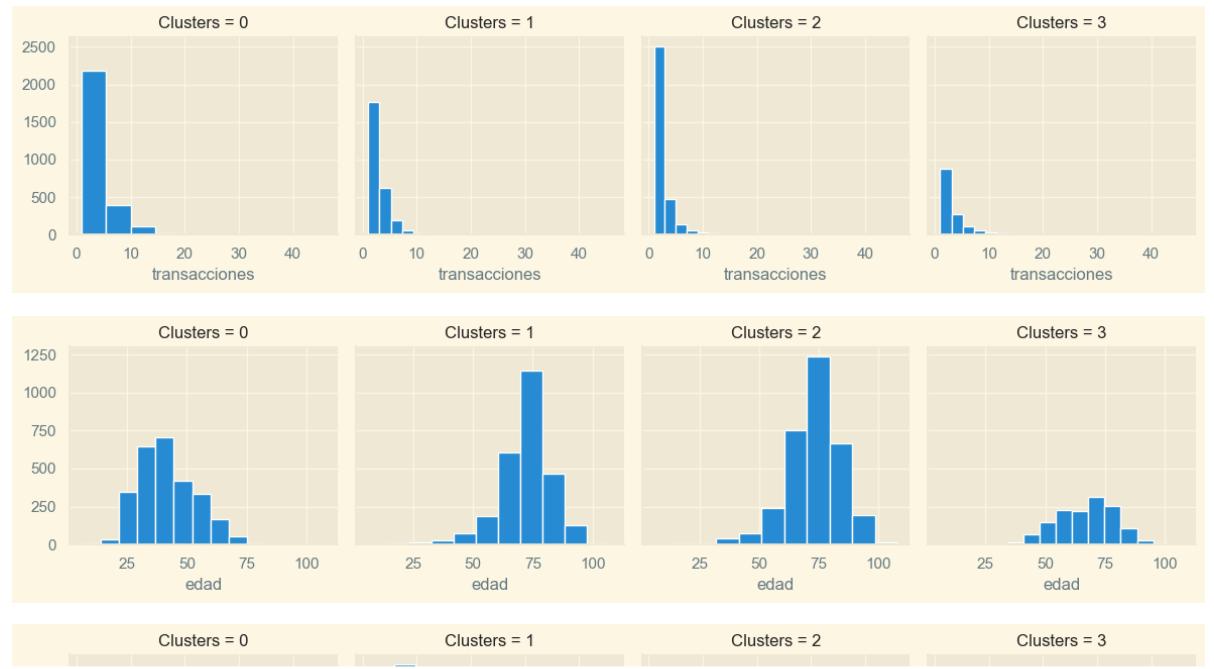
Grupos definidos para X=valor_mvto - Y=valor_saldo - Z=antiguedad



In [196]: #Nuevo

#Caracteriza Variable por cluster

```
for c in final_aglo:
    grid= sns.FacetGrid(final_aglo, col='Clusters')
    grid.map(plt.hist, c)
```



In [199]: description = final_aglo.groupby("Clusters")[["edad", "antiguedad", "valor_mvto", "n_clients = description.size()"]
description = description.mean()
description['n_clients'] = n_clients
description

Out[199]:

	edad	antiguedad	valor_mvto	valor_saldo	n_clients
Clusters					
0	41.702425	13.448935	248986.898971	218467.845702	2722
1	72.172427	11.839594	154457.229151	138351.730278	2662
2	72.698084	13.798517	501926.398949	150712.775958	3236
3	67.007971	24.040580	114220.660145	205610.215942	1380

In [200]: `final_aglo.groupby("Clusters")["valor_mvto"].describe()`

Out[200]:

	count	mean	std	min	25%	50%	75%	max
Clusters								
0	2722.0	248986.898971	217602.348624	12.0	72204.5	190242.0	380000.0	855950.0
1	2662.0	154457.229151	91548.605968	58.0	73339.5	168170.0	226107.0	301289.0
2	3236.0	501926.398949	152376.708561	235000.0	400000.0	455000.0	600000.0	857002.0
3	1380.0	114220.660145	104325.249904	53.0	23438.0	86629.5	198290.0	830000.0

Grupo 0: Personas con alto nivel de ingresos (saldo) y alto límite de crédito que toman efectivo por adelantado.

Grupo 1: Personas con bajo nivel de ingresos. Compras no frecuentes.

Grupo 2: Saldo bajo, pero el saldo se actualiza con frecuencia, es decir. Más no. de las transacciones. Compran mayoritariamente a plazos

Clúster 3 Compran principalmente de una sola vez con una alta frecuencia. El porcentaje del pago total pagado es bajo (deudores).

In [193]: `final_aglo`

Out[193]:

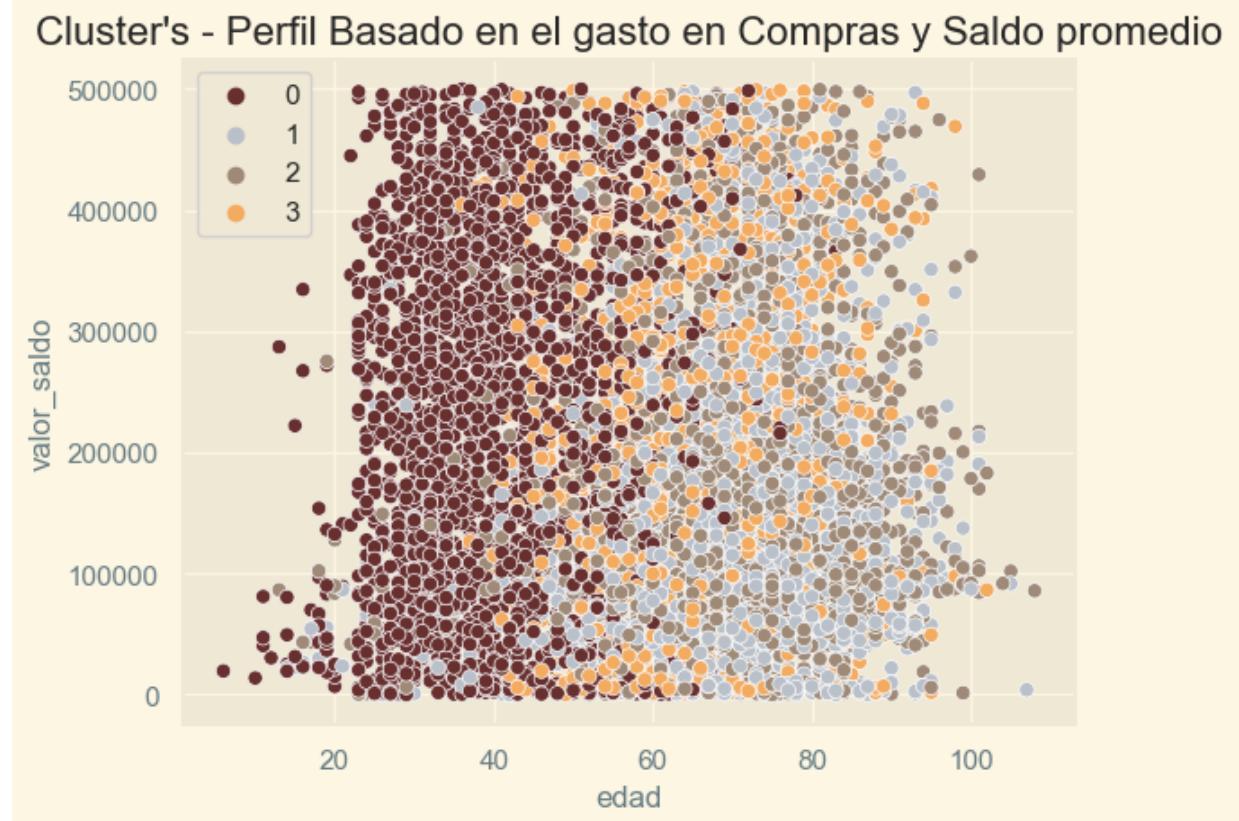
	transacciones	edad	antiguedad	apertura	ocupacion	tipo_vivienda	genero	no_per
SK_CLIENTE								
68937	2	80.0	26.0	26.0	6	2	1	
80196	2	38.0	12.0	12.0	3	0	1	
80399	3	43.0	17.0	17.0	0	2	2	
80476	3	82.0	25.0	22.0	6	4	2	
80488	2	65.0	24.0	24.0	0	4	2	
...
660298	1	62.0	16.0	16.0	6	4	2	
660302	2	43.0	19.0	19.0	6	4	1	
660387	2	50.0	7.0	7.0	6	0	2	
660390	3	77.0	5.0	5.0	6	4	2	
660556	1	73.0	7.0	7.0	6	0	2	

10000 rows × 15 columns



In [143]: #Nuevo

```
pl = sns.scatterplot(data = data,x=final_aglo["edad"], y=final_aglo["valor_saldo"]
pl.set_title("Cluster's - Perfil Basado en el gasto en Compras y Saldo promedio")
plt.legend()
plt.show()
```



In [153]: #Nuevo

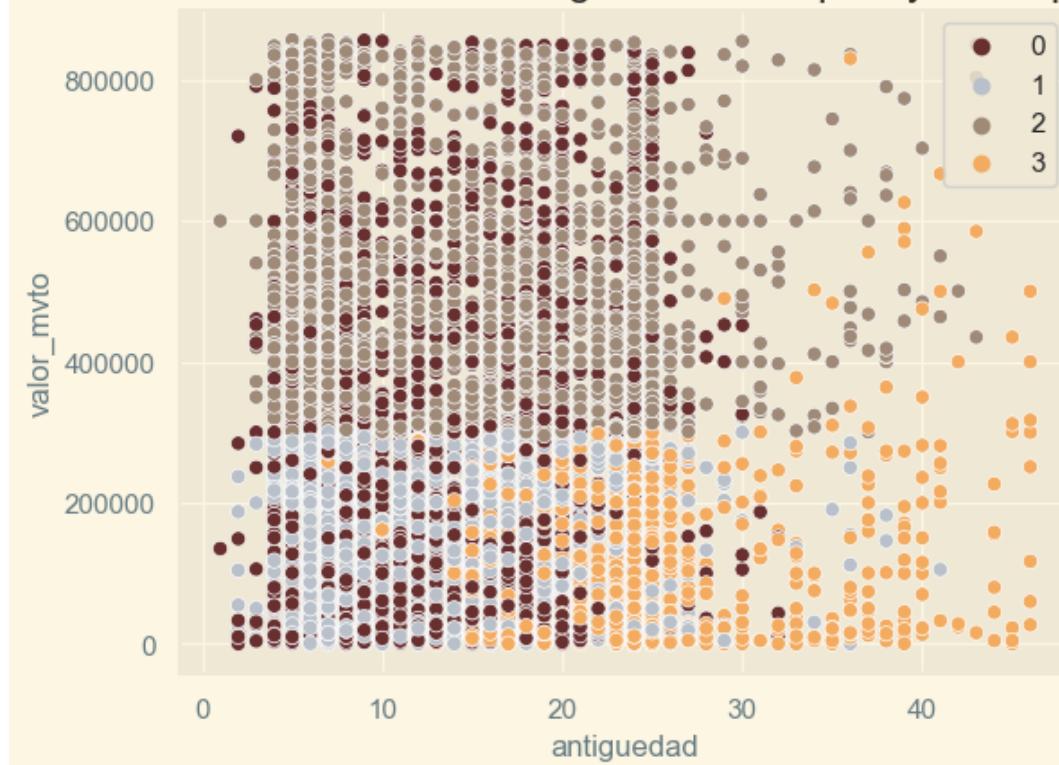
"""

El grupo 1 esta definido por clientes con una antiguedad entreo los 0 y 20 años y
El grupo 2 esta definido por clientes con una antiguedad entreo los 0 y 20 años y
El grupo 3 esta definido por clientes con mas de 15 años en la entidad y compras

"""

```
pl = sns.scatterplot(data = data,x=final_aglo["antiguedad"], y=final_aglo["valor_mvt0"])
pl.set_title("Cluster's - Perfil Basado en el gasto en Compras y antiguedad")
plt.legend()
plt.show()
```

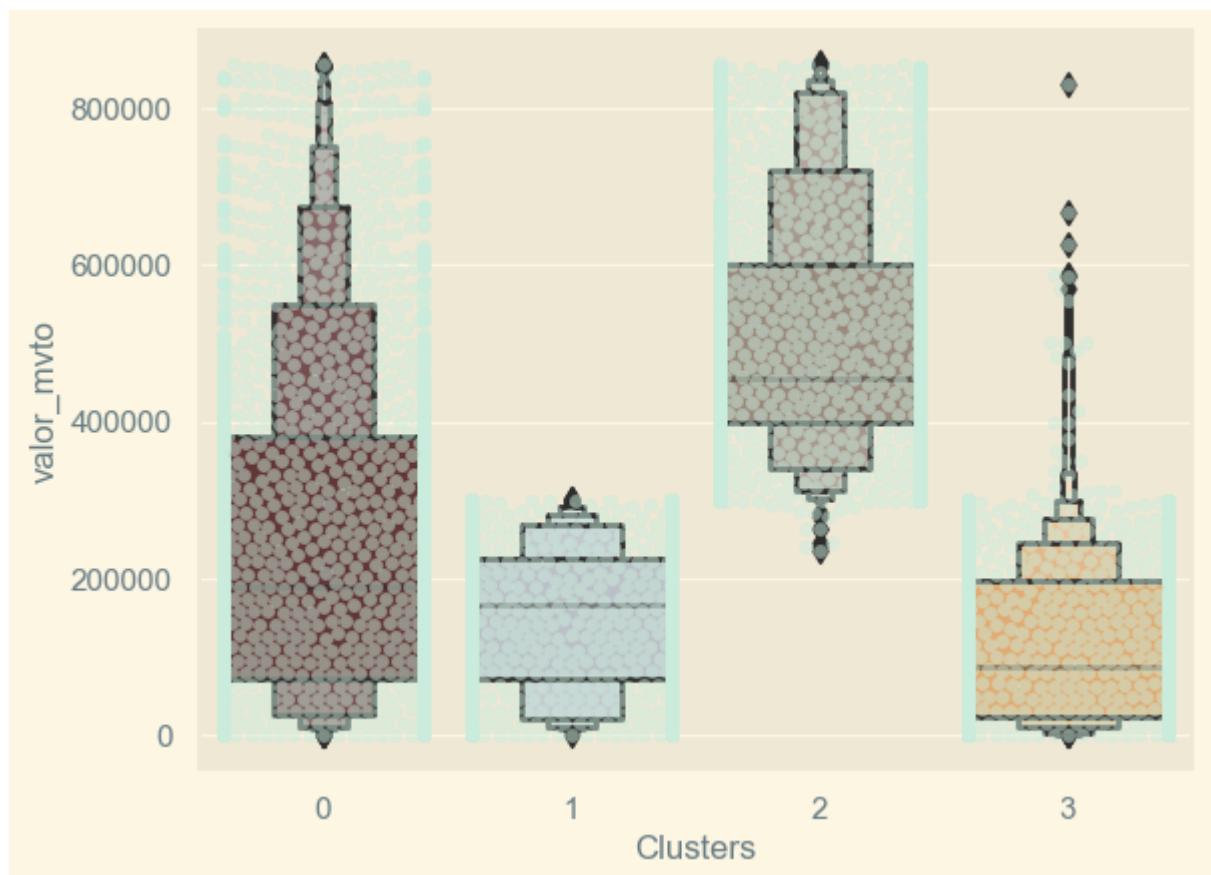
Cluster's - Perfil Basado en el gasto en Compras y Saldo promedio



Se caracteriza el promedio de compras alto para los grupos 0 y 1, sin embargo el promedio del saldo aunque es muy disperso tiene sus valores mas bajos en el grupo 1 y 3.

In [144]:

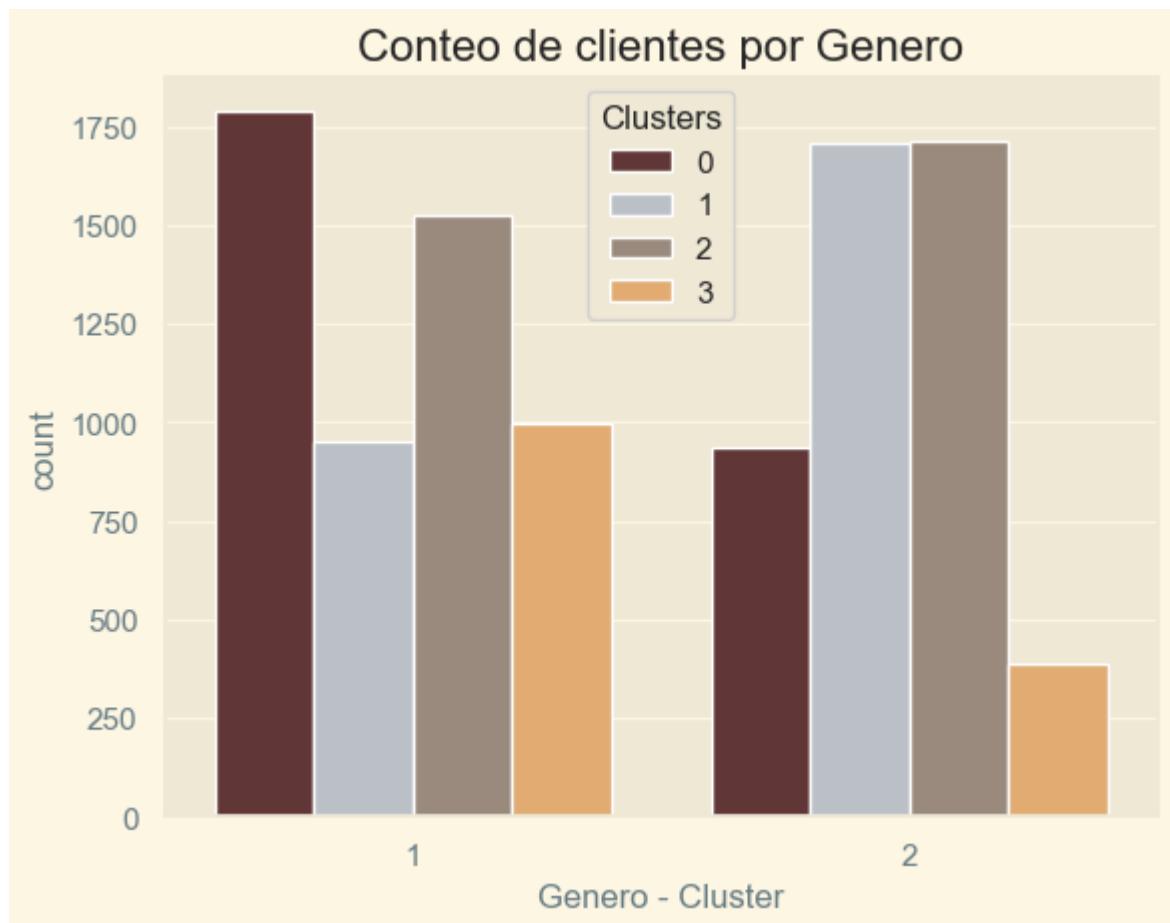
```
plt.figure()
pl=sns.swarmplot(x=final_aglo["Clusters"], y=final_aglo["valor_mvto"], color= "#C8E6C9")
pl=sns.boxenplot(x=final_aglo["Clusters"], y=final_aglo["valor_mvto"], palette=pa
plt.show()
```



En el gráfico anterior, se puede observar que el grupo "0" seguido del grupo "1" es el conjunto de clientes con gasto promedio más alto. De esta manera se puede explorar en qué está gastando cada grupo para las estrategias de marketing dirigidas, recomendar productos o promociones manejadas por la entidad enfocadas a este tipo de gastos.

In [145]: #El Cluster con mayor gasto esta dominado por los de sexo masculino

```
plt.figure()
pl = sns.countplot(x=final_aglo["genero"],hue=final_aglo["Clusters"], palette= pa
pl.set_title("Conteo de clientes por Genero")
pl.set_xlabel("Genero - Cluster")
plt.show()
```



In [154]: final_aglo

Out[154]:

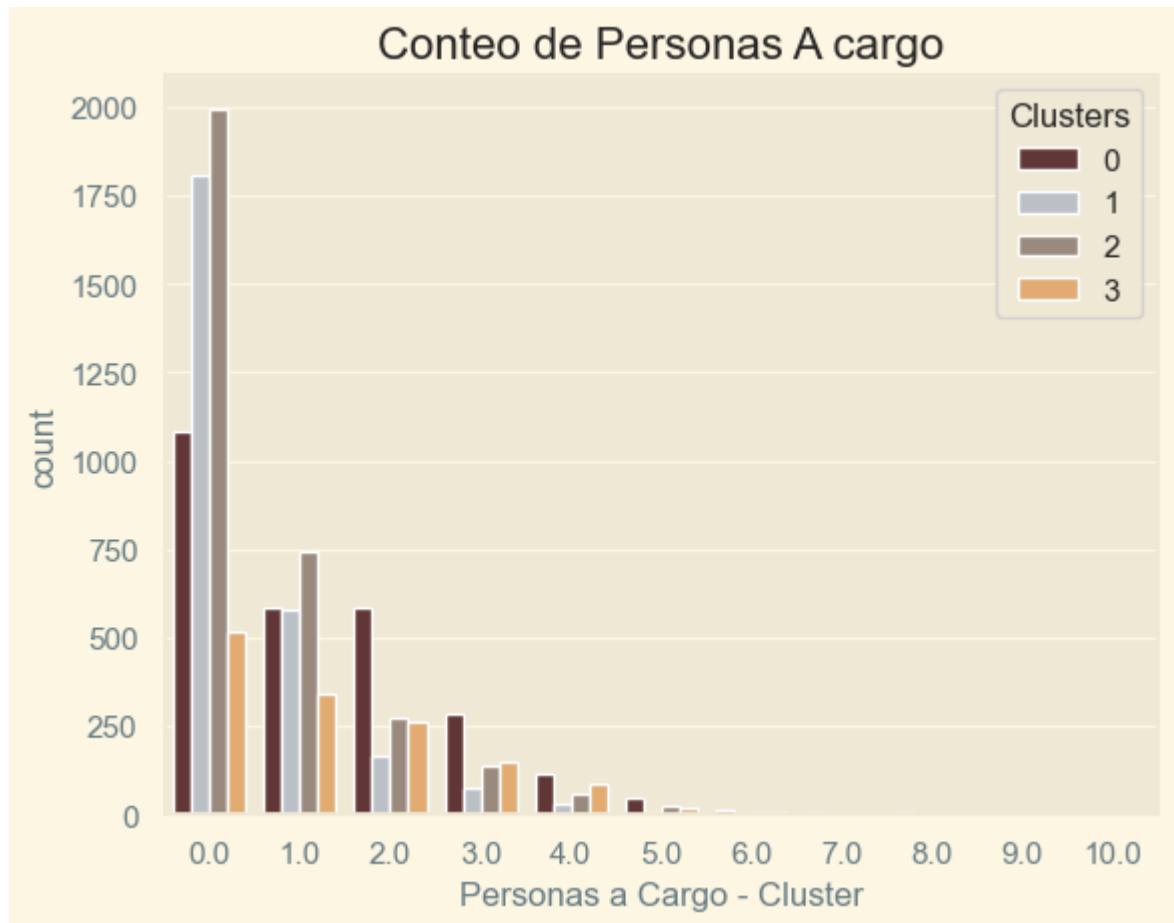
ad	apertura	ocupacion	tipo_vivienda	genero	no_personas_a_cargo	valor_mvto	valor_saldo	cant
6.0	26.0	6	2	1		2.0	309950.0	184304.0
2.0	12.0	3	0	1		1.0	27500.0	307678.0
7.0	17.0	0	2	2		1.0	32333.0	209667.0
5.0	22.0	6	4	2		0.0	258667.0	154333.0
4.0	24.0	0	4	2		0.0	450000.0	194878.0
...
6.0	16.0	6	4	2		0.0	510000.0	55292.0
9.0	19.0	6	4	1		2.0	100000.0	304509.0
7.0	7.0	6	0	2		1.0	215000.0	49538.0
5.0	5.0	6	4	2		0.0	286667.0	89516.0
7.0	7.0	6	0	2		0.0	15137.0	89636.0



In [146]:

```
# Es mayor el gasto en clientes sin personas a cargo
```

```
plt.figure()
pl = sns.countplot(x=final_aglo["no_personas_a_cargo"],hue=final_aglo["Clusters"])
pl.set_title("Conteo de Personas A cargo")
pl.set_xlabel("Personas a Cargo - Cluster")
plt.show()
```

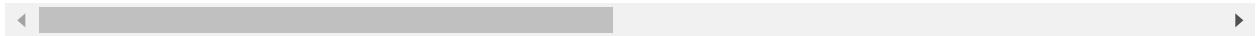


In [135]: final_aglo

Out[135]:

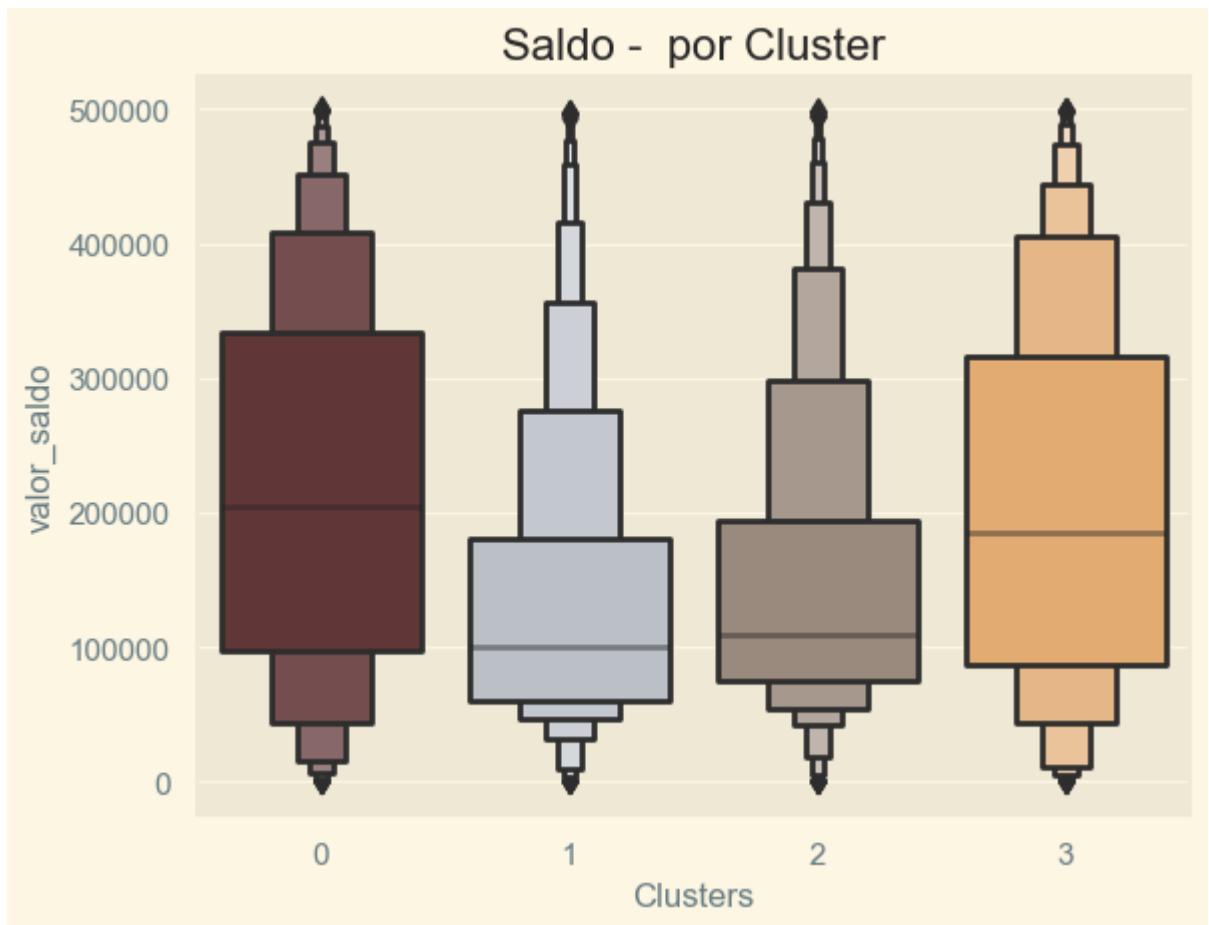
SK_CLIENTE	transacciones	edad	antiguedad	apertura	ocupacion	tipo_vivienda	genero	no_per
68937	2	80.0	26.0	26.0	6	2	1	
80196	2	38.0	12.0	12.0	3	0	1	
80399	3	43.0	17.0	17.0	0	2	2	
80476	3	82.0	25.0	22.0	6	4	2	
80488	2	65.0	24.0	24.0	0	4	2	
...
660294	1	65.0	17.0	17.0	6	4	1	
660298	1	62.0	16.0	16.0	6	4	2	
660302	2	43.0	19.0	19.0	6	4	1	
660387	2	50.0	7.0	7.0	6	0	2	
660390	3	77.0	5.0	5.0	6	4	2	

10000 rows × 15 columns



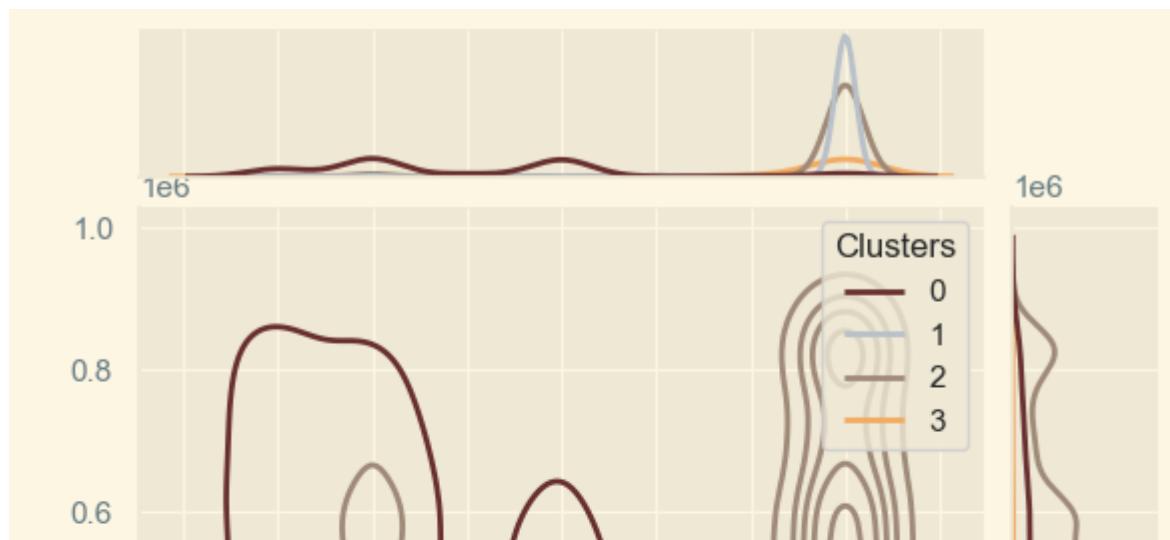
In [147]: #El saldo Promedio mas alto esta definido en el cluster "0"

```
plt.figure()
pl=sns.boxenplot(y=final_aglo["valor_saldo"],x=final_aglo["Clusters"], palette=paleta)
pl.set_title("Saldo - por Cluster")
plt.show()
```



```
In [148]: Personal = [ "edad", "ocupacion", "genero", "edad", "tipo_vivienda", "no_personas_a_cuidar"]

for i in Personal:
    plt.figure()
    sns.jointplot(x=final_aglo[i], y=final_aglo["valor_mvto"], hue=final_aglo["categoría"])
    plt.show()
```



In [171]: #Nuevo

```
#Evaluacion de Metricas INternas para segmentacion de La data Limitada a 10 mil c

from sklearn import metrics
from sklearn import datasets
import pandas as pd
from sklearn.cluster import KMeans, AgglomerativeClustering, AffinityPropagation, SpectralClustering

data = datasets.load_digits()
X, y = df_aglo_SC, Aglo

algorithms = []
algorithms.append(KMeans(n_clusters=4, random_state=1))
algorithms.append(AffinityPropagation())
algorithms.append(SpectralClustering(n_clusters=4, random_state=1,
                                      affinity='nearest_neighbors'))
algorithms.append(AgglomerativeClustering(n_clusters=4))

data = []
for algo in algorithms:
    algo.fit(X)
    data.append({
        'ARI': metrics.adjusted_rand_score(y, algo.labels_),
        'AMI': metrics.adjusted_mutual_info_score(y, algo.labels_),
        'Homogeneity': metrics.homogeneity_score(y, algo.labels_),
        'Completeness': metrics.completeness_score(y, algo.labels_),
        'V-measure': metrics.v_measure_score(y, algo.labels_),
        'Silhouette': metrics.silhouette_score(X, algo.labels_)})

results = pd.DataFrame(data=data, columns=[ 'ARI', 'AMI', 'Homogeneity',
                                             'Completeness', 'V-measure',
                                             'Silhouette'],
                           index=[ 'K-means', 'Affinity',
                                   'Spectral', 'Agglomerative'])

results
```

Out[171]:

	ARI	AMI	Homogeneity	Completeness	V-measure	Silhouette
K-means	0.214237	0.262698	0.262380	0.263514	0.262946	0.406325
Affinity	0.004738	0.075195	0.740921	0.130724	0.222238	0.126939
Spectral	0.148330	0.210887	0.211989	0.210318	0.211150	0.379421
Agglomerative	0.188491	0.246203	0.248894	0.244060	0.246453	0.327345

```
In [159]: from sklearn import metrics
from sklearn import datasets
import pandas as pd
from sklearn.cluster import KMeans, AgglomerativeClustering, AffinityPropagation,
```



```
data = datasets.load_digits()
X, y = data.data, data.target
```

```
In [170]: = final_aglo.drop(columns=['Clusters'], errors='ignore')
```

```
In [169]: .shape
```

```
Out[169]: (10000,)
```

HASTA AQUI NO MAS.....

```
In [102]: kmeans = KMeans(n_clusters=4, random_state = 42)

Kmeans_clusters = kmeans.fit_predict(PCA_dataset)
PCA_dataset['Clusters'] = Kmeans_clusters

# Se agregarán los cluster sl dataset original
PCA_dataset['Clusters'] = Kmeans_clusters
```

```
In [ ]:
```

In [103]: PCA_dataset

Out[103]:

	PC1	PC2	PC3	Clusters
0	-0.178640	2.437408	-0.416915	0
1	2.934107	0.664009	0.954725	0
2	2.315459	1.447683	0.555481	0
3	-1.660088	1.385579	-0.385512	3
4	-1.373565	1.878523	0.004612	3
...
231579	-1.148087	-0.608498	-1.971444	2
231580	0.847839	2.383557	-0.801905	0
231581	-1.881830	0.177744	0.214942	3
231582	1.420540	1.384701	-1.941324	2
231583	-2.124412	-0.272248	0.199191	3

231584 rows × 4 columns

Type *Markdown* and *LaTeX*: α^2

In [76]:

```
# Creamos el codificador indicandole el orden de la variables
encoder = OrdinalEncoder(categories=[['-', 'Sin estudio', 'Primaria', 'Secundaria',
                                         'Tecnologico', 'Profesional', 'Especializaci
                                         'Doctorado']])
```

```
# Ajustamos el codificador con la variable nivel_estudios y la transformamos
encoder.fit(df[["nivel_estudios"]])
df[["nivel_estudios"]] = encoder.transform(df[["nivel_estudios"]])
```

In [77]:

```
encoder2 = OrdinalEncoder(categories=[['-', 'Hipotecada', 'Arrendada', 'Familiar',
                                         # Ajustamos el codificador con la variable tipo_vivienda y la transformamos
                                         encoder2.fit(df[["tipo_vivienda"]])
                                         df[["tipo_vivienda"]] = encoder2.transform(df[["tipo_vivienda"]])
```

In [78]:

```
df['nivel_estudios'].unique()
```

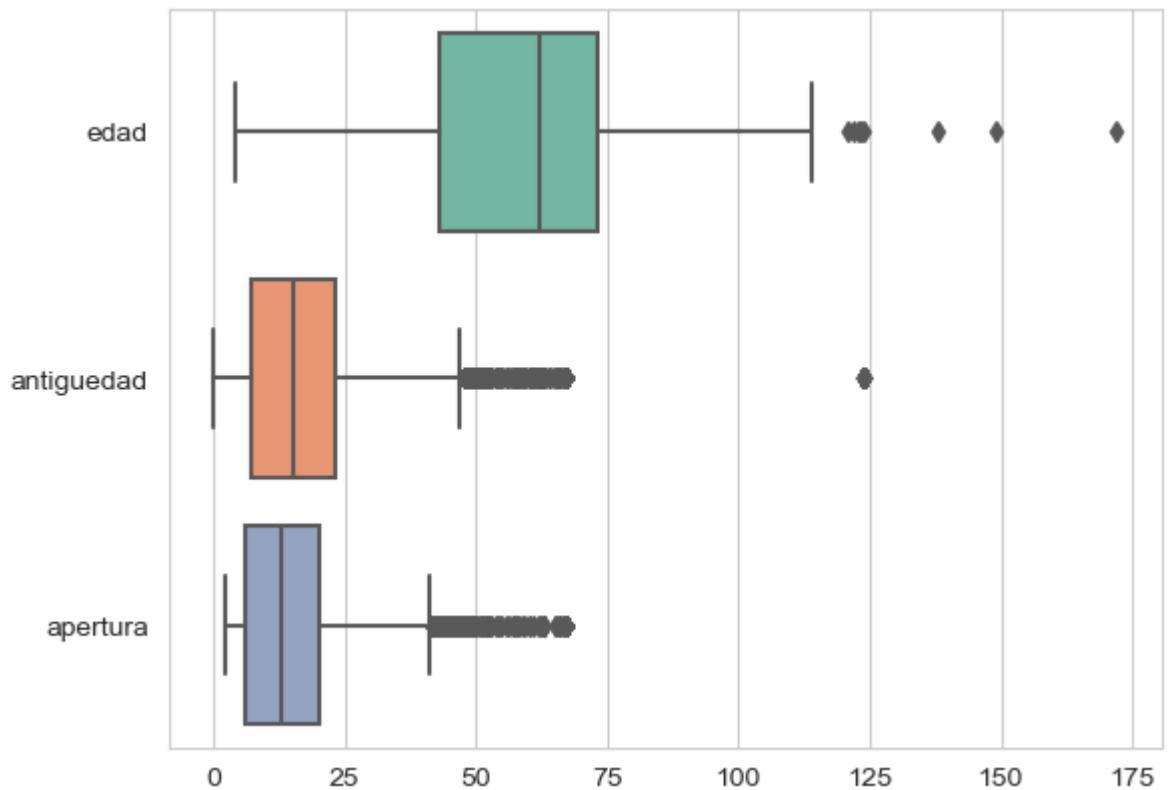
Out[78]:

```
array([6., 2., 4., 1., 7., 0., 9., 5., 3., 8.])
```

In [79]:

```
# Visualizacion de Edad con graficos de Caja
dt_cajas = df[['edad', 'antiguedad', 'apertura']]
```

In [80]: # Obteniendo valores de distribucion, mediana, rango intercuartilico, outliers c
sns.boxplot(data=dt_cajas, orient='h', palette='Set2')
plt.show()



In [81]: df.info()

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 321166 entries, 61862 to 4560024
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   transacciones    321166 non-null   int64  
 1   edad              321166 non-null   float64 
 2   antiguedad       321166 non-null   float64 
 3   apertura          321166 non-null   float64 
 4   pais               321166 non-null   object  
 5   ocupacion          321166 non-null   object  
 6   tipo_vivienda     321166 non-null   float64 
 7   estado_civil      321166 non-null   object  
 8   genero             321166 non-null   int64  
 9   nivel_estudios    321166 non-null   float64 
 10  no_personas_a_cargo 321166 non-null   float64 
 11  valor_mvto         321166 non-null   float64 
 12  cant_productos    321166 non-null   int64  
 13  vl_compra_cat     321078 non-null   category 
dtypes: category(1), float64(7), int64(3), object(3)
memory usage: 34.6+ MB
```

In [84]: #df.loc[61862, ['valor_mvto']]

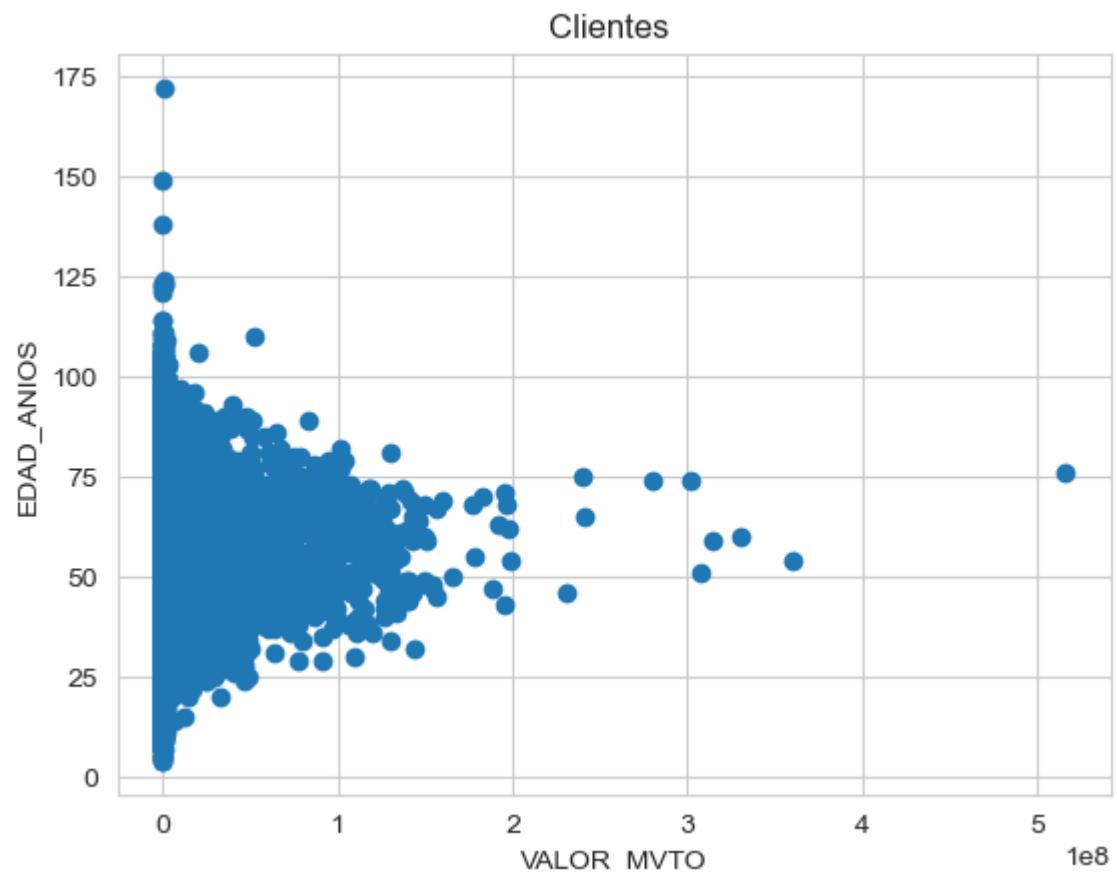
In [82]: df

	transacciones	edad	antiguedad	apertura	pais	ocupacion	tipo_vivienda	estad
SK_CLIENTE								
61862	2	45.0	24.0	15.0	Colombia	Policia Nacional	4.0	Ca
63954	2	87.0	24.0	21.0	Colombia	Pensionado	3.0	Ca
68937	2	80.0	26.0	26.0	Colombia	Pensionado	3.0	Ca
69976	1	71.0	27.0	23.0	Colombia	Pensionado	4.0	Se
73971	3	67.0	23.0	23.0	Colombia	Pensionado	4.0	Ca
...
4559998	5	57.0	4.0	4.0	Colombia	Pensionado	4.0	Se
4560003	3	61.0	26.0	25.0	Colombia	Empleado Privado	4.0	Ca
4560008	1	28.0	9.0	9.0	Colombia	Estudiante	3.0	Se
4560012	3	77.0	21.0	21.0	Colombia	Pensionado	4.0	Ca
4560024	2	46.0	3.0	3.0	Colombia	Empleado Privado	4.0	Se

321166 rows × 14 columns

In [83]: # Valor de Movientos por Edad

```
plt.scatter(df["valor_mvto"], df["edad"])
plt.xlabel("VALOR_MVTO")
plt.ylabel("EDAD_ANIOS")
plt.title("Clientes ")
plt.show()
```



In [84]: df.corr()

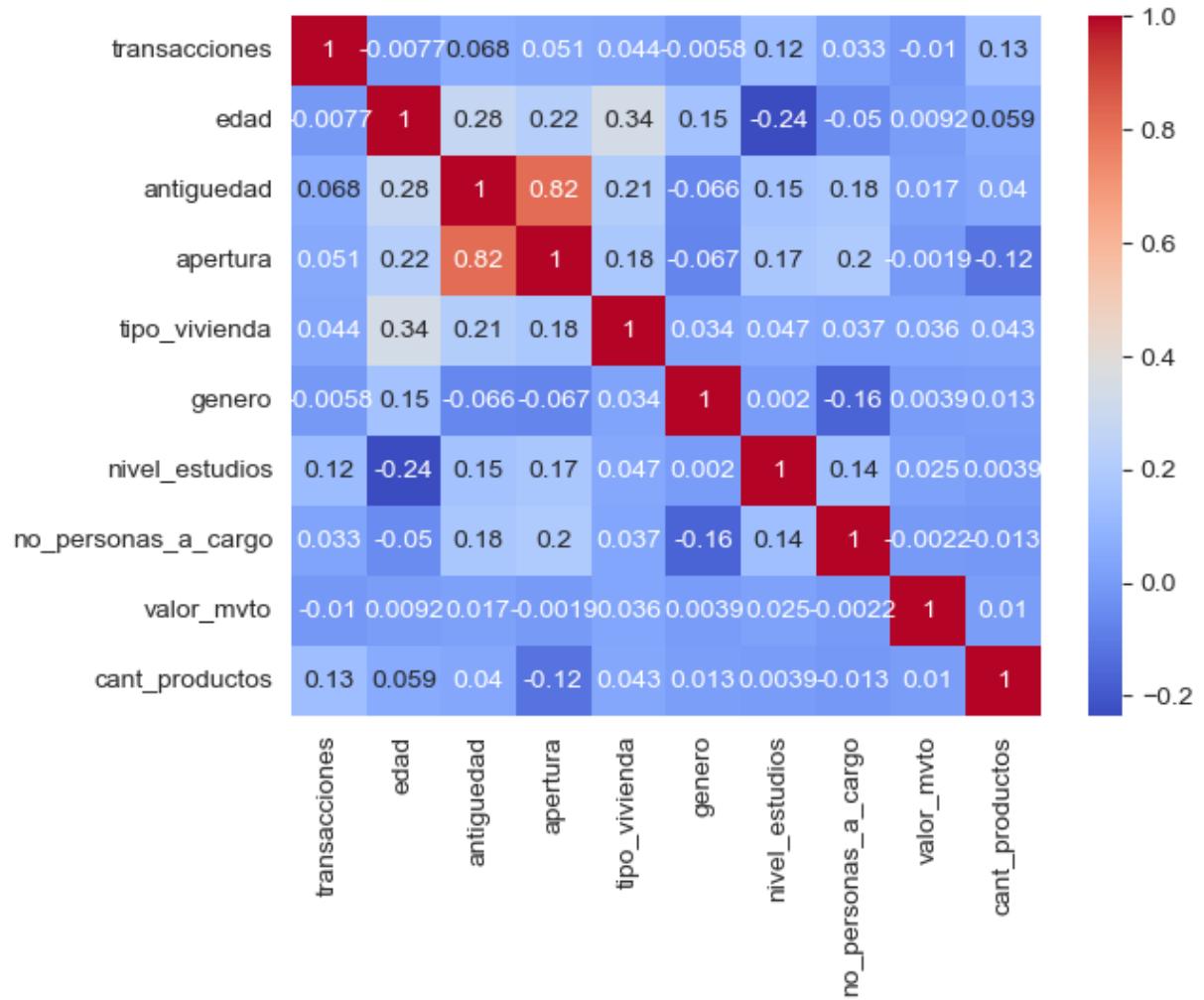
Out[84]:

	transacciones	edad	antiguedad	apertura	tipo_vivienda	genero	niv
transacciones	1.000000	-0.007743	0.068286	0.051104	0.044264	-0.005804	
edad	-0.007743	1.000000	0.278229	0.219943	0.337406	0.153061	
antiguedad	0.068286	0.278229	1.000000	0.815268	0.211688	-0.066056	
apertura	0.051104	0.219943	0.815268	1.000000	0.179848	-0.066759	
tipo_vivienda	0.044264	0.337406	0.211688	0.179848	1.000000	0.034111	
genero	-0.005804	0.153061	-0.066056	-0.066759	0.034111	1.000000	
nivel_estudios	0.123323	-0.235402	0.153449	0.171496	0.046607	0.001989	
no_personas_a_cargo	0.033266	-0.049685	0.178903	0.195084	0.036886	-0.163553	
valor_mvto	-0.010345	0.009162	0.017004	-0.001886	0.036279	0.003926	
cant_productos	0.131387	0.058848	0.040439	-0.120044	0.042658	0.012505	



In [85]: #Correlacion Heatmap

```
corr_matriz = df.corr()
sns.heatmap(corr_matriz, annot=True, cmap='coolwarm')
plt.show()
```



In [86]: *****

```
df.describe()
```

Out[86]:

	transacciones	edad	antiguedad	apertura	tipo_vivienda	genero
count	321166.000000	321166.000000	321166.000000	321166.000000	321166.000000	321166.000000
mean	3.072604	58.693831	15.138274	13.341487	3.377531	1.438225
std	3.274631	18.412249	9.328723	8.357539	0.842052	0.496176
min	1.000000	4.000000	0.000000	2.000000	0.000000	0.000000
25%	1.000000	43.000000	7.000000	6.000000	3.000000	1.000000
50%	2.000000	62.000000	15.000000	13.000000	4.000000	1.000000
75%	4.000000	73.000000	23.000000	20.000000	4.000000	2.000000
max	911.000000	172.000000	124.000000	67.000000	4.000000	2.000000

Aplicación de K-Means con dos variables

```
In [87]: data_seg = df[['antiguedad', 'transacciones']]
data_seg.reset_index(drop=True, inplace=True)
data_seg.head()
```

Out[87]:

	antiguedad	transacciones
0	24.0	2
1	24.0	2
2	26.0	2
3	27.0	1
4	23.0	3

```
In [88]: #Estan quedando registros en na para el df
data_seg.isnull().sum()
```

Out[88]:

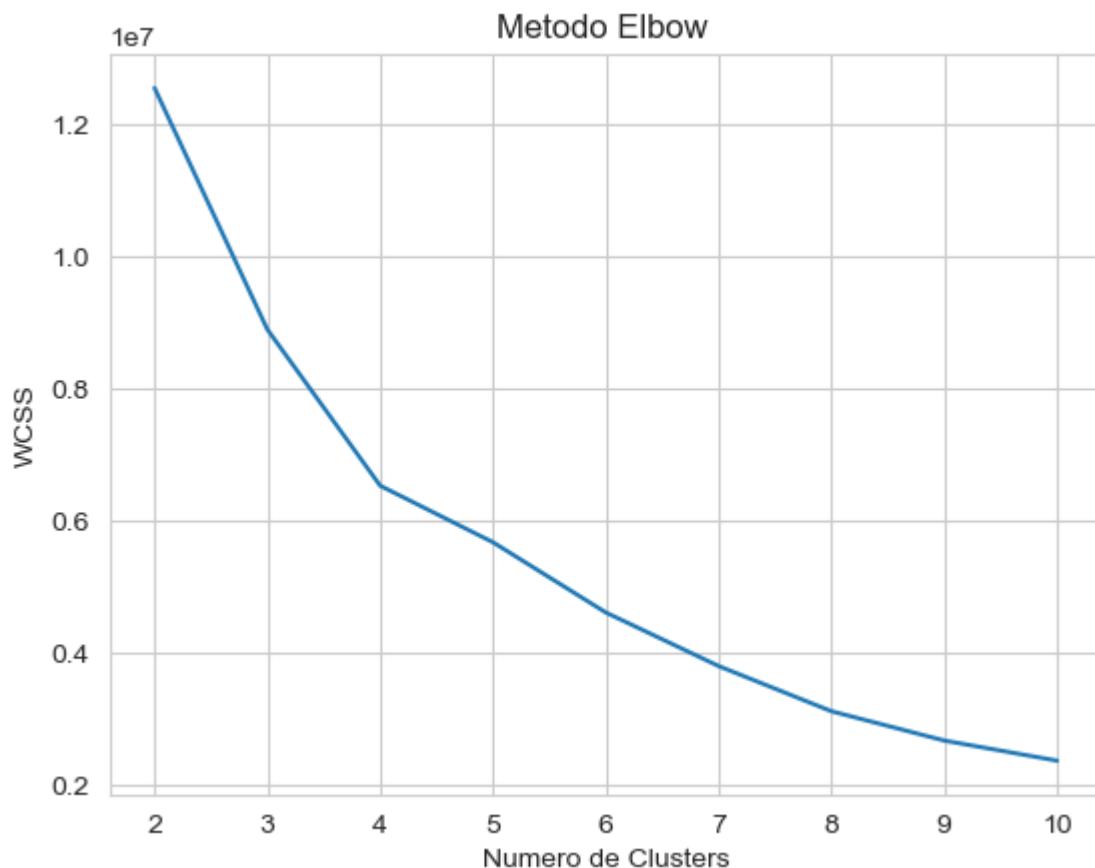
antiguedad	0
transacciones	0
dtype:	int64

```
In [89]: data_seg=data_seg.dropna()
```

Metodo Elbow

```
In [90]: # Encontrar el número de clusters
from sklearn.cluster import KMeans
wcss = []
for i in range(2, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 12)
    kmeans.fit(data_seg)
    wcss.append(kmeans.inertia_)
```

```
In [94]: import matplotlib.pyplot as plt  
plt.plot(range(2, 11), wcss)  
plt.title('Metodo Elbow ')  
plt.xlabel('Numero de Clusters')  
plt.ylabel('WCSS')  
plt.show()
```



In [95]: `data_seg`

Out[95]:

	antiguedad	transacciones
0	24.0	2
1	24.0	2
2	26.0	2
3	27.0	1
4	23.0	3
...
321161	4.0	5
321162	26.0	3
321163	9.0	1
321164	21.0	3
321165	3.0	2

321166 rows × 2 columns

In [96]:

```
import warnings
warnings.filterwarnings('ignore')
kmeans = KMeans(n_clusters = 4, init = "k-means++", random_state = 42).fit(data_seg)
data_seg['cluster'] = kmeans.labels_
data_seg.head()
```

Out[96]:

	antiguedad	transacciones	cluster
0	24.0	2	0
1	24.0	2	0
2	26.0	2	0
3	27.0	1	0
4	23.0	3	0

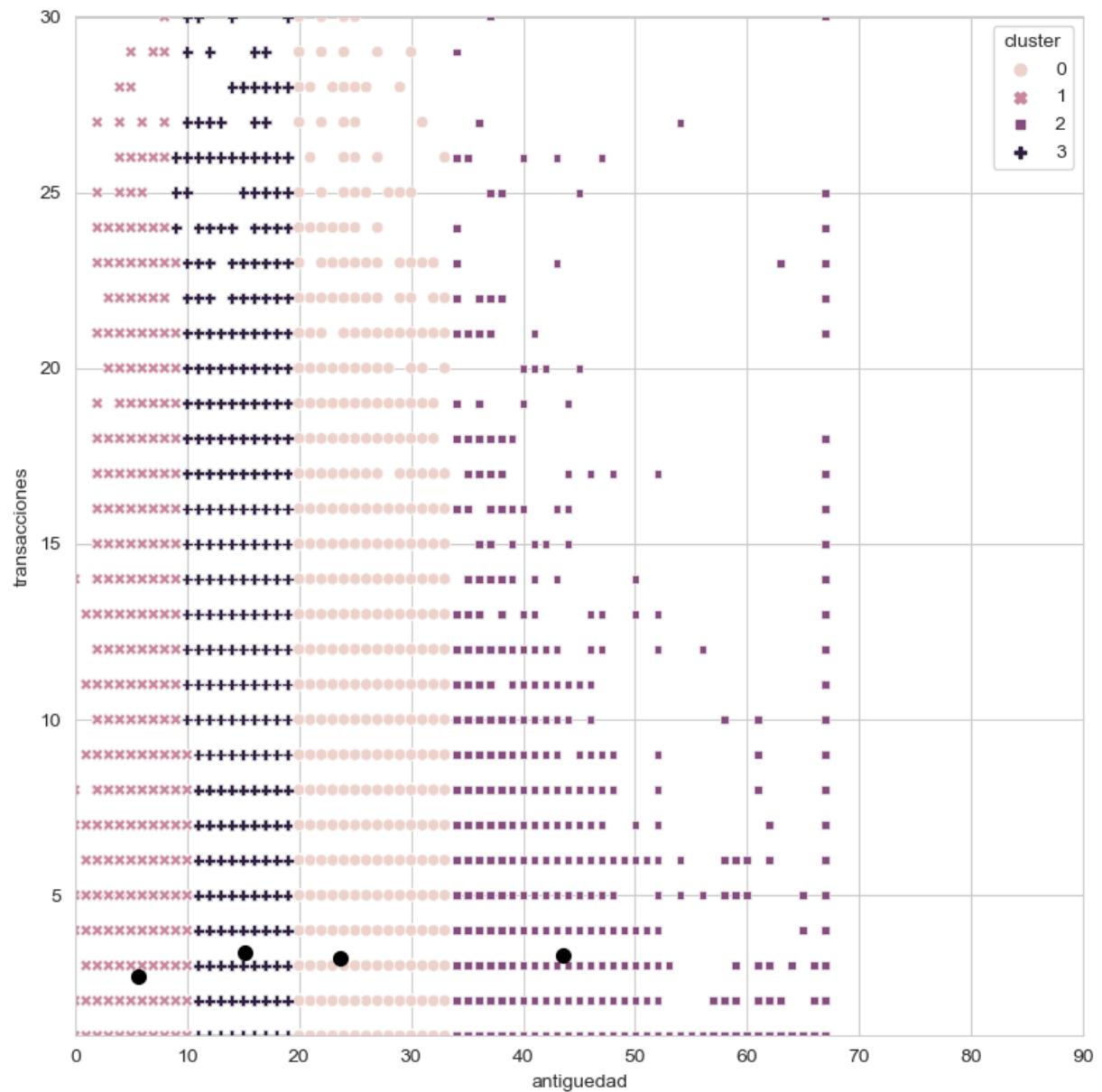
In [97]:

```
centers = pd.DataFrame(kmeans.cluster_centers_, columns=['antiguedad', 'transacciones'])
centers['freq']=data_seg['cluster'].value_counts()
centers['freq_r']=data_seg['cluster'].value_counts(normalize=True)
print(centers)
```

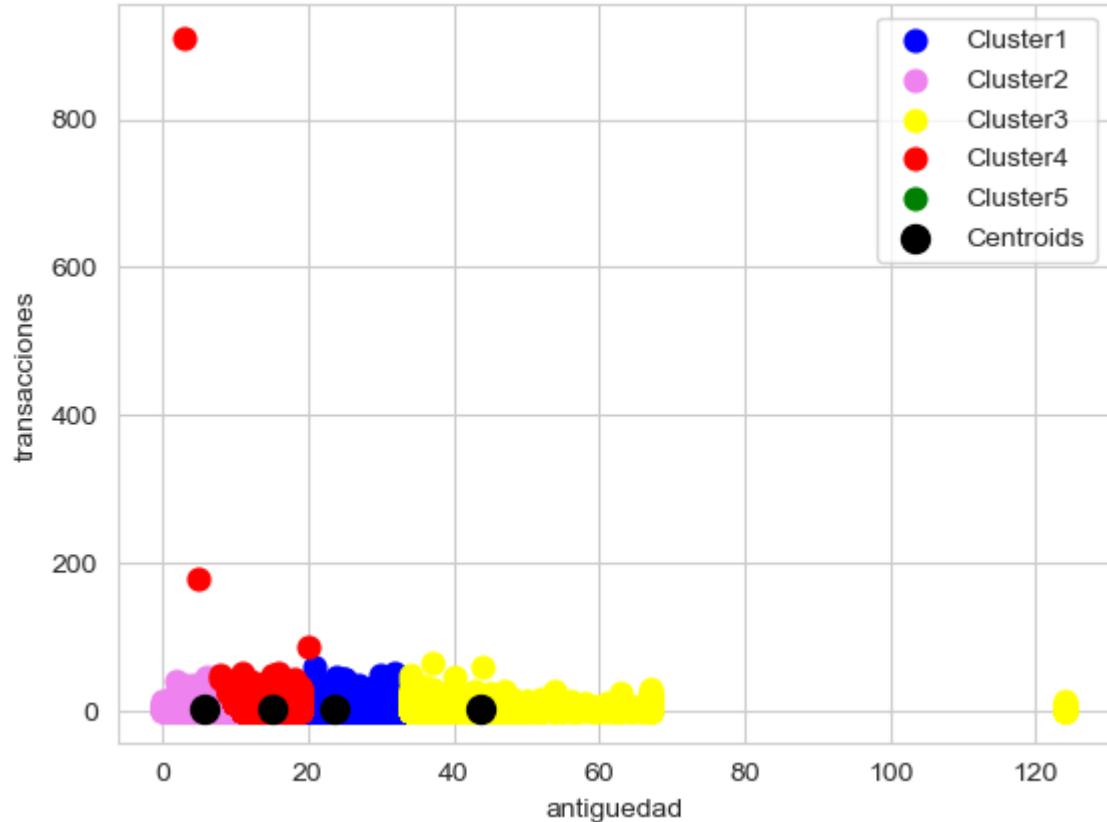
	antiguedad	transacciones	freq	freq_r
0	23.714188	3.227733	105132	0.327345
1	5.622200	2.706449	119547	0.372228
2	43.615811	3.270441	7754	0.024143
3	15.218990	3.364848	88733	0.276284

Visualización clusters con dos variables

```
In [98]: import seaborn as sns
fig, ax = plt.subplots(figsize=(8, 8))
ax = sns.scatterplot(x='antiguedad', y='transacciones', hue='cluster', style='clu
                           ax=ax, data=data_seg)
ax.set_xlim(0, 90)
ax.set_ylim(1, 30)
centers.plot.scatter(x='antiguedad', y='transacciones', ax=ax, s=50, color='black')
plt.tight_layout()
plt.show()
```



```
In [99]: import matplotlib.pyplot as plt  
plt.scatter(data_seg[data_seg['cluster']==0]['antiguedad'], data_seg[data_seg['c']  
plt.scatter(data_seg[data_seg['cluster']==1]['antiguedad'], data_seg[data_seg['c']  
plt.scatter(data_seg[data_seg['cluster']==2]['antiguedad'], data_seg[data_seg['c']  
plt.scatter(data_seg[data_seg['cluster']==3]['antiguedad'], data_seg[data_seg['c']  
plt.scatter(data_seg[data_seg['cluster']==4]['antiguedad'], data_seg[data_seg['c']  
plt.scatter(centers['antiguedad'], centers['transacciones'], s = 100, c = 'black'  
plt.xlabel('antiguedad')  
plt.ylabel('transacciones')  
plt.legend()  
plt.show()
```



K-means con todas las variables

```
In [100]: # preprocessing  
data1 = df[['transacciones', 'antiguedad', 'tipo_vivienda', 'nivel_estudios', 'gene  
'no_personas_a_cargo', 'cant_productos']]
```

```
In [101]: df.columns  
df.head()
```

```
Out[101]:
```

SK_CLIENTE	transacciones	edad	antiguedad	apertura	pais	ocupacion	tipo_vivienda	estac
61862	2	45.0	24.0	15.0	Colombia	Policia Nacional	4.0	Ca
63954	2	87.0	24.0	21.0	Colombia	Pensionado	3.0	Ca
68937	2	80.0	26.0	26.0	Colombia	Pensionado	3.0	Ca
69976	1	71.0	27.0	23.0	Colombia	Pensionado	4.0	Se
73971	3	67.0	23.0	23.0	Colombia	Pensionado	4.0	Ca

```
In [102]: data1.reset_index(drop=True, inplace=True)  
data1.head()
```

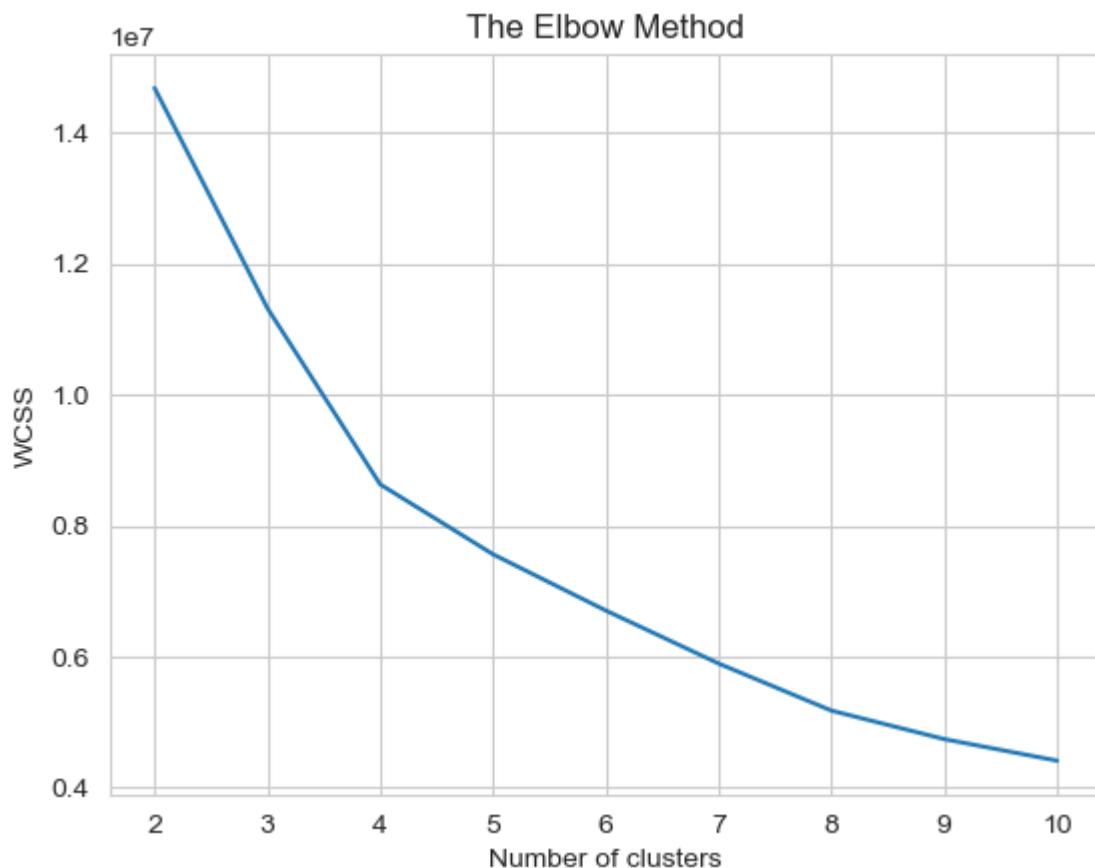
```
Out[102]:
```

	transacciones	antiguedad	tipo_vivienda	nivel_estudios	genero	no_personas_a_cargo	cant_pro
0	2	24.0	4.0	6.0	1		2.0
1	2	24.0	3.0	2.0	1		1.0
2	2	26.0	3.0	6.0	1		2.0
3	1	27.0	4.0	2.0	1		2.0
4	3	23.0	4.0	6.0	2		1.0

```
In [103]: #data1.info()  
data1=data1.dropna()
```

```
In [104]: #from sklearn.cluster import KMeans
data_km=data1.copy()
wcss = []
for i in range(2, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 12)
    kmeans.fit(data_km)
    wcss.append(kmeans.inertia_)

#import matplotlib.pyplot as plt
plt.plot(range(2, 11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



```
In [105]: #import warnings
#warnings.filterwarnings('ignore')
kmeans1 = KMeans(n_clusters = 4, init = "k-means++", random_state = 42).fit(data_km)
data_km['cluster'] = kmeans1.labels_

centers1 = pd.DataFrame(kmeans1.cluster_centers_, columns=['transacciones', 'antiguedad', 'no_personas_a_cargo', 'cant_productos'])

centers1['freq']=data_km['cluster'].value_counts()
centers1['freq_r']=data_km['cluster'].value_counts(normalize=True)
centers1
```

	transacciones	antiguedad	tipo_vivienda	nivel_estudios	genero	no_personas_a_cargo	cant_productos
0	3.382873	15.186078	3.360688	3.929146	1.392616	1.007806	
1	2.690790	5.601620	3.184667	2.948843	1.500097	0.551161	
2	3.270441	43.615811	3.648955	3.469435	1.444287	0.860975	
3	3.226617	23.714284	3.590104	3.758573	1.406491	1.183822	

```
In [106]: pip install pandoc
```

Note: you may need to restart the kernel to use updated packages.

WARNING: Retrying (Retry(total=4, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ProxyError('Cannot connect to proxy.', OSErr('Tunnel connection failed: 407 authenticationrequired'))': /simple/pandoc/
 WARNING: Retrying (Retry(total=3, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ProxyError('Cannot connect to proxy.', OSErr('Tunnel connection failed: 407 authenticationrequired'))': /simple/pandoc/
 WARNING: Retrying (Retry(total=2, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ProxyError('Cannot connect to proxy.', OSErr('Tunnel connection failed: 407 authenticationrequired'))': /simple/pandoc/
 WARNING: Retrying (Retry(total=1, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ProxyError('Cannot connect to proxy.', OSErr('Tunnel connection failed: 407 authenticationrequired'))': /simple/pandoc/
 WARNING: Retrying (Retry(total=0, connect=None, read=None, redirect=None, status=None)) after connection broken by 'ProxyError('Cannot connect to proxy.', OSErr('Tunnel connection failed: 407 authenticationrequired'))': /simple/pandoc/
 ERROR: Could not find a version that satisfies the requirement pandoc (from versions: none)
 ERROR: No matching distribution found for pandoc

