Master UNIR

K-prototypes

In [499]:
```
#!pip install umap
#!pip install umap-learn
```

In [3]:
```
#Importación de librerías necesarias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import seaborn as sns
from datetime import datetime as dt
from datetime import date
from kmodes.kprototypes import KPrototypes
from kmodes.util.dissim import matching_dissim
from kmodes.util.dissim import euclidean_dissim
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import umap.umap_ as umap
from sklearn.model_selection import cross_val_score
import logging
import shap
```

Realizamos como primer paso la importación del dataset a utilizar:

In [4]:
```
#Código para cargar el Dataset
#date=pd.read_csv("Laboratorio_dataset_car.csv", sep=";")
data=pd.read_csv("202112_1-1.csv")
dscopy=data
```

In [5]:
```python
pd.set_option('display.max_columns', 30)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 95151 entries, 0 to 95150
Data columns (total 30 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   SK_CLIENTE             95151 non-null  object 
 1   NO_PRODUCTO            77616 non-null  float64
 2   SK_TRANSACCION         77616 non-null  float64
 3   DS_NOMBRE              77616 non-null  object 
 4   SK_FE_TRANSACCION      77616 non-null  float64
 5   SK_PRODUCTO_SERVICIO   77616 non-null  float64
 6   DS_LINEA_PRODUCTO      77616 non-null  object 
 7   DS_FAMILIA             77616 non-null  object 
 8   DS_CLASE               77616 non-null  object 
 9   VALOR_MVTO             77616 non-null  float64
 10  CANTIDAD_TRANSACCION   77616 non-null  float64
 11  DS_SECTOR_PIB          77616 non-null  object 
 12  DS_TIPO_EMPRESA        77616 non-null  object 
 13  DS_TIPO_PERSONA        77616 non-null  object 
 14  DS_GENERO              77616 non-null  object 
 15  FE_VINCULACION_CLIENTE 77606 non-null  object 
 16  DS_ESTADO_CIVIL        77616 non-null  object 
 17  FE_NACIMIENTO          70247 non-null  object 
 18  DS_PAIS_NACIMIENTO     77616 non-null  object 
 19  DS_NIVEL_ESTUDIOS      77616 non-null  object 
 20  DS_PROFESION           77616 non-null  object 
 21  DS_OCUPACION           77616 non-null  object 
 22  DS_TIPO_VIVIENDA       77616 non-null  object 
 23  DS_RAZON_SOCIAL        77616 non-null  object 
 24  ID_CLIENTE             77616 non-null  float64
 25  NO_PERSONAS_A_CARGO    77616 non-null  float64
 26  REF_NUM                58685 non-null  float64
 27  FE_APERTURA            77616 non-null  object 
 28  DK_PERSONA             77616 non-null  object 
 29  SK_RC                  77616 non-null  float64
dtypes: float64(10), object(20)
memory usage: 21.8+ MB
```

In [503]:
```
data.head(5)
```

Out[503]:

	SK_CLIENTE	NO_PRODUCTO	SK_TRANSACCION	DS_NOMBRE	SK_FE_TRANSACCION	SK
0	2582480	1.100141e+11	714.0	IVA SOBRE CHEQUERAS (IGUAL SER	20211206.0	
1	2582480	1.100141e+11	758.0	COBRO DE CHEQUERA (IGUAL SERIE	20211206.0	
2	2582480	1.100141e+11	644.0	ND.TIMBRES CHEQUERA	20211206.0	
3	2582480	1.100141e+11	2562.0	CONSIGNACION AVAL	20211206.0	
4	2582480	1.100141e+11	345.0	ABONOS POR A.C.H	20211206.0	

Procedemos a limpiar las columnas que no contienen información de relevancia para la construcción del modelo y las filas con datos nulos.

In [6]:
```
drop_list=["DS_FAMILIA","REF_NUM","ID_CLIENTE","DK_PERSONA","DS_NOMBRE","DS_RAZ
           "DS_TIPO_EMPRESA","SK_RC","DK_PERSONA","DS_PAIS_NACIMIENTO","FE_APER
data = data.drop(drop_list, axis=1)
data=data.dropna()
data=data.drop_duplicates()
```

Además las columnas de fecha son cambiadas por años, con el fin de mejorar la interpretación de éstas en la construcción del modelo.

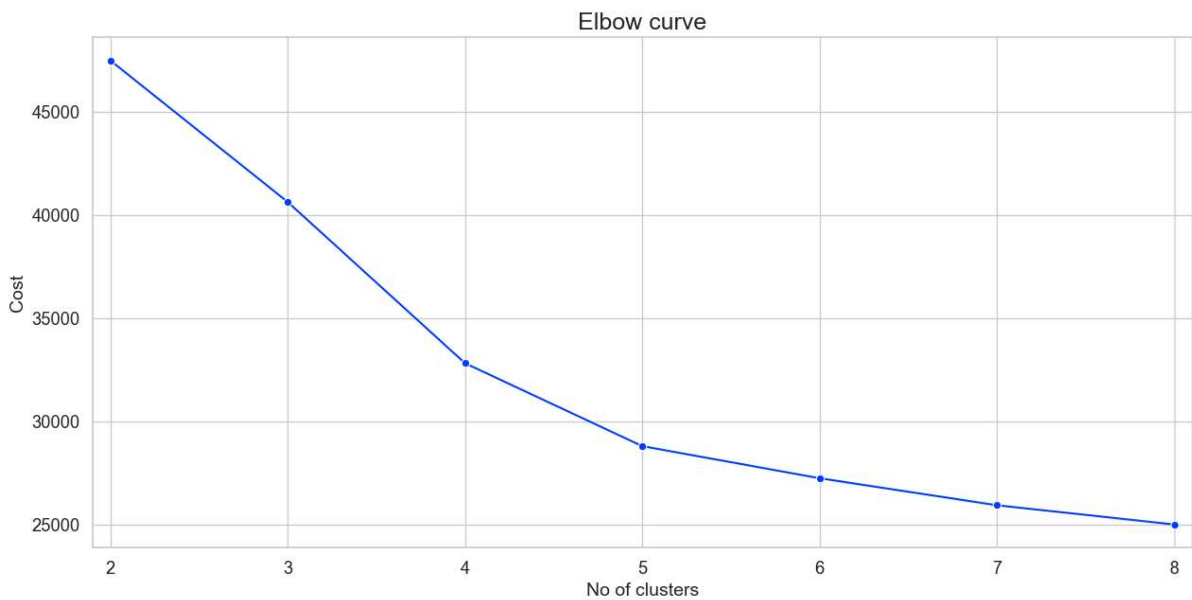
In [7]:
```
data['EDAD'] = (dt.now()-pd.to_datetime(data['FE_NACIMIENTO'],errors='coerce'))
data['ANTIGUEDAD_CLIENTE'] = (dt.now()-pd.to_datetime(data['FE_VINCULACION_CLIE
data=data.dropna()
```

Se reordenan las columnas para separarlas en categoricas y numéricas:

```python
In [8]: data = data[['SK_CLIENTE',
               'SK_PRODUCTO_SERVICIO', 'DS_LINEA_PRODUCTO', 'DS_CLASE',
               'CANTIDAD_TRANSACCION','DS_GENERO',
               'DS_ESTADO_CIVIL',
               'DS_NIVEL_ESTUDIOS', 'DS_PROFESION',
               'DS_OCUPACION', 'DS_TIPO_VIVIENDA', 'NO_PERSONAS_A_CARGO','EDAD','ANTIGU
               'VALOR_MVTO']]
```

Teniendo el dataset limpio, se procede a realizar la curva de codo, con el fin de estimar el valor óptimo para el número de clusters.

In [9]:
```
categorical_index = list(range(0,10))
df =data.drop(["SK_CLIENTE"], axis=1).sample(10000, random_state=40)
scaled_X = StandardScaler().fit_transform(df[['NO_PERSONAS_A_CARGO','EDAD','ANT
df[['NO_PERSONAS_A_CARGO','EDAD','ANTIGUEDAD_CLIENTE','VALOR_MVTO']] = scaled_X
# Function for plotting elbow curve
def plot_elbow_curve(start, end, data):
    no_of_clusters = list(range(start, end+1))
    cost_values = []

    for k in no_of_clusters:
        test_model = KPrototypes(n_clusters=k, init='Huang', random_state=42)
        test_model.fit_predict(data, categorical=categorical_index)
        cost_values.append(test_model.cost_)

    sns.set_theme(style="whitegrid", palette="bright", font_scale=1.2)

    plt.figure(figsize=(15, 7))
    ax = sns.lineplot(x=no_of_clusters, y=cost_values, marker="o", dashes=False
    ax.set_title('Elbow curve', fontsize=18)
    ax.set_xlabel('No of clusters', fontsize=14)
    ax.set_ylabel('Cost', fontsize=14)
    ax.set(xlim=(start-0.1, end+0.1))
    plt.plot();

# Plotting elbow curve for k=2 to k=7
plot_elbow_curve(2,8,df)
```



Se utilizará k=4 para la realización del cluster, debido a que es un valor apropiado observando la gráfica. Por tanto, se procede a la construcción del modelo.

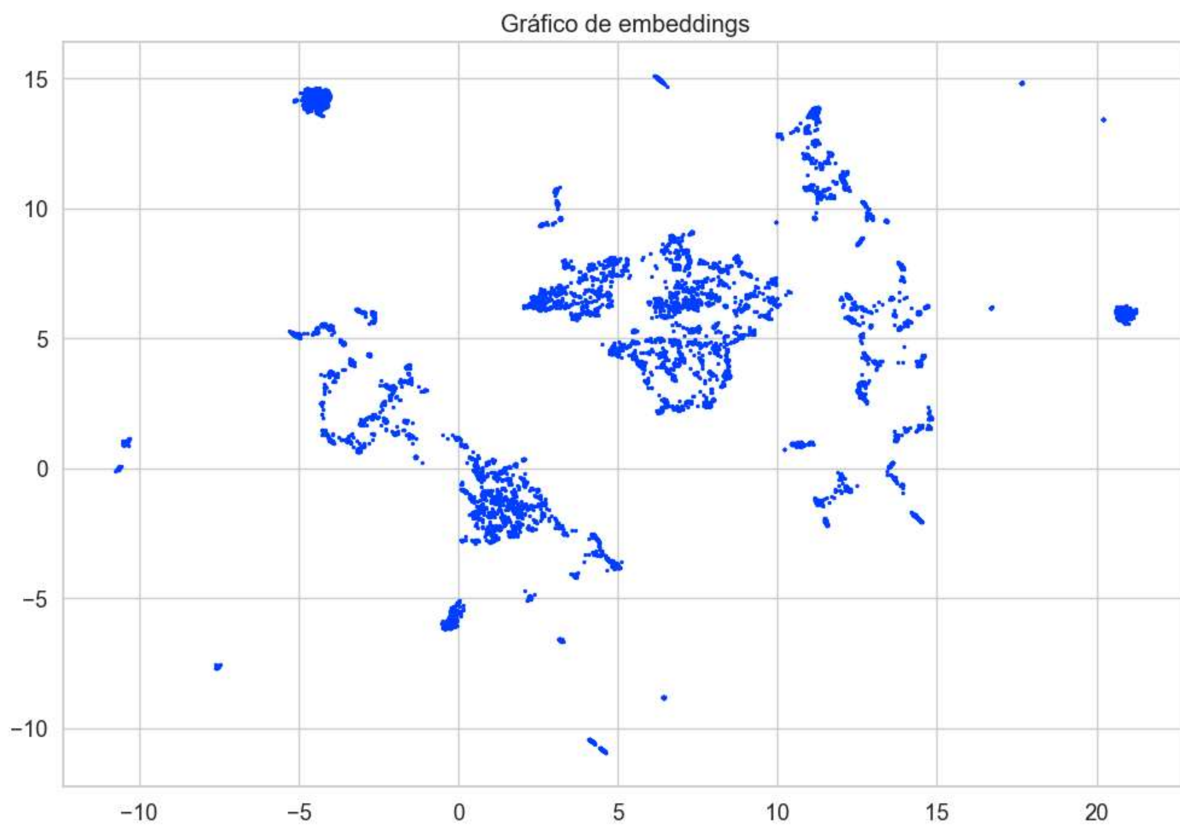
In [10]:
```
model_4 = KPrototypes(n_clusters=4, init='Huang', random_state=42, n_jobs=-1)
model_4.fit_predict(df, categorical=categorical_index)
labels = model_4.labels_
print(model_4.cost_)
```

```
32825.607173671764
```

Con el fin de gráficar nuestro dataset, se debe realizar algún método de reducción de dimensionalidad. Se trabajará por medio de reducción con embeddings, utilizando el método UMAP, pues con éste se tiene la particularidad de poder utilizar datos mixtos, de tipo categórico y numérico.

In [13]:
```python
##preprocessing categorical
numerical = df[['NO_PERSONAS_A_CARGO','EDAD','ANTIGUEDAD_CLIENTE','VALOR_MVTO']
categorical = df[['SK_PRODUCTO_SERVICIO', 'DS_LINEA_PRODUCTO', 'DS_CLASE', 'CAN
                'DS_ESTADO_CIVIL','DS_NIVEL_ESTUDIOS', 'DS_PROFESION','DS_OCUPA
categorical_dummies = pd.get_dummies(categorical)

#Percentage of columns which are categorical is used as weight parameter in emb
categorical_weight = len(categorical) / df.shape[1]

#Embedding numerical & categorical
fit1 = umap.UMAP(metric='l2').fit(numerical)
fit2 = umap.UMAP(metric='dice').fit(categorical_dummies)

#Augmenting the numerical embedding with categorical
intersection = umap.general_simplicial_set_intersection(fit1.graph_, fit2.graph
intersection = umap.reset_local_connectivity(intersection)
embedding = umap.simplicial_set_embedding(fit1._raw_data, intersection, fit1.n_
                                         fit1._initial_alpha, fit1._a, f
                                         fit1.repulsion_strength, fit1.r
                                         200, 'random', np.random, fit1.
                                         fit1._metric_kwds, False, densm

plt.figure(figsize=(12, 8))
x,y = zip(*embedding[0])
plt.scatter(x,y, s=2, cmap='Spectral', alpha=1.0)
plt.title('Gráfico de embeddings')
plt.show()
```

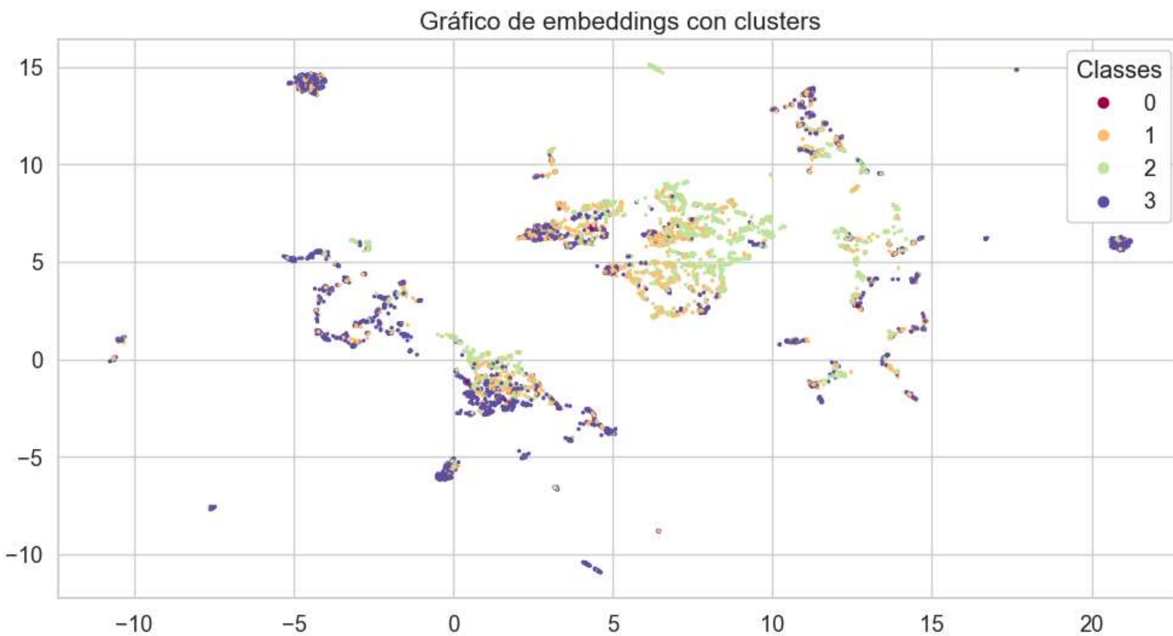
```
gradient function is not yet implemented for dice distance metric; inverse_tr
ansform will be unavailable
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignored
```



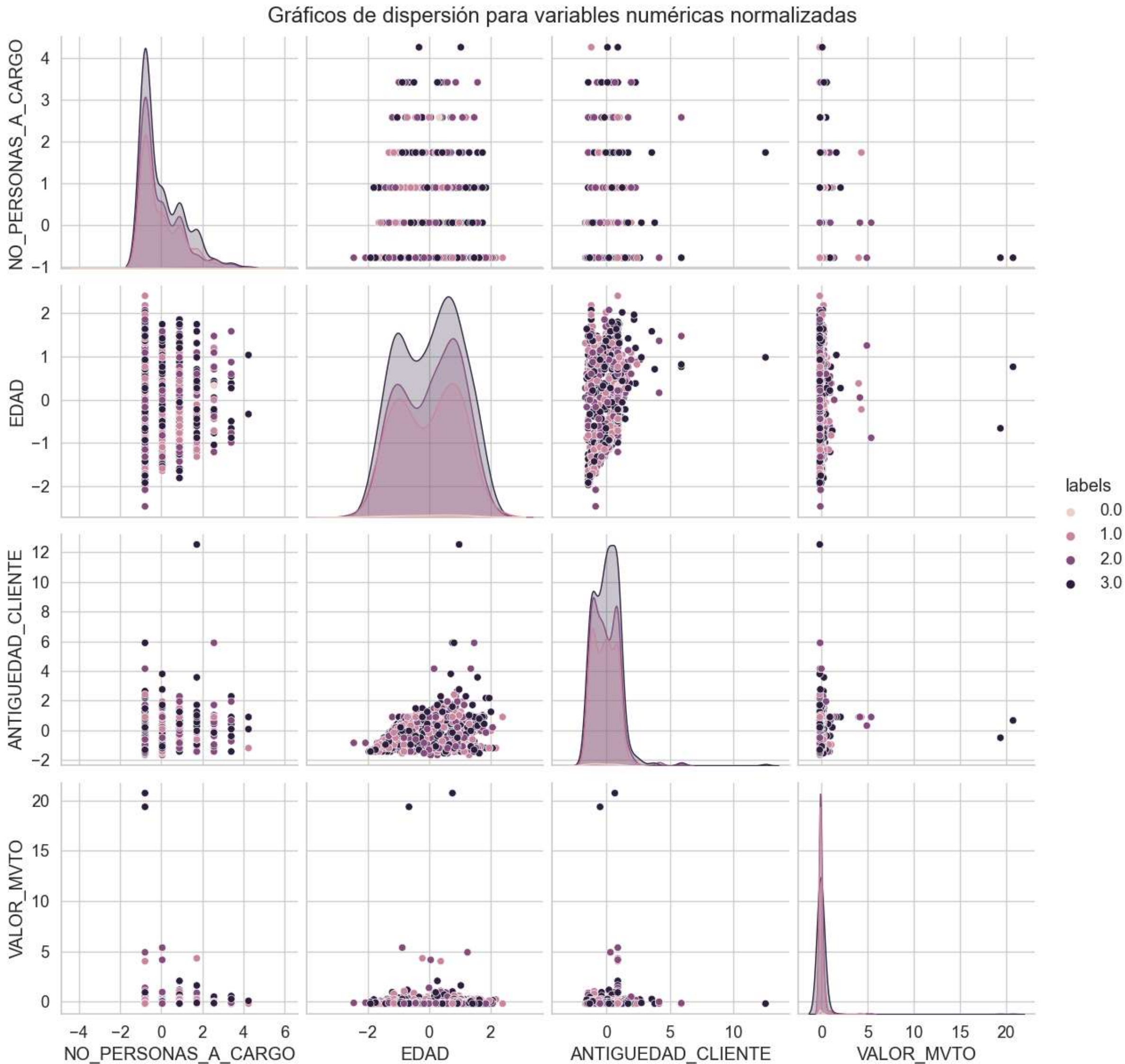
Ahora que ya se tiene el gráfico del embedding, es importante revisar cómo se aprecia la separación de clusters en éste.

In [14]:
```
fig, ax = plt.subplots(figsize=(12, 6))
x,y = zip(*embedding[0])
scatter = ax.scatter(x,y,s=2, cmap='Spectral', alpha=1.0, c=labels)
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="upper right", title="Classes")
ax.add_artist(legend1)
ax.set_title('Gráfico de embeddings con clusters',fontsize=15)
```

Out[14]:
```
Text(0.5, 1.0, 'Gráfico de embeddings con clusters')
```



In [15]:
```
df_pairplot = numerical.copy()
df_pairplot.loc[:,"labels"]=pd.Series(labels)
ax = sns.pairplot(df_pairplot, hue='labels',height=3, aspect=1)
ax.fig.suptitle('Gráficos de dispersión para variables numéricas normalizadas',
sns.set_context("paper", rc={"axes.labelsize":10})
```



Ahora que tenemos los clusters definidos, realicemos una validación de estos por medio del coeficiente de silueta y tener una medida de qué tan apropiado resultó el agrupamiento.

In [19]:
```
df_model4 = pd.concat([categorical_dummies, numerical], axis=1, ignore_index= T
from sklearn.metrics import silhouette_score
silhouette_score(df_model4, labels)
```

Out[19]: 0.15703955461258348

In [ ]:
```

```

In [20]:
```
from scipy.stats import chi2_contingency

def cramers_V(var1, var2):
    crosstab = np.array(pd.crosstab(var1, var2))
    stats = chi2_contingency(crosstab)[0]
    cram_V = stats / (np.sum(crosstab) * (min(crosstab.shape) - 1))
    return cram_V

def cramers_col(column_name):
    col = pd.Series(np.empty(df.columns.shape), index=df.columns, name=column_r
    for row in df:
        cram = cramers_V(df[column_name], df[row])
        col[row] = round(cram, 2)
    return col

df.apply(lambda column: cramers_col(column.name))
```

Out[20]:

	SK_PRODUCTO_SERVICIO	DS_LINEA_PRODUCTO	DS_CLASE	CAN
SK_PRODUCTO_SERVICIO	1.00	1.00	1.00	
DS_LINEA_PRODUCTO	1.00	1.00	1.00	
DS_CLASE	1.00	1.00	0.99	
CANTIDAD_TRANSACCION	0.00	0.00	0.00	
DS_GENERO	0.00	0.00	0.00	
DS_ESTADO_CIVIL	0.00	0.00	0.00	
DS_NIVEL_ESTUDIOS	0.00	0.00	0.00	
DS_PROFESION	0.02	0.02	0.06	
DS_OCUPACION	0.03	0.03	0.11	
DS_TIPO_VIVIENDA	0.00	0.00	0.00	
NO_PERSONAS_A_CARGO	0.00	0.00	0.00	

Otra forma de evaluar internamente qué tan eficiente es el cluster, es utilizando el coeficiente F1 de validación cruzada, para esto nos apoyaremos en un clasificador LGBM, que nos permite trabajar variables de tipo mixto.

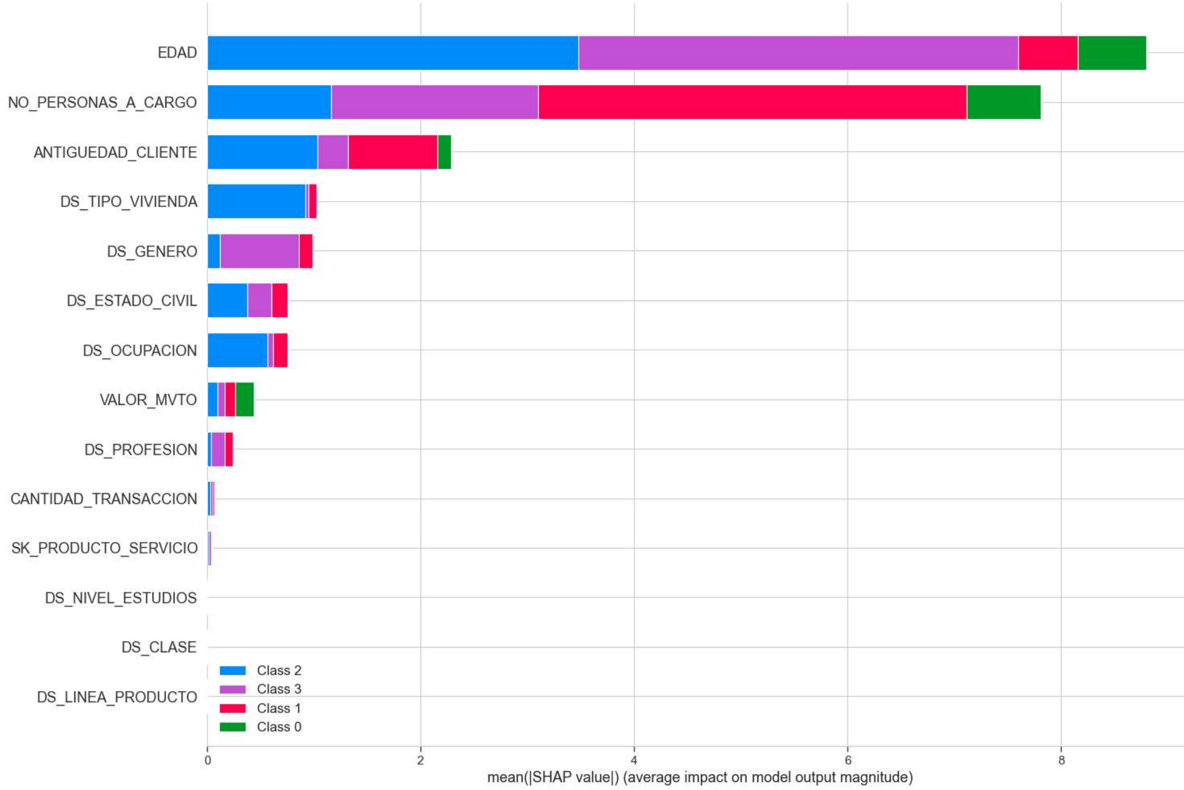
In [22]:
```python
from lightgbm import LGBMClassifier
for c in df.select_dtypes(include='object'):
    df[c] = df[c].astype('category')
clf_kp = LGBMClassifier(colsample_by_tree=0.8)
cv_scores_kp = cross_val_score(clf_kp, df, labels, scoring='f1_weighted')
print(f'CV F1 score for K-Prototypes clusters is {np.mean(cv_scores_kp)}')
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Dado que el valor de validación cruzada es 0.985, se puede establecer que el dataset fue clasificado en grupos con una separación distinguible y representativa. Podemos ahora revisar la significancia de cada una de las variables usadas en el clustering, para lo cual primero entrenaremos un clasificador y luego apoyandonos en los valores SHAP, se revisará la importancia de cada variable en el cluster.

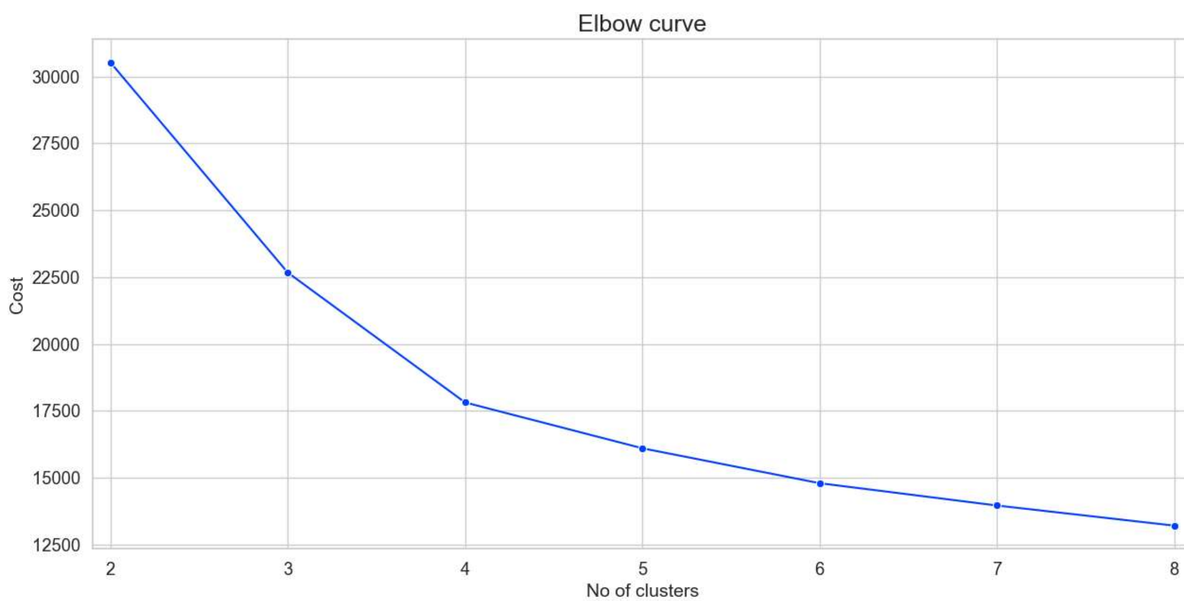
In [23]:
```
clf_kp.fit(df, labels)
explainer_kp = shap.TreeExplainer(clf_kp)
shap_values_kp = explainer_kp.shap_values(df)
shap.summary_plot(shap_values_kp, df, plot_type="bar", plot_size=(15, 10))
```

```
[LightGBM] [Warning] Unknown parameter: colsample_by_tree
[LightGBM] [Warning] Unknown parameter: colsample_by_tree
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.000410 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 591
[LightGBM] [Info] Number of data points in the train set: 10000, number of us
ed features: 14
[LightGBM] [Info] Start training from score -5.776353
[LightGBM] [Info] Start training from score -1.376344
[LightGBM] [Info] Start training from score -1.220780
[LightGBM] [Info] Start training from score -0.799842
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```



Basándonos en la gráfica realizada, repetiremos el cluster usando solamente aquellas variables que demuestran significancia.

```
In [25]: categorical_index = list(range(0,5))
drop_list2= ["CANTIDAD_TRANSACCION","DS_PROFESION","SK_PRODUCTO_SERVICIO","DS_C
df2 =df.drop(drop_list2, axis=1)

# Function for plotting elbow curve
def plot_elbow_curve(start, end, data):
    no_of_clusters = list(range(start, end+1))
    cost_values = []

    for k in no_of_clusters:
        test_model = KPrototypes(n_clusters=k, init='Huang', random_state=42)
        test_model.fit_predict(data, categorical=categorical_index)
        cost_values.append(test_model.cost_)

    sns.set_theme(style="whitegrid", palette="bright", font_scale=1.2)

    plt.figure(figsize=(15, 7))
    ax = sns.lineplot(x=no_of_clusters, y=cost_values, marker="o", dashes=False
    ax.set_title('Elbow curve', fontsize=18)
    ax.set_xlabel('No of clusters', fontsize=14)
    ax.set_ylabel('Cost', fontsize=14)
    ax.set(xlim=(start-0.1, end+0.1))
    plt.plot();

# Plotting elbow curve for k=2 to k=7
plot_elbow_curve(2,8,df2)
```

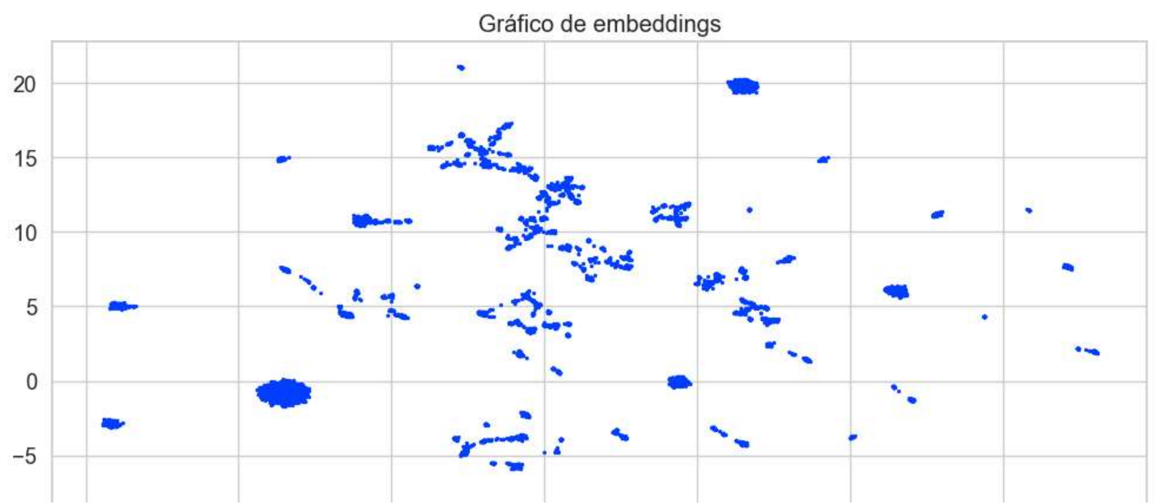



```
In [26]: model_val = KPrototypes(n_clusters=4, init='Huang', random_state=42, n_jobs=-1)
model_val.fit_predict(df2, categorical=categorical_index)
labels_2 = model_val.labels_
print(model_val.cost_)
```

```
17813.23543781217
```

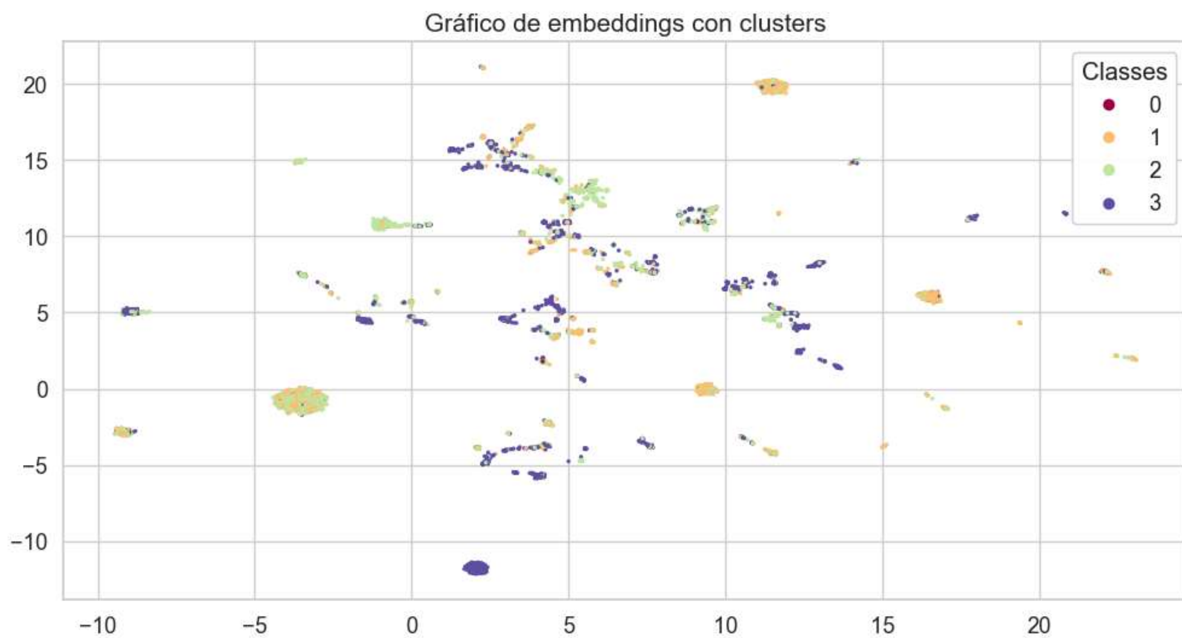
In [29]:
```
##preprocessing categorical
numerical_2 = df2[['EDAD','ANTIGUEDAD_CLIENTE','VALOR_MVTO']]
categorical_2 = df2[['DS_LINEA_PRODUCTO','DS_GENERO',
                'DS_ESTADO_CIVIL','DS_OCUPACION', 'DS_TIPO_VIVIENDA']]
categorical_dummies_2 = pd.get_dummies(categorical_2)

#Percentage of columns which are categorical is used as weight parameter in emb
categorical_weight = len(categorical_2) / df2.shape[1]

#Embedding numerical & categorical
fit1 = umap.UMAP(metric='l2').fit(numerical_2)
fit2 = umap.UMAP(metric='dice').fit(categorical_dummies_2)

#Augmenting the numerical embedding with categorical
intersection = umap.general_simplicial_set_intersection(fit1.graph_, fit2.graph
intersection = umap.reset_local_connectivity(intersection)
embedding = umap.simplicial_set_embedding(fit1._raw_data, intersection, fit1.n_
                                                fit1._initial_alpha, fit1._a, f
                                                fit1.repulsion_strength, fit1.r
                                                200, 'random', np.random, fit1.
                                                fit1._metric_kwds, False, densm

plt.figure(figsize=(12, 6))
x,y = zip(*embedding[0])
plt.scatter(x,y, s=2, cmap='Spectral', alpha=1.0)
plt.title('Gráfico de embeddings')
plt.show()
```

```
gradient function is not yet implemented for dice distance metric; inverse_
transform will be unavailable
No data for colormapping provided via 'c'. Parameters 'cmap' will be ignore
d
```



In [30]:
```
fig, ax = plt.subplots(figsize=(12, 6))
x,y = zip(*embedding[0])
scatter = ax.scatter(x,y,s=2, cmap='Spectral', alpha=1.0, c=labels_2)
legend1 = ax.legend(*scatter.legend_elements(),
                    loc="upper right", title="Classes")
ax.add_artist(legend1)
ax.set_title('Gráfico de embeddings con clusters',fontsize=15)
```

Out[30]: Text(0.5, 1.0, 'Gráfico de embeddings con clusters')



In [31]:
```
df_model_val = pd.concat([categorical_dummies_2, numerical_2], axis=1, ignore_i
from sklearn.metrics import silhouette_score
silhouette_score(df_model_val, labels_2)
```

Out[31]: 0.1980837034323855

Como se puede apreciar el valor del coeficiente de silueta tiene una mejora considerable, demostrando una optimización del cluster.

In [33]:

```python
from lightgbm import LGBMClassifier
for c in df2.select_dtypes(include='object'):
    df2[c] = df[c].astype('category')
clf_kp = LGBMClassifier(colsample_by_tree=0.8)
cv_scores_kp = cross_val_score(clf_kp, df2, labels_2, scoring='f1_weighted')
print(f'CV F1 score for K-Prototypes clusters is {np.mean(cv_scores_kp)}')
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

El coeficiente de validación cruzada se mantiene en un valor parecido, por lo cual internamente con respecto al dataset el cluster hace una segmentación adecuada.