Teresa de Silva Echeguren

# Trabajo final-Modelos no supervisados


Una compañía e-comerce que vende productos alimenticios lanza regularmente ofertas y campañas a sus clientes. Con el objetivo de optimizar el ratio de éxito cuando crea una nueva campaña o cuando llega un nuevo cliente (y no tiene todavía suficiente información para modelar la afinidad entre una campaña y sus clientes) quiere realizar una segmentación de sus clientes y campañas para abordar este problema con mayor facilidad.

En los modelos no supervisados, un algoritmo divide los datos en un conjunto para encontrar sus características ocultas, y asi encontrar su estructura oculta para poder detectar errores. Esto nos ayuda a incrementar el conocimiento estructural de los datos que tenemos asi como de datos futuros. Lo podemos hacer recopilando los datos mas parecidos (creando clusters) o reduciendo la dimensionalidad (utilizando PCA).

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import seaborn as sns
import plotly.graph_objects as go
warnings.filterwarnings('ignore')

from time import sleep
from random import shuffle
from numpy import atleast_2d

from sklearn.decomposition import PCA
from sklearn.cluster import DBSCAN
from sklearn.metrics import adjusted_mutual_info_score
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import load_iris
from sklearn.neighbors import KDTree
from sklearn.datasets import make_moons
from sklearn.metrics import adjusted_rand_score
from sklearn.model_selection import train_test_split
from sklearn.manifold import TSNE

from matplotlib.colors import ListedColormap
from matplotlib import cm
import matplotlib.ticker as ticker

from IPython import display
from matplotlib.legend_handler import HandlerLineCollection, HandlerTuple
```

Para esta tarea vamos a utilizar tres bases de datos diferentes.

- La primera, customers, regrupa los datos del cliente. Entre estos datos podemos encontar sus diferentes caracteristicas como año de nacimiento, nivel educativo o estado marital.
- La segunda, campaigns, regrupa los datos de diferentes campañas, que son 100.
- Por ultimo tenemos campaigns to customers. Donde nos encontramos con el numero de clientes la campaña que se le atribuye y por ultimo la columna de results que es para ver si la campaña funciona (1 si funciona, 0 si no funciona).

In [2]:
```
customers = pd.read_csv('customers.csv')
campaigns = pd.read_csv('campaigns.csv')
campaigns_to_customers = pd.read_csv('campaigns_to_customers_launched.csv')
```

Lo primero que hacemos es analizar las bases de datos para ver si hay NA´s.

In [3]:
```
def missing_summary(dataframe):
    num_elementos = dataframe.count()
    num_missing = dataframe.isna().sum()
    missing_summary = pd.DataFrame(index=num_elementos.index,
                                   data={'total':num_elementos,
                                         'missing':num_missing,
                                         'missing_rate (%)': round(num_missing/
    return missing_summary
```

In [4]:
```
missing_summary(campaigns_to_customers)
```

Out[4]:

| | total | missing | missing_rate (%) |
|---|---|---|---|
| **Unnamed: 0** | 13440 | 0 | 0.0 |
| **customer** | 13440 | 0 | 0.0 |
| **campaign** | 13440 | 0 | 0.0 |
| **result** | 13440 | 0 | 0.0 |

In [5]:
```
missing_summary(customers)
```

Out[5]:

| | total | missing | missing_rate (%) |
|---|---|---|---|
| **customer_id** | 2240 | 0 | 0.00 |
| **Year_Birth** | 2240 | 0 | 0.00 |
| **Education** | 2240 | 0 | 0.00 |
| **Marital_Status** | 2240 | 0 | 0.00 |
| **Income** | 2216 | 24 | 1.08 |
| **Kidhome** | 2240 | 0 | 0.00 |
| **Teenhome** | 2240 | 0 | 0.00 |
| **Dt_Customer** | 2240 | 0 | 0.00 |
| **Recency** | 2240 | 0 | 0.00 |
| **MntWines** | 2240 | 0 | 0.00 |
| **MntFruits** | 2240 | 0 | 0.00 |
| **MntMeatProducts** | 2240 | 0 | 0.00 |
| **MntFishProducts** | 2240 | 0 | 0.00 |
| **MntSweetProducts** | 2240 | 0 | 0.00 |
| **MntGoldProds** | 2240 | 0 | 0.00 |
| **NumDealsPurchases** | 2240 | 0 | 0.00 |
| **NumWebPurchases** | 2240 | 0 | 0.00 |
| **NumCatalogPurchases** | 2240 | 0 | 0.00 |
| **NumStorePurchases** | 2240 | 0 | 0.00 |
| **NumWebVisitsMonth** | 2240 | 0 | 0.00 |
| **Complain** | 2240 | 0 | 0.00 |
| **Z_CostContact** | 2240 | 0 | 0.00 |
| **Z_Revenue** | 2240 | 0 | 0.00 |
| **Response** | 2240 | 0 | 0.00 |

In [6]:

```
missing_summary(campaigns)
```

Out[6]:

| | total | missing | missing_rate (%) |
|---|---|---|---|
| **campaign_id** | 100 | 0 | 0.0 |
| **prob_food** | 100 | 0 | 0.0 |
| **prob_drink** | 100 | 0 | 0.0 |
| **outdate** | 100 | 0 | 0.0 |
| **price** | 100 | 0 | 0.0 |
| **discount** | 100 | 0 | 0.0 |
| **name** | 100 | 0 | 0.0 |

Como vemos, en el dataset de customers faltan 24 datos de la variable income. Estos NA´s los vamos a sustituir por la media de income, para no perder a esos clientes.

In [7]:
```
mean_value=customers['Income'].mean()
```

In [8]:
```
customers['Income']=customers['Income'].fillna(mean_value)
```

In [9]:
```
missing_summary(customers)
```

Out[9]:

| | total | missing | missing_rate (%) |
|---|---|---|---|
| **customer_id** | 2240 | 0 | 0.0 |
| **Year_Birth** | 2240 | 0 | 0.0 |
| **Education** | 2240 | 0 | 0.0 |
| **Marital_Status** | 2240 | 0 | 0.0 |
| **Income** | 2240 | 0 | 0.0 |
| **Kidhome** | 2240 | 0 | 0.0 |
| **Teenhome** | 2240 | 0 | 0.0 |
| **Dt_Customer** | 2240 | 0 | 0.0 |
| **Recency** | 2240 | 0 | 0.0 |
| **MntWines** | 2240 | 0 | 0.0 |
| **MntFruits** | 2240 | 0 | 0.0 |
| **MntMeatProducts** | 2240 | 0 | 0.0 |
| **MntFishProducts** | 2240 | 0 | 0.0 |
| **MntSweetProducts** | 2240 | 0 | 0.0 |
| **MntGoldProds** | 2240 | 0 | 0.0 |
| **NumDealsPurchases** | 2240 | 0 | 0.0 |
| **NumWebPurchases** | 2240 | 0 | 0.0 |
| **NumCatalogPurchases** | 2240 | 0 | 0.0 |
| **NumStorePurchases** | 2240 | 0 | 0.0 |
| **NumWebVisitsMonth** | 2240 | 0 | 0.0 |
| **Complain** | 2240 | 0 | 0.0 |
| **Z_CostContact** | 2240 | 0 | 0.0 |
| **Z_Revenue** | 2240 | 0 | 0.0 |
| **Response** | 2240 | 0 | 0.0 |

## 1.- Para hacer una posterior evaluación de los métodos desarrollados, divide el dataset campaigns_to_customers_launched.csv en training, validation y test.

Hemos dividido el dataset en train, validate y test.

- Train: Conjunto de entrenamiento.
- Validate: La técnica de Validación Cruzada la utilizaremos para calcular el comportamiento de los modelos que hemos creado, para asi poder encontrar mas rapidamente el mejor modelo.
- Test: Conjunto de prueba.

Entrenaremos el modelo con train y comprobaremos si es eficaz con test, para comprobar si funciona test calcularemos el MSE mas adelante.

In [10]:
```
train_size = 0.6
validate_size = 0.2
train, validate, test = np.split(campaigns_to_customers.sample(frac=1), [ir
```

2.- Realiza las acciones adecuadas sobre customers.csv:

2.1 Realiza una transformación adecuada del dataset customers.csv para su posterior procesamiento. (Tips: recuerda eliminar ids, transformar variables discretas, y cambiar fechas a días hasta el día actual, por ejemplo.)

Primero elimino las variables customer id, z cozt contract, z revenue y response ya que no nos seran utiles durante el ejercicio.

In [11]:
```
customers_numerico = customers.drop(columns=['customer_id','Z_CostContact',
```

Como nos encontramos con dos variables categoricas (educacion y marital status), vamos a tranformarlas a variables numericas para poder analizarlas posteriormente.

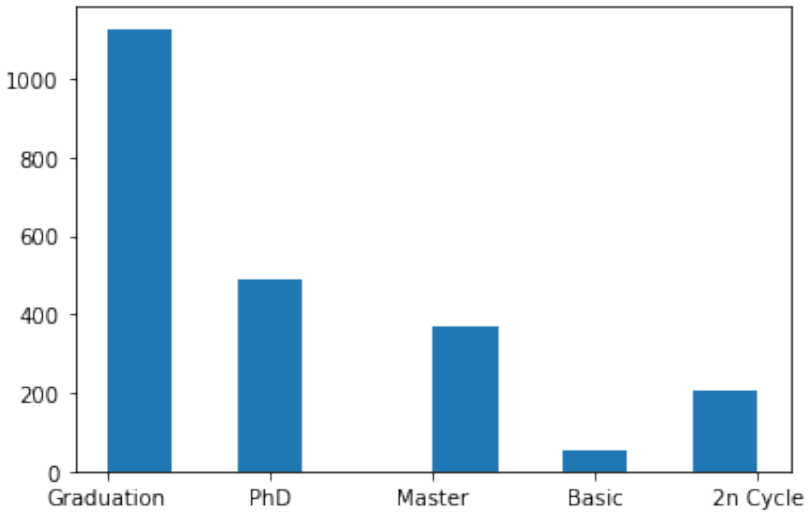
In [12]:
```
customers["Education"].value_counts()
```

Out[12]:
```
Graduation    1127
PhD            486
Master         370
2n Cycle       203
Basic           54
Name: Education, dtype: int64
```

En este grafico de el nivel de educacion, podemos ver que la mayoria de nuestros clientes estan graduados, esto nos muestra que la mayoria tienen por lo menos un grado universitario.

In [13]:
```
plt.hist(customers["Education"])
```

Out[13]:
```
(array([1127.,    0.,  486.,    0.,    0.,  370.,    0.,   54.,    0.,
         203.]),
 array([0. , 0.4, 0.8, 1.2, 1.6, 2. , 2.4, 2.8, 3.2, 3.6, 4. ]),
 <BarContainer object of 10 artists>)
```



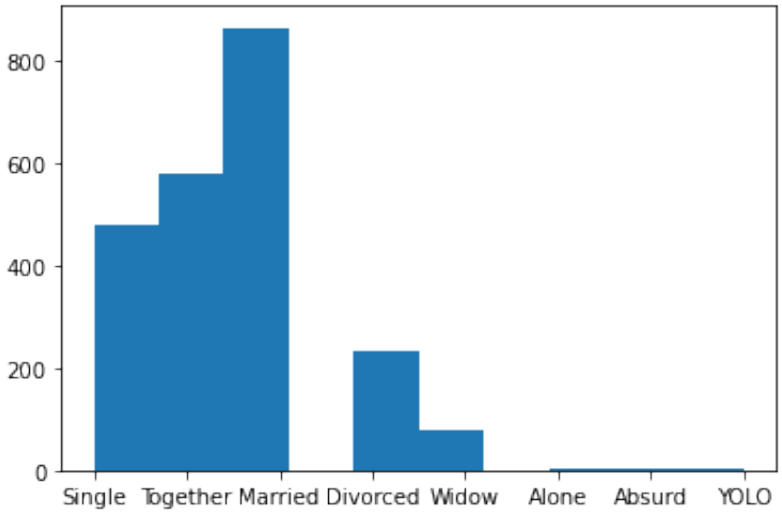

In [14]:
```
customers["Marital_Status"].value_counts()
```

Out[14]:
```
Married     864
Together    580
Single      480
Divorced    232
Widow        77
Alone         3
Absurd        2
YOLO          2
Name: Marital_Status, dtype: int64
```

En este grafico de estado marital podemos ver que la mayoria de nuestros clientes estan casados.

In [15]:
```
plt.hist(customers["Marital_Status"])
```

Out[15]:
```
(array([480., 580., 864.,   0., 232.,  77.,   0.,   3.,   2.,   2.]),
 array([0. , 0.7, 1.4, 2.1, 2.8, 3.5, 4.2, 4.9, 5.6, 6.3, 7. ]),
 <BarContainer object of 10 artists>)
```




Usamos dummies para transformar las variables categoricas no ordinales ya que en vez de ser 0,1,2,3 que asi les estariamos dando un rango de importancia que no queremos, por lo que ahora se diferencian con 0 y 1 donde 0 es cuando no hay y 1 cuando si.

In [16]:
```
customers_numerico= pd.get_dummies(customers_numerico, columns=["Marital_St
```

Por ultimo, vamos a tranformar la variable Dt Customer ya que era una fecha y lo pasamos a dias. Utilizamos una funcion que actualiza los dias a ahora cada vez que lo runeamos.

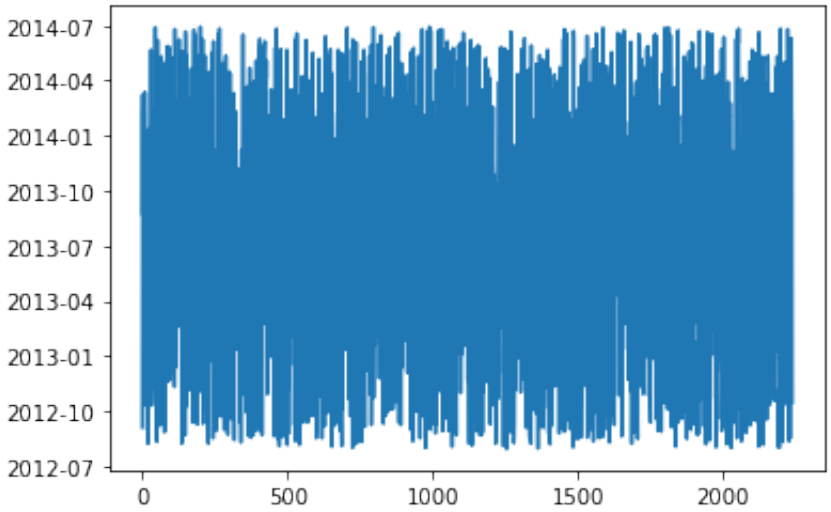
In [17]:
```
from datetime import datetime
from datetime import timedelta
```

In [18]:
```
customers['Dt_Customer'] = pd.to_datetime(customers['Dt_Customer'], format=
```

Para ver si hay alguna fecha incorrecta creamos un plot. En el observamos que las fechas van de julio de 2012 a julio de 2014 y vemos que no hay anomalias.

In [19]:
```
customers["Dt_Customer"].plot()
```

Out[19]: <AxesSubplot:>




In [20]:
```
now = datetime.now()
dias=now-customers['Dt_Customer']
```

In [21]:
```
customers['Days']=dias
customers_numerico['Days']=dias
```

Cambiamos los datos que estaban como Object a numerico para poder analizarlos y normalizarlos.

In [22]:
```
customers_numerico= customers_numerico.apply(pd.to_numeric, errors='coerce'
```

### 2.2.- ¿Debes normalizar los datos?. Si es así, realiza la normalización.

Al normalizar los datos, hacemos que los valores de una columna estén a una misma escala, esto nos es muy útil para columnas con rangos diferentes y asi poderlos comparar objetivamente, realizar los futuros clusters y la reduccion dimensional. Nosotros hemos normalizado los valores de -1 al 1, para esto hemos usado una funcion donde restamos a cada valor su media y dividimos este resultado por la desviacion estandar.

In [23]:
```
customers_numerico_normalizado=(customers_numerico-customers_numerico.mean(
```

Al normalizar los datos nos damos que Dt customer no se puede normalizar por lo que la eliminamos.

In [24]:
```
customers_numerico_normalizado= customers_numerico_normalizado.drop(columns
```

2.3.- Puede ser interesante realizar un análisis descriptivo con algún mapa de correlación y un t-sne, pca o similar para ver como se distribuyen los datos y eliminar outliers (recuerda eliminarlos también de la tabla que relaciona clientes y campañas)

Al realizar el analisis descriptivo de la correlacion, hemos eliminado las dummies creadas ya que sino nos salia un grafico muy aglomerado y confuso.

In [26]:
```
customers_corr=customers_numerico_normalizado.drop(columns=['Marital_Status
       'Marital_Status_Divorced', 'Marital_Status_Married',
       'Marital_Status_Single', 'Marital_Status_Together',
       'Marital_Status_Widow', 'Marital_Status_YOLO', 'Education_2n Cycle',
       'Education_Basic', 'Education_Graduation', 'Education_Master',
       'Education_PhD'])
```

In [28]:
```
customers_corr
```

Out[28]:

| | Year_Birth | Income | Kidhome | Teenhome | Recency | MntWines | MntFruits | MntM |
|---|---|---|---|---|---|---|---|---|
| **0** | -0.985125 | 0.235274 | -0.825033 | -0.929687 | 0.306971 | 0.983562 | 1.551231 | |
| **1** | -1.235457 | -0.235774 | 1.032328 | 0.906732 | -0.383579 | -0.870285 | -0.636159 | |
| **2** | -0.317572 | 0.773461 | -0.825033 | -0.929687 | -0.797908 | 0.362642 | 0.570677 | |
| **3** | 1.267866 | -1.022504 | 1.032328 | -0.929687 | -0.797908 | -0.870285 | -0.560732 | |
| **4** | 1.017534 | 0.241465 | 1.032328 | -0.929687 | 1.549959 | -0.388998 | 0.419822 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **2235** | -0.150684 | 0.358488 | -0.825033 | 0.906732 | -0.107359 | 1.203409 | 0.419822 | |
| **2236** | -1.903010 | 0.469959 | 2.889690 | 0.906732 | 0.237916 | 0.303224 | -0.661302 | |
| **2237** | 1.017534 | 0.189064 | -0.825033 | -0.929687 | 1.446377 | 1.794620 | 0.545534 | |
| **2238** | -1.068569 | 0.678884 | -0.825033 | 0.906732 | -1.419402 | 0.368584 | 0.092971 | |
| **2239** | -1.235457 | 0.024832 | 1.032328 | 0.906732 | -0.314524 | -0.653409 | -0.585874 | |

2240 rows × 18 columns

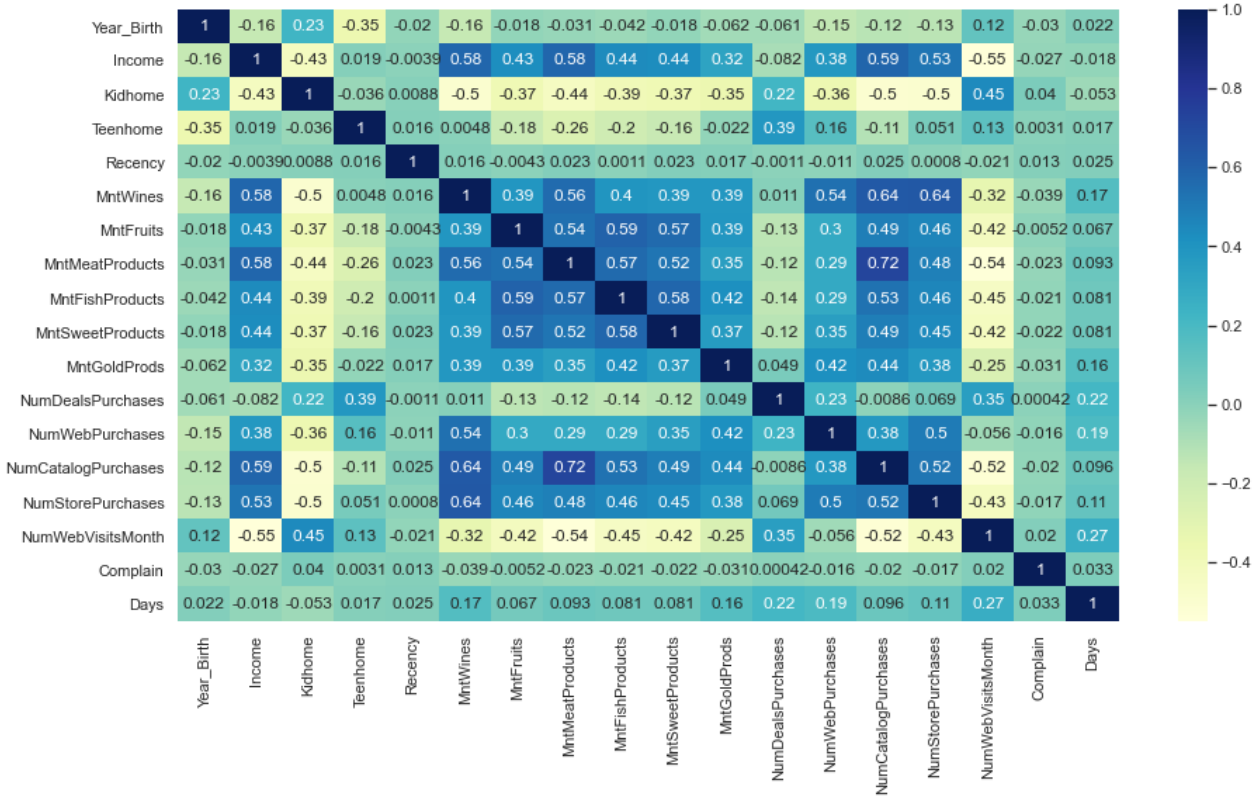
El mapa de calor nos proporciona un claro resumen visual de forma bidimensional a color de la información de nuestro dataset, donde cada color representa la correlacion entre las diferentes variables. Los colores mas intensos se atribuyen a los mas correlacionados y los mas claros a los menos correlacionados.

Visualmente destacamos la diagonal que siempre es 1 ya que es la correlacion de un valor consigo mismo, esto no nos interesa. Despues las dos variables mas correlacionadas son MntMeatProducts (cantidad gastada en carne en los dos últimos años) con NumCatalogPurchases (Número de compras a través de catálogo), con lo cual sabemos que los clientes cuando compran carne lo hacen a traves del catalogo.

Otra caracteristica a señalar, es que los clientes con un income alto hacen mucho gasto en vino, carne y compras en catalogo. En cambio vemos que no hacen uso de las ofertas.

Los clientes que hacen mayor uso de estas ofertas son aquellos con niños y adolescentes y que tienen un menor income.

In [27]:
```
sns.set(rc = {'figure.figsize':(15,8)})
dataplot = sns.heatmap(customers_corr.corr(),cmap="YlGnBu",annot=True)
```



PAra analizar la distribucion de los datos y los outliers, usaremos utilizaremos algoritmos de clusterizacion como t-SNE o DBscan y reduccion de componentes principales

Con las columnas del dataset, podemos usar t-SNE o PCA para reducirlo a dos dimensiones. PCA lo usaremos para detectar outliers y t-SNE junto a DBSCAN, para ver los datos graficamente.

DBSCAN
Density-based spatial clustering of applications with noise, llamado agrupamiento espacial basado en densidad de aplicaciones con ruido, es un algoritmo de data clustering que se basa en la densidad y asi encuentra el numero de clusters.

Al realizar el algorithmo necesitamos dos parámetros:

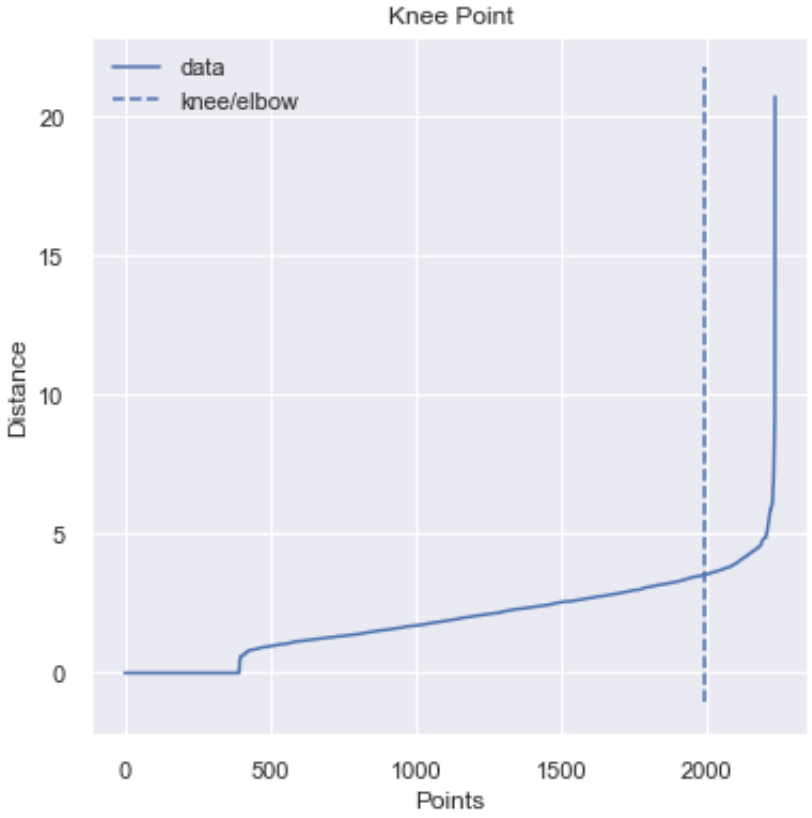
- Epsilon: Es lo cerca que deben de estar los puntos entre ellos para ser considerados parte de un clúster. Esto significa que, si la distancia es inferior o igual al épsilon, estos puntos se consideran vecinos. Para encontrar el epsilon hemos realizado un grafico de rodilla.
- Min samples: Es el numero mínimo de observaciones dentro de la región epsilon. Cuantos mas datos tengamos, mayor será el valor. Este valor es normalmente el doble de la dimension del dataframe.

In [145…
```
from sklearn.neighbors import NearestNeighbors
neigh = NearestNeighbors(n_neighbors=2)
nbrs = neigh.fit(customers_numerico_normalizado)
distances, indices = nbrs.kneighbors(customers_numerico_normalizado)
```

In [146…
```
distances = np.sort(distances, axis=0)
distances = distances[:,1]
```

In [147…
```
from kneed import KneeLocator

i = np.arange(len(distances))
knee = KneeLocator(i, distances, S=1, curve='convex', direction='increasing

fig = plt.figure(figsize=(5, 5))
knee.plot_knee()
plt.xlabel("Points")
plt.ylabel("Distance")

print(distances[knee.knee])
```

```
3.5247674437941883
<Figure size 360x360 with 0 Axes>
```




En el grafico de rodilla obtenemos que la distacia del epsilon es 3.4541262006086355 que es el ratio que rodea el corplot.

In [148…
```
customers_numerico_normalizado.ndim
```

Out[148… 2

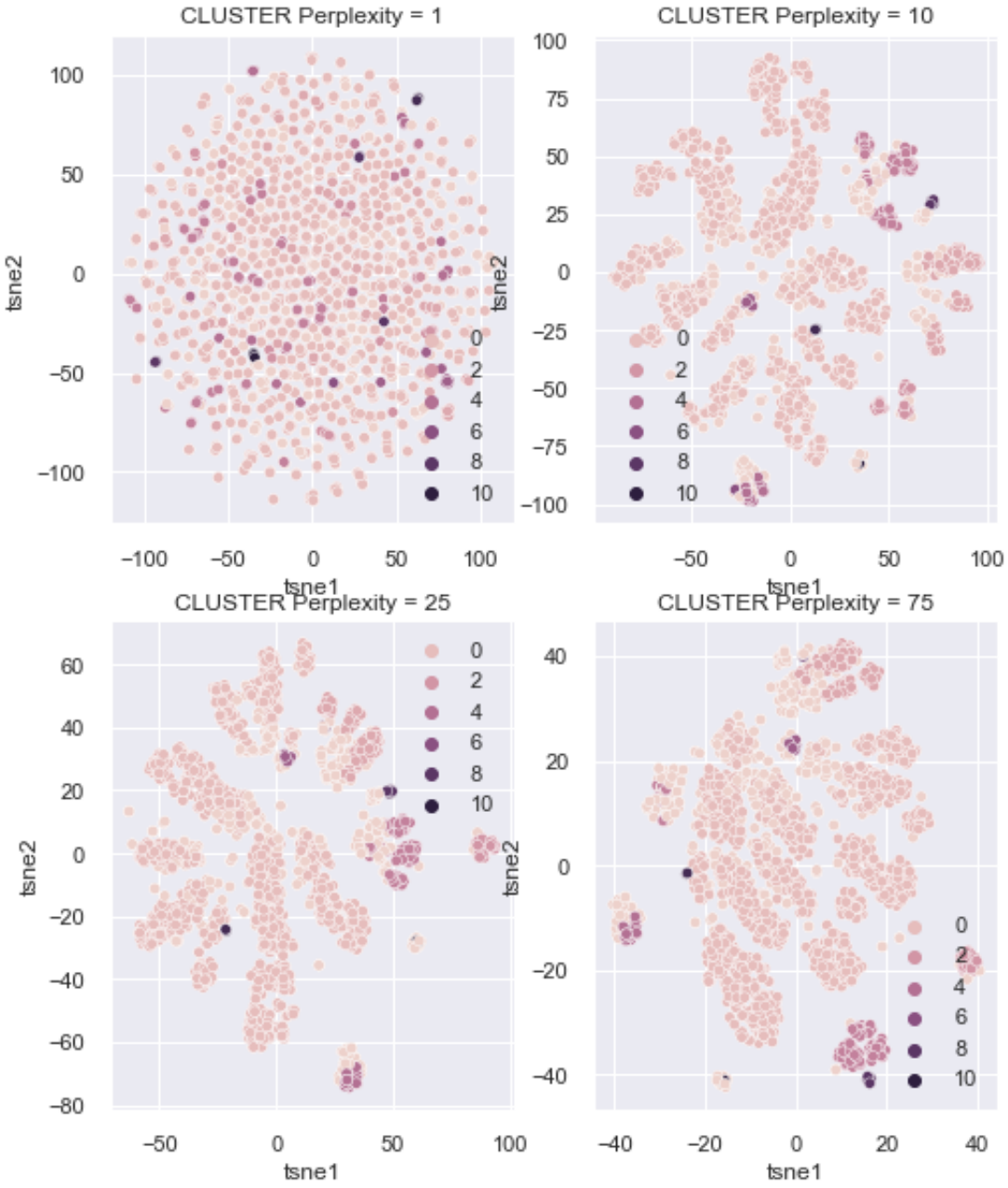
Aqui hemos calculado la dimension de nuestro dataframe, es 2, por lo que el min samples sera 4, ya que es el doble de la dimension.

In [149…
```
epsilon = 3.4541262006086355
min_samples = 4

db = DBSCAN(eps=epsilon, min_samples=min_samples)
labels=db.fit_predict(customers_numerico_normalizado)
```

Representamos los datos relacionados con labels que hemos obtenido anteriormente en DBScan.

In [150…

```
plt.figure(figsize =(8,4))
plt.subplots_adjust(top=2.0)

for index, p in enumerate([1,10,25,75]):
    tsne=TSNE(n_components=2, perplexity=p, random_state=100)
    tsne_results=tsne.fit_transform(customers_numerico_normalizado)

    tsne_results=pd.DataFrame(tsne_results, columns = ['tsne1', 'tsne2'])

    plt.subplot(2,2,index+1)
    sns.scatterplot(tsne_results['tsne1'], tsne_results['tsne2'],hue=labels
    plt.title('CLUSTER Perplexity = '+ str(p))

plt.show()
```



Obtenemos 4 graficos con una perplexidad diferente, pero eligiriamos el de la perplexidad = 25 ya que es el que mas agrupados estan.

T-SNE

Esto es una técnica de reducción de la dimensionalidad que se utiliza para representar un conjunto de datos de alta dimensión en un espacio de baja dimensión de dos o tres dimensiones para poder visualizarlo. A diferencia de otros algoritmos de reducción de la dimensionalidad, como el PCA, que se limita a maximizar la varianza, el t-SNE crea un espacio de características reducido en el que las muestras similares son modeladas por puntos cercanos y las muestras disímiles son modeladas por puntos distantes con alta probabilidad.

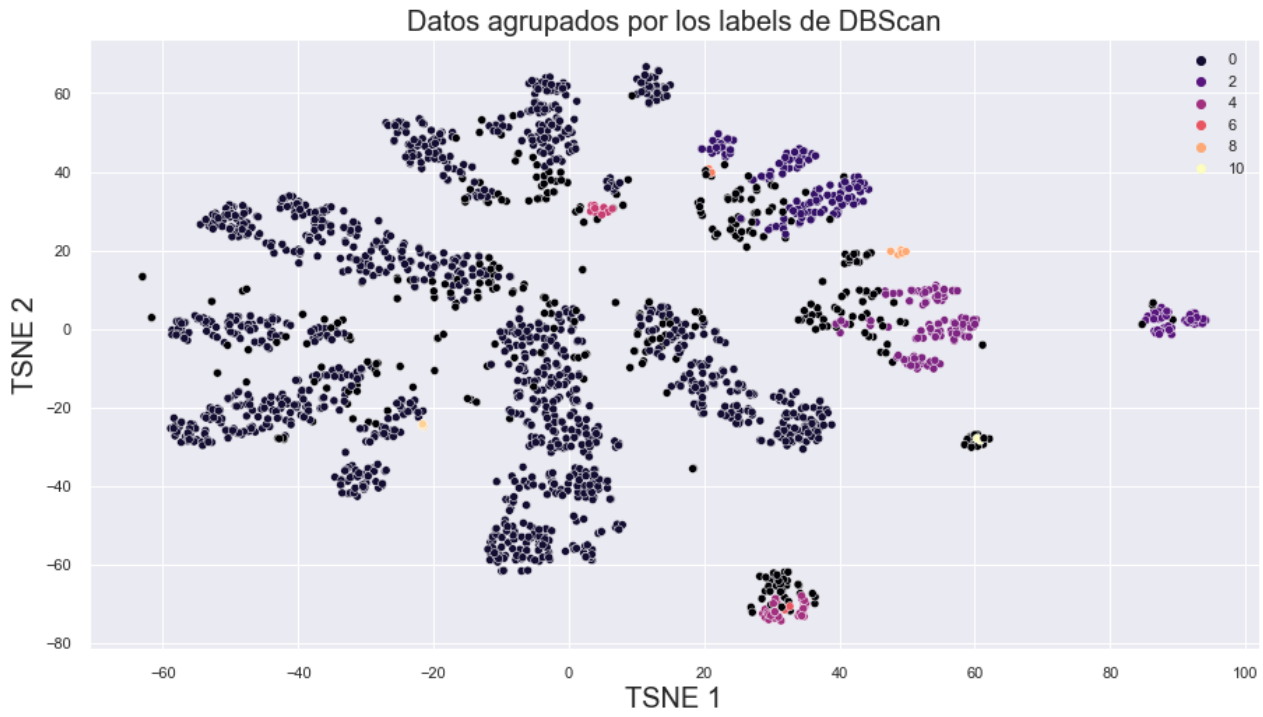
Elejimos 75 para la perplejidad ya que es la que mas se ha ajustado antes, el numero de componentes 2 ya que es la dimension y por ultimo 100.

In [151…

```
tsne_customers= TSNE(n_components=2,perplexity=25,random_state=100)
data_tsne=tsne_customers.fit_transform(customers_numerico_normalizado)
```

In [152…

```
sns.scatterplot(x=data_tsne[:,0],y=data_tsne[:,1],palette="magma",sizes=(50
plt.title("Datos agrupados por los labels de DBScan", fontsize = 20)
plt.xlabel("TSNE 1", fontsize = 20)
plt.ylabel("TSNE 2", fontsize = 20)
plt.legend(loc='best')
plt.show()
```



## PCA

Usaremos PCA para detectar outliers. Vamos a reducir los componentes, para ello haremos un train y test del dataframe de customers.

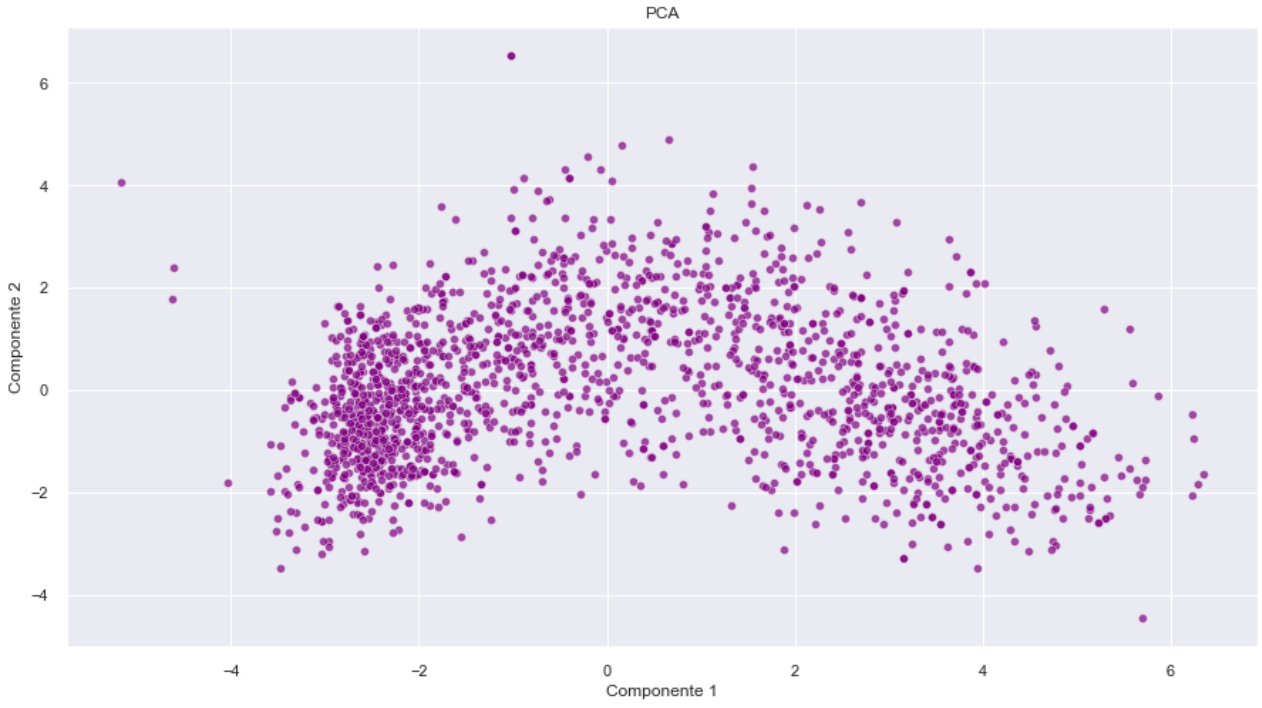
In [153…
```
trainpca,testpca = train_test_split(customers_numerico_normalizado, test_si
```

In [156…
```
pca = PCA(n_components=31, random_state=100,whiten=False)
pcaNumModel = pca.fit(trainpca)
principalComponents = pcaNumModel.transform(trainpca)
```

In [163…
```
x_trans = principalComponents[:, 0]
y_trans = principalComponents[:, 1]
```

In [164…
```
sns.scatterplot(x_trans, y_trans,
                sizes=(4,4),color='purple',alpha=0.7)
plt.title('PCA')
plt.xlabel('Componente 1')
plt.ylabel('Componente 2')
```

Out[164… `Text(0, 0.5, 'Componente 2')`




Aunque no buscamos una reduccion dimensional, asi quedarian las dos primeras componentes reducidas.

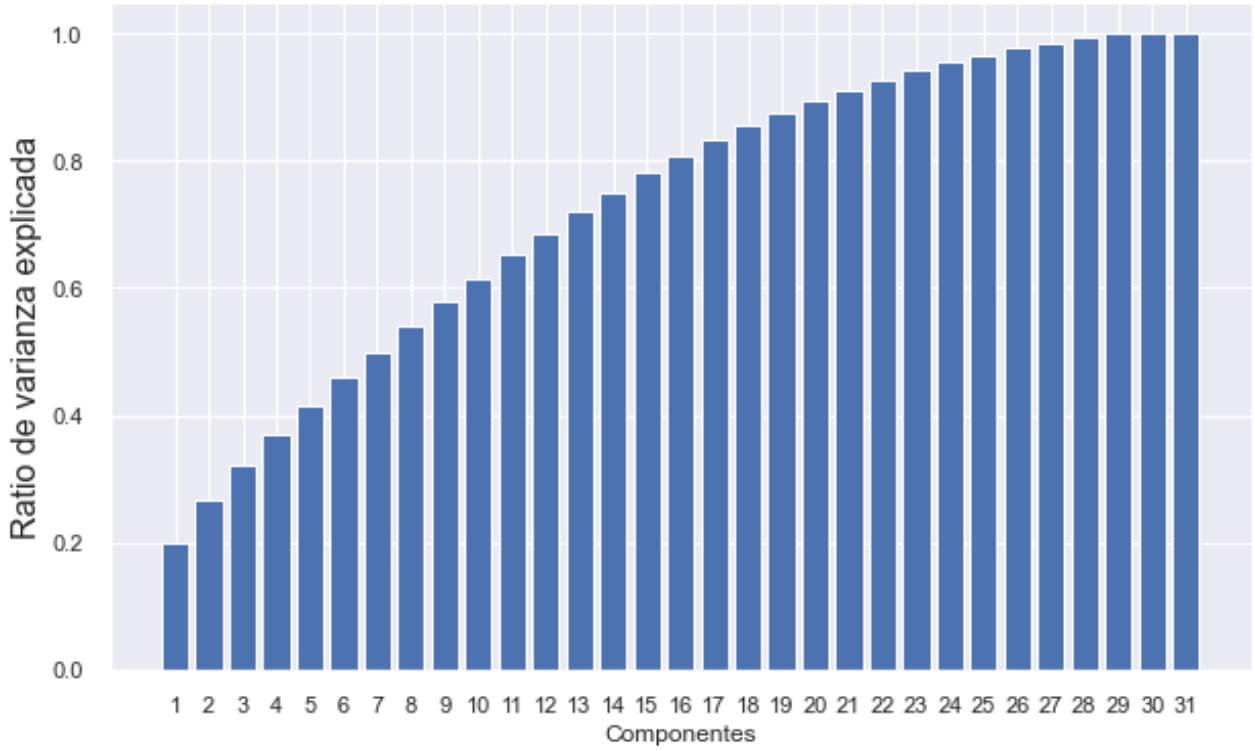
Usamos las 31 columnas para usar todos los datos, ya que queremos detectar los outliers.

In [165…

```
print(np.sum(pcaNumModel.explained_variance_ratio_))

ev1=np.cumsum(pcaNumModel.explained_variance_ratio_)

fig, ax = plt.subplots(figsize=(10, 6))

ax.bar(np.arange(1, len(ev1) + 1, 1), ev1)
ax.set_xticks(np.arange(1, len(ev1) + 1, 1))
ax.set_xlabel('Componentes')
ax.set_ylabel('Ratio de varianza explicada', fontsize=16)
```

```
1.0
```

Out[165… `Text(0, 0.5, 'Ratio de varianza explicada')`




Aunque cogemos todas las variables, vemos que con 16 componentes ya podríamos explicar 80% de la varianza. Ahora vamos a calcular el MSE y asi poder ver los outliers representados.

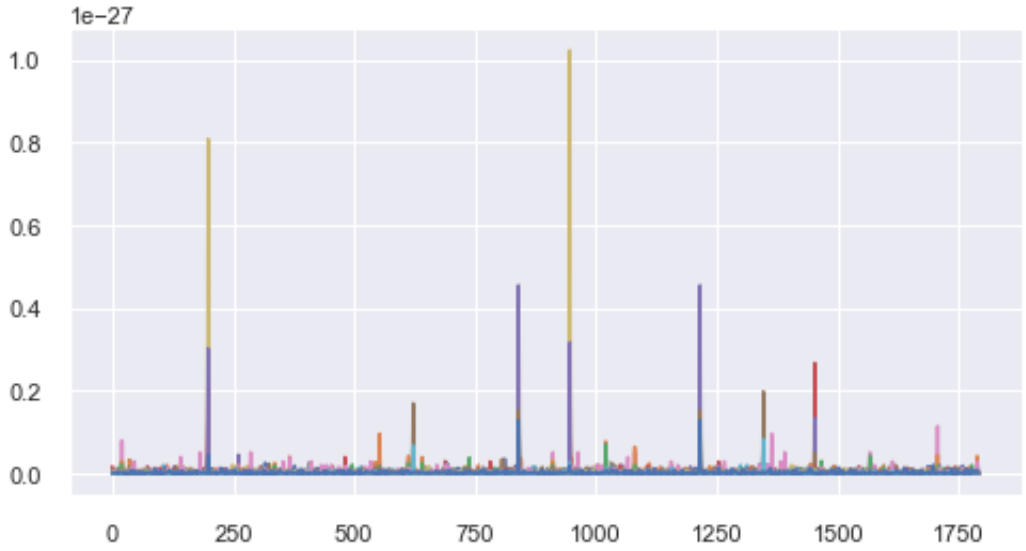
In [169…

```
trainpca_numpy= trainpca.to_numpy()
train_inverse=pca.inverse_transform(principalComponents)
#MSE
mse_pca= (trainpca_numpy - train_inverse)**2

total=np.sum(mse_pca,1)

total_mean = np.mean(total)

total_std = np.std(total)

outlier= np.where(total > total_mean + 3*total_std)
```

Para poder clasificar una observacion como outlier, tiene que ser superior a la media mas tres veces la desviacion estandar.

In [170…
```
plt.plot(mse_pca)
```

Out[170…
```
[<matplotlib.lines.Line2D at 0x7fc5173705b0>,
 <matplotlib.lines.Line2D at 0x7fc517370bb0>,
 <matplotlib.lines.Line2D at 0x7fc5173708b0>,
 <matplotlib.lines.Line2D at 0x7fc517370e20>,
 <matplotlib.lines.Line2D at 0x7fc5173701c0>,
 <matplotlib.lines.Line2D at 0x7fc517bb51f0>,
 <matplotlib.lines.Line2D at 0x7fc518eb2b20>,
 <matplotlib.lines.Line2D at 0x7fc5173613d0>,
 <matplotlib.lines.Line2D at 0x7fc5173613a0>,
 <matplotlib.lines.Line2D at 0x7fc517361760>,
 <matplotlib.lines.Line2D at 0x7fc5179df4c0>,
 <matplotlib.lines.Line2D at 0x7fc517361af0>,
 <matplotlib.lines.Line2D at 0x7fc5173614f0>,
 <matplotlib.lines.Line2D at 0x7fc517361280>,
 <matplotlib.lines.Line2D at 0x7fc517361370>,
 <matplotlib.lines.Line2D at 0x7fc5173618e0>,
 <matplotlib.lines.Line2D at 0x7fc517361fd0>,
 <matplotlib.lines.Line2D at 0x7fc5173616a0>,
 <matplotlib.lines.Line2D at 0x7fc5173610d0>,
 <matplotlib.lines.Line2D at 0x7fc517361160>,
 <matplotlib.lines.Line2D at 0x7fc5173619a0>,
 <matplotlib.lines.Line2D at 0x7fc517361880>,
 <matplotlib.lines.Line2D at 0x7fc517361ac0>,
 <matplotlib.lines.Line2D at 0x7fc517bfb3a0>,
 <matplotlib.lines.Line2D at 0x7fc5179adee0>,
 <matplotlib.lines.Line2D at 0x7fc517a15670>,
 <matplotlib.lines.Line2D at 0x7fc517a15220>,
 <matplotlib.lines.Line2D at 0x7fc517a151c0>,
 <matplotlib.lines.Line2D at 0x7fc517a153d0>,
 <matplotlib.lines.Line2D at 0x7fc517a15f10>,
 <matplotlib.lines.Line2D at 0x7fc517a15bb0>]
```

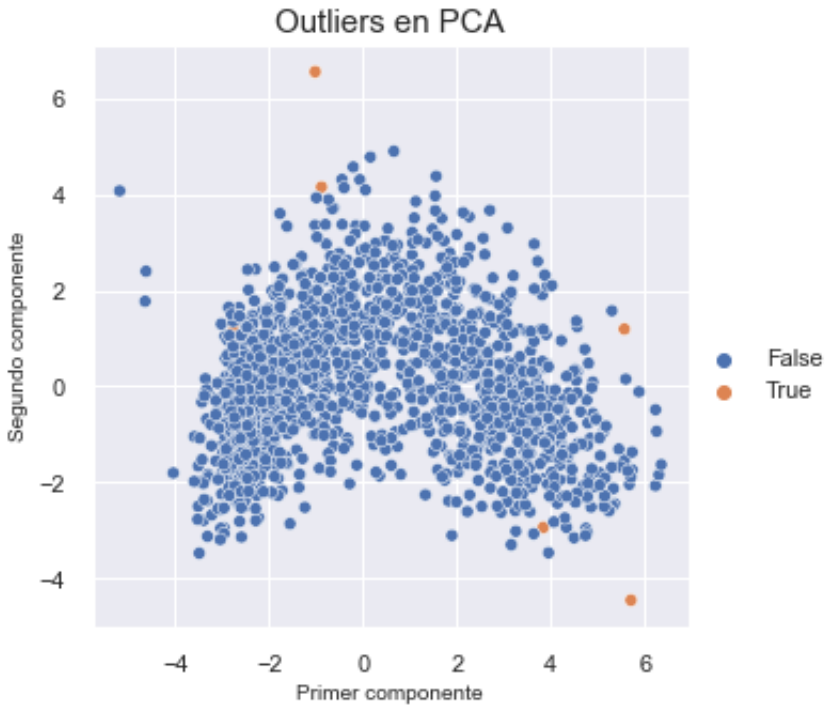


In [171…

```
mark_outlier = np.where(total > total_mean + 3*total_std, True, False)
plt.subplots(figsize=(5,5))

sns.scatterplot(x=x_trans, y=y_trans, hue= mark_outlier)
plt.xlabel("Primer componente", fontsize = 10)
plt.ylabel("Segundo componente", fontsize = 10)
plt.title("Outliers en PCA", fontsize = 15)
plt.legend(bbox_to_anchor=(1, 0.5), loc=2, borderaxespad=0.1)
sns.set(rc={"figure.figsize":(8, 4)})
plt.figure(num=None, figsize=(10, 10), dpi=80, facecolor='r', edgecolor='k'
```

Out[171…

```
<Figure size 800x800 with 0 Axes>
```




```
<Figure size 800x800 with 0 Axes>
```

Podemos observar la representacion del PCA, destacando los outliers en naranja. Estos ultimos aunque haya pocos los vamos a tratar a continuacion.

### OUTLIERS

Utilizando la media y desviacion tipica identificamos los outliers. Creamos una tabla donde aparezcan solo los outliers. El rango es de -33.4 a 25.9, por lo que los outliers seran todo numero fuera del rango.

In [172…

```
logvero = pcaNumModel.score_samples(customers_numerico_normalizado)
mean = logvero.mean()
std = logvero.std()

print(mean-std)
print(mean+std)
logvero

# Voy a calcular los outliers restandole a la media tres veces su desviaci

outliers = customers_numerico_normalizado.loc[logvero <= mean-3*std,]
outliers
```

```
-33.40549682778025
25.885049829394326
```

Out[172…

| | Year_Birth | Income | Kidhome | Teenhome | Recency | MntWines | MntFruits |
|---|---|---|---|---|---|---|---|
| **21** | 0.850646 | -1.989003e+00 | 1.032328 | -0.929687 | -0.245469 | -0.899994 | -0.636159 |
| **27** | 1.434754 | 8.717969e-16 | 1.032328 | -0.929687 | -1.039600 | -0.888111 | -0.636159 |
| **131** | -0.901681 | 3.628014e-01 | 1.032328 | 0.906732 | -0.245469 | 0.683500 | -0.535589 |
| **138** | 0.349981 | -6.545005e-01 | 1.032328 | 0.906732 | -0.418106 | -0.858402 | -0.661302 |
| **153** | 1.601643 | -7.217588e-01 | 1.032328 | -0.929687 | -1.281293 | -0.888111 | -0.485305 |
| **164** | 0.349981 | 4.193490e+00 | -0.825033 | 0.906732 | 1.688069 | -0.843547 | -0.611017 |
| **687** | 1.100978 | 4.335675e+00 | -0.825033 | -0.929687 | -0.970545 | -0.739565 | -0.259023 |
| **1653** | 0.683757 | 4.189616e+00 | -0.825033 | -0.929687 | -1.246765 | -0.899994 | -0.661302 |
| **1806** | -0.234128 | -1.801407e+00 | -0.825033 | 2.743150 | 1.480904 | -0.662322 | -0.560732 |
| **1898** | -1.986454 | 2.455757e+00 | -0.825033 | -0.929687 | -1.384875 | -0.885140 | -0.611017 |
| **1975** | 0.016205 | -1.909883e+00 | -0.825033 | 0.906732 | -1.695622 | -0.855431 | -0.560732 |
| **2093** | 2.018863 | 1.078240e+00 | -0.825033 | -0.929687 | 0.306971 | 0.496333 | 1.903224 |
| **2134** | -0.985125 | 5.287905e-01 | -0.825033 | -0.929687 | -0.038304 | -0.189947 | 1.023240 |
| **2177** | 0.349981 | -1.523797e-01 | -0.825033 | 0.906732 | -1.592040 | 0.053667 | -0.585874 |
| **2202** | 0.349981 | -1.523797e-01 | -0.825033 | 0.906732 | -1.592040 | 0.053667 | -0.585874 |
| **2233** | 0.683757 | 2.453965e+01 | 1.032328 | -0.929687 | -0.901490 | -0.876227 | -0.309308 |

16 rows × 31 columns

Despues de identificar los outliers, retiramos de nuestra tabla aquellos clientes que pertenecen a ellos. Igualmente los eliminamos de campains to customers.

In [173…

```
customers_numerico_normalizado_sin=customers_numerico_normalizado.drop(outl
```

Verificamos que hemos eliminado los outliers, para ellos vemos solo del 19 al 31. Y vemos que hemos hecho bien la limpieza ya que no aparecen los clientes 21 y 27.

In [174…
```
customers_numerico_normalizado_sin[19:30]
```

Out[174…

| | Year_Birth | Income | Kidhome | Teenhome | Recency | MntWines | MntFruits | MntMea |
|---|---|---|---|---|---|---|---|---|
| **19** | 1.351310 | -0.736297 | 1.032328 | -0.929687 | 1.273740 | -0.891082 | -0.233881 | |
| **20** | 1.100978 | -0.607372 | -0.825033 | -0.929687 | -0.279996 | -0.647467 | -0.611017 | |
| **22** | -1.652678 | 0.254006 | -0.825033 | 0.906732 | 0.479608 | 1.672812 | -0.661302 | |
| **23** | -1.235457 | 0.522280 | -0.825033 | 0.906732 | -1.695622 | 0.237864 | -0.661302 | |
| **24** | -1.485789 | -0.461632 | -0.825033 | 0.906732 | 0.686773 | -0.100820 | -0.585874 | |
| **25** | 0.016205 | -1.344298 | -0.825033 | -0.929687 | 1.377322 | -0.885140 | -0.560732 | |
| **26** | 0.600313 | 0.044403 | 1.032328 | 0.906732 | -1.557512 | -0.388998 | -0.560732 | |
| **28** | 1.685087 | -0.554651 | 1.032328 | -0.929687 | -0.797908 | -0.796012 | -0.611017 | |
| **29** | -0.317572 | 1.292875 | -0.825033 | -0.929687 | 1.619014 | 1.129136 | 1.852940 | |
| **30** | 1.685087 | -1.648238 | -0.825033 | -0.929687 | -0.521688 | -0.879198 | -0.560732 | |
| **31** | -0.484460 | -0.544267 | -0.825033 | -0.929687 | 0.237916 | -0.570223 | -0.233881 | |

11 rows × 31 columns

Hemos quitado los outliers de customers, y como sabemos que customers son outliers, los podemos eliminar de campaigns_to_customers.

In [175…
```
indice_outliers=[21,27,131,138,153,164,687,1653,1806,1898,1975,2093,2134,21
```

In [176…
```
campaigns_to_customers=campaigns_to_customers.set_index('customer',drop=Fal
```

Retiramos los outliers haciendo que le indice coincida con el numero de los clientes.

In [177…
```
campaigns_to_customers.drop(outliers.index)
```

Out[177…

| customer | Unnamed: 0 | customer | campaign | result |
|---|---|---|---|---|
| **1650** | 0 | 1650 | 34 | 1 |
| **1168** | 1 | 1168 | 85 | 0 |
| **2142** | 2 | 2142 | 36 | 0 |
| **1536** | 3 | 1536 | 12 | 0 |
| **1072** | 4 | 1072 | 74 | 0 |
| **...** | ... | ... | ... | ... |
| **706** | 13435 | 706 | 13 | 0 |
| **1628** | 13436 | 1628 | 86 | 0 |
| **1142** | 13437 | 1142 | 13 | 0 |
| **1510** | 13438 | 1510 | 91 | 0 |
| **890** | 13439 | 890 | 16 | 0 |

13348 rows × 4 columns

## 2.4.- Haz un clustering haciendo uso de K-means, de la base de datos de customers. Recuerda optimizar el valor adecuado de componentes (seleccionar adecuadamente la K).

### K-MEANS

Vamos a realiizar un cluster utilizando K-means. K-means es un algoritmo de clasificación no supervisada, tambien llamado clustering, que reune datos en k grupos basándose en sus características. Este agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo.

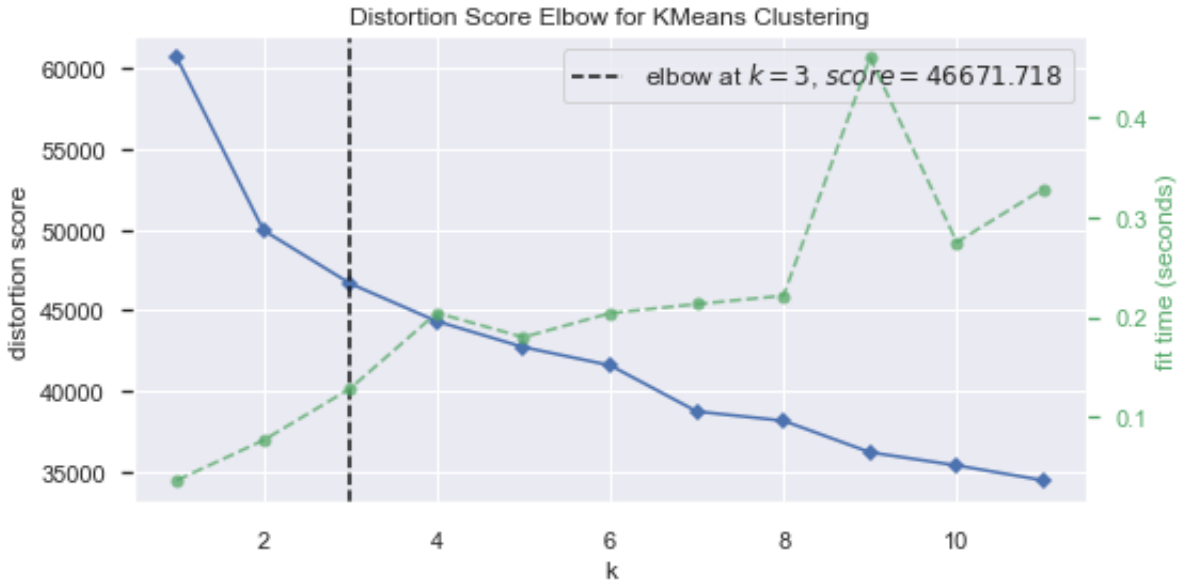
In [178…

```
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances_argmin_min
```

Para encontrar k, realizams este grafico de codo. Obtenemos un 3, que va a ser nuestro numero de clusters.

In [179…

```
from yellowbrick.cluster import KElbowVisualizer
model = KMeans()
visualizer = KElbowVisualizer(model, k=(1,12)).fit(customers_numerico_norma
visualizer.show()
```




Out[179… <AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>

Creamos una variable que crea los 3 clusters.

In [180…
```
kmeans_customers = KMeans(n_clusters=3).fit(customers_numerico_normalizado_
```

Obtenemos los centroides a los que se une cada punto.

In [181…
```
centroids_customers = kmeans_customers.cluster_centers_
print(centroids_customers)
```

```
[[-4.68453039e-02  9.13216513e-01 -7.54703038e-01 -5.94643167e-01
   2.17180190e-02  8.61764632e-01  1.11234192e+00  1.23671854e+00
   1.16365377e+00  1.08040614e+00  6.46346245e-01 -5.52118691e-01
   4.03063356e-01  1.07832757e+00  8.36986794e-01 -1.02696407e+00
  -2.58355456e-02 -2.98873903e-02 -3.66126086e-02 -3.48727373e-02
  -6.75215188e-02  8.17772651e-02  1.00663206e-02  4.74905683e-02
  -2.98873903e-02  2.60560478e-02 -1.45916721e-01  1.36579314e-01
  -7.39987153e-02 -6.28577472e-02 -5.72453205e-04]
 [ 2.61394197e-01 -7.03705107e-01  6.68610315e-01 -1.44736600e-01
   1.91503430e-02 -7.89745780e-01 -5.38738515e-01 -6.39621766e-01
  -5.53846384e-01 -5.31502303e-01 -5.50624276e-01 -1.67684337e-01
  -7.28103397e-01 -7.29782100e-01 -8.11481101e-01  4.77052559e-01
   3.68028035e-02 -2.98873903e-02 -3.66126086e-02 -2.01984615e-02
   1.98334315e-03  6.64323035e-02 -1.69750488e-02 -6.31734928e-02
  -2.98873903e-02  7.90385986e-02  1.86269603e-01 -3.20883923e-02
  -1.11591282e-02 -7.53864794e-02 -1.61588491e-01]
 [-3.79008760e-01  2.24561753e-01 -3.64337751e-01  7.71234446e-01
  -3.56479449e-02  4.74095174e-01 -1.56242986e-01 -1.51627800e-01
  -1.83274569e-01 -1.53412928e-01  2.54044487e-01  7.38502551e-01
   7.39513274e-01  1.23668102e-01  5.42603588e-01  2.06613876e-01
  -3.21146140e-02 -2.98873903e-02 -3.66126086e-02  7.22418147e-02
   6.54548110e-02 -1.77863823e-01  2.53047496e-02  6.11909871e-02
  -2.98873903e-02 -1.40661818e-01 -1.57135691e-01 -7.05998223e-02
   8.78337905e-02  1.62951754e-01  2.55399254e-01]]
```

Creamos una varible recogiendo cada etiqueta de cluster para poder usarla en el ejercicio 4.

In [182…
```
labels_customers=kmeans_customers.labels_
```

In [184…
```
sns.scatterplot(data=customers_numerico_normalizado_sin, x="Income", y="Mnt
plt.scatter(kmeans_customers.cluster_centers_[:,0], kmeans_customers.cluste
            marker="X", c="r", s=80, label="centroids")
plt.legend()
plt.show()
```


Representamos nuestros 3 grupos con sus centroides comparando Income con productos "Gourmet", ya que esta relacion nos parece interesante.

El primer grupo vemos que son clientes con menor poder adquisitivo, mientras que los otros dos gastan mas y tienen mas ingresos.

Observamos unos puntos dispersos perteneciendo al cluster 1, esto de puede tratar de outliers o de errores.

3.- Realiza las acciones adecuadas sobre campaigns.csv:

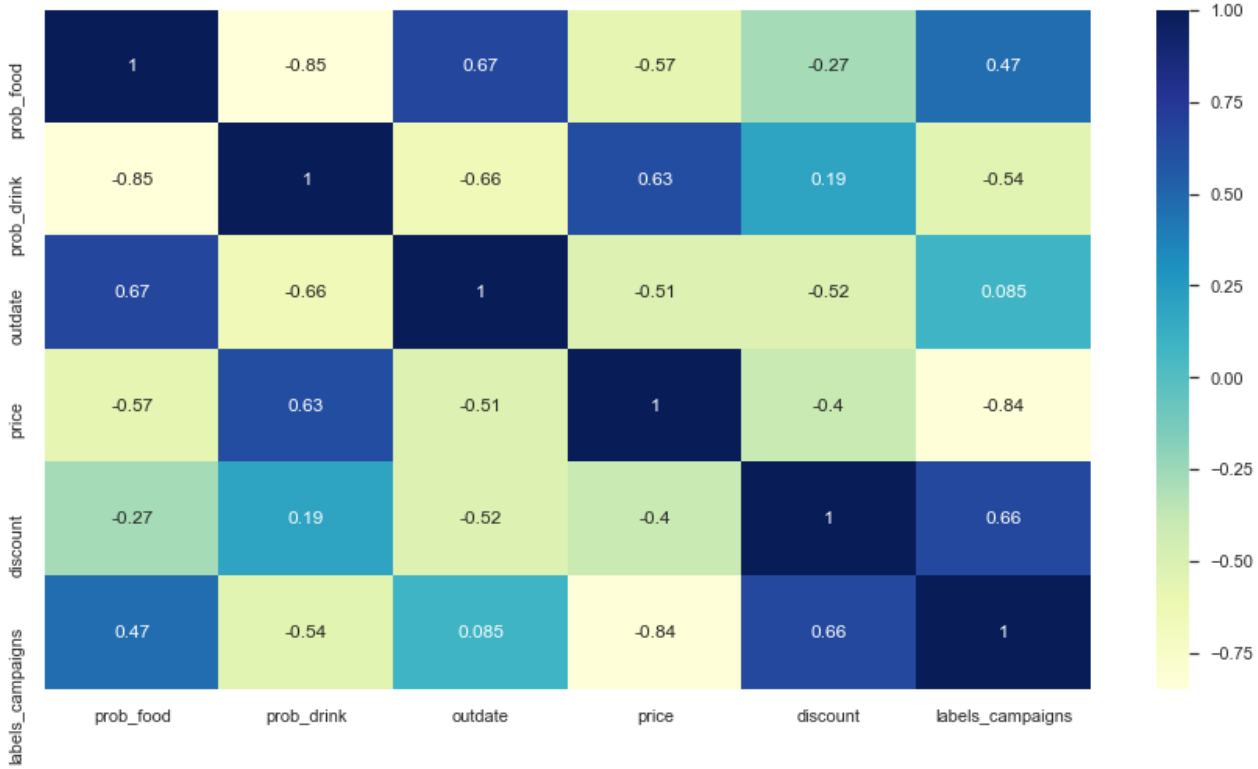
```
3.1 Transformación del dataset si es necesario.
3.2 Normaliza si es necesario.
3.3 Quizá sea interesante un pequeño análisis descriptivo.
3.4 Haz un clustering haciendo uso de K-means, de la base de
datos de customers. Recuerda optimizar el valor adecuado de
componentes.
```

Transformamos el dataset, para ello empezamos haciendo una limpieza de los datos que nos parecen innecesarios. Eliminamos la columna de id y la del nombre. Normalizamos porque vamos a hacer un mapa de correlacion y un K-means mas adelante.

In [185…
```
campaigns_numerico=campaigns.drop(columns=['campaign_id', 'name'])
```

In [186…
```
campaigns_numerico_normalizado=(campaigns_numerico-campaigns_numerico.mean(
```

In [187…
```
sns.set(rc = {'figure.figsize':(15,8)})
dataplot = sns.heatmap(campaigns_numerico_normalizado.corr(),cmap="YlGnBu",
```



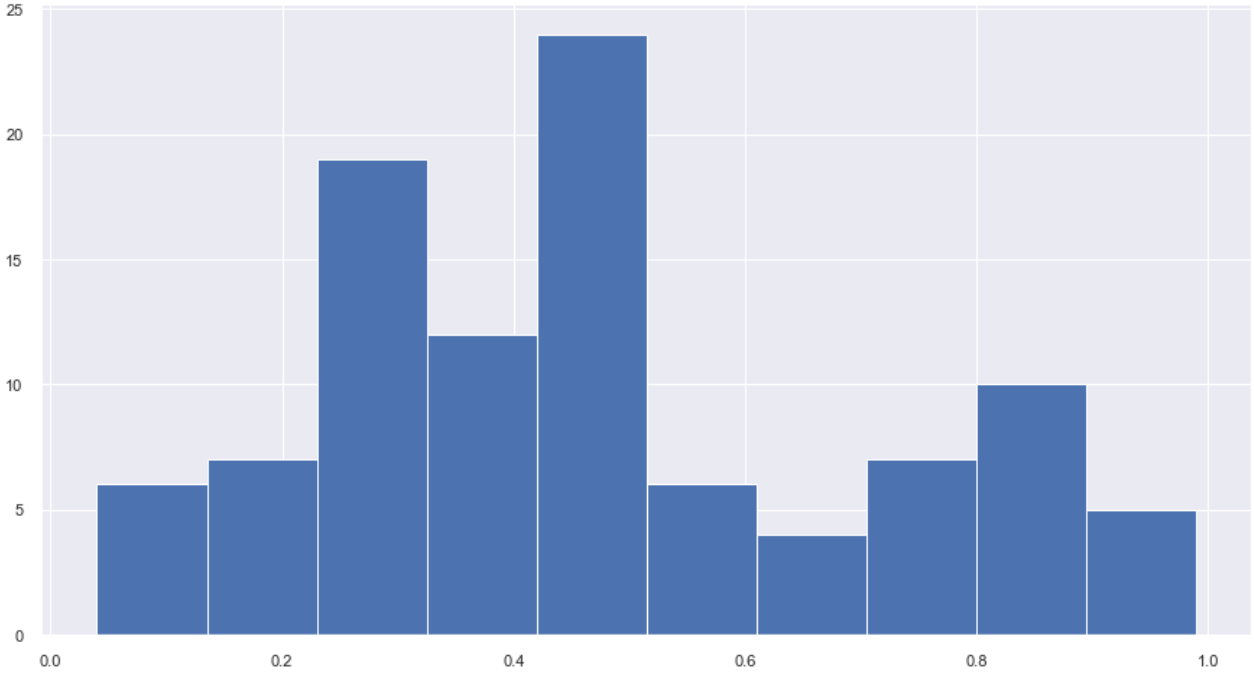

En este mapa de corrrelacion, observamos una alta correlacion entre las campañas de comida y los dias de duracion, por lo que las campañas de comida tienen mas duracion que las de bebida, que estan negativamente correlacionadas. Hay una correlacion alta entre el descuento y las campañas lo que no deberiamos de tener en cuento ya que no tiene sentido.

Analizamos la relacion que tienen las probabilidades de comida y las de bebida. Realizamos dos grficos de la relacion de precio con comida y bebida.

- Vemos que la probabilidad de que sea de comida se concentra más entre 0.4 a 0.7, y la de bebida entre 0.3 y 0.5.
- Vemos que la densidad de precio y probabilidad de comida está cuando el valor es más altos y en bebida más bajos.

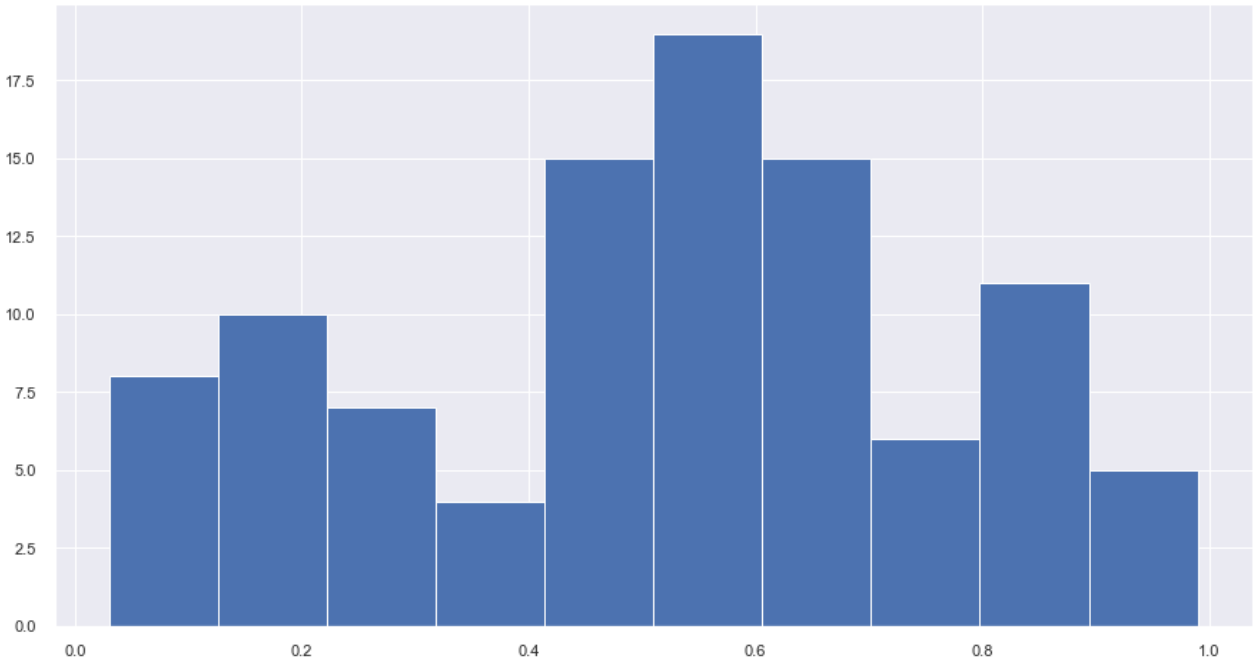
In [196…
```
plt.hist(campaigns["prob_drink"])
```

Out[196…
```
(array([ 6.,  7., 19., 12., 24.,  6.,  4.,  7., 10.,  5.]),
 array([0.04 , 0.135, 0.23 , 0.325, 0.42 , 0.515, 0.61 , 0.705, 0.8  ,
        0.895, 0.99 ]),
 <BarContainer object of 10 artists>)
```




In [197…
```
plt.hist(campaigns["prob_food"])
```

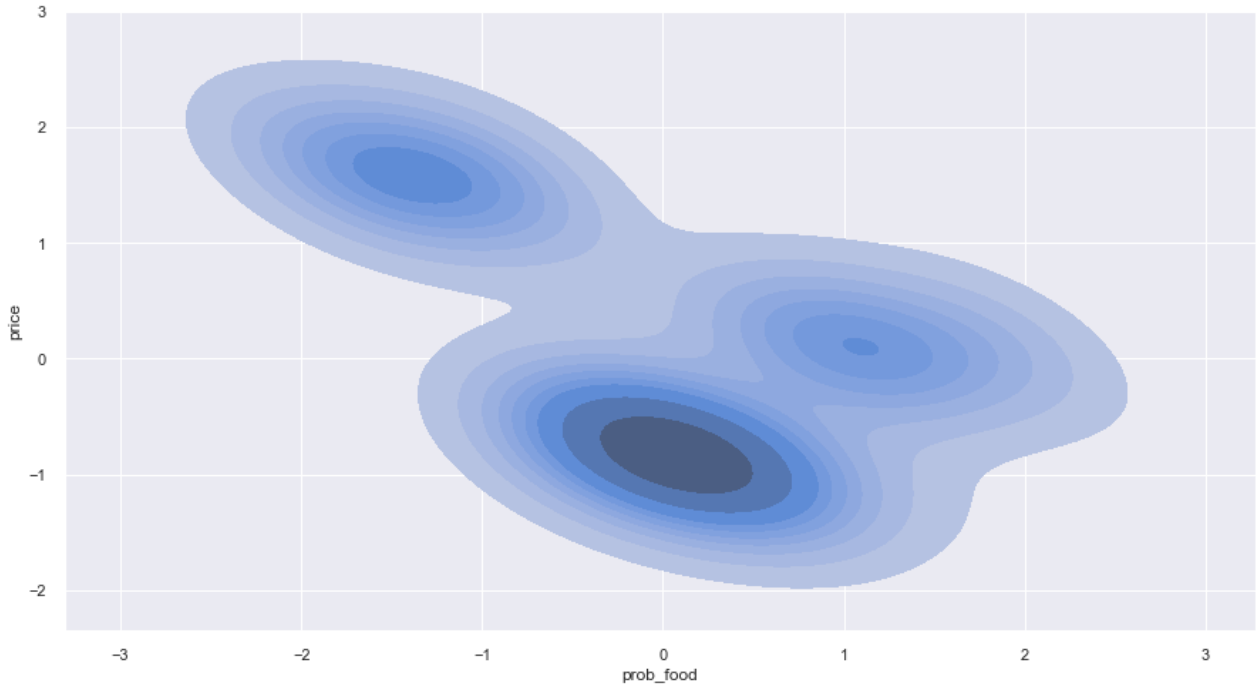
Out[197…
```
(array([ 8., 10.,  7.,  4., 15., 19., 15.,  6., 11.,  5.]),
 array([0.03 , 0.126, 0.222, 0.318, 0.414, 0.51 , 0.606, 0.702, 0.798,
        0.894, 0.99 ]),
 <BarContainer object of 10 artists>)
```



In [195…

```
import io
import requests
sns.kdeplot(campaigns_numerico_normalizado['prob_food'],campaigns_numerico_
```

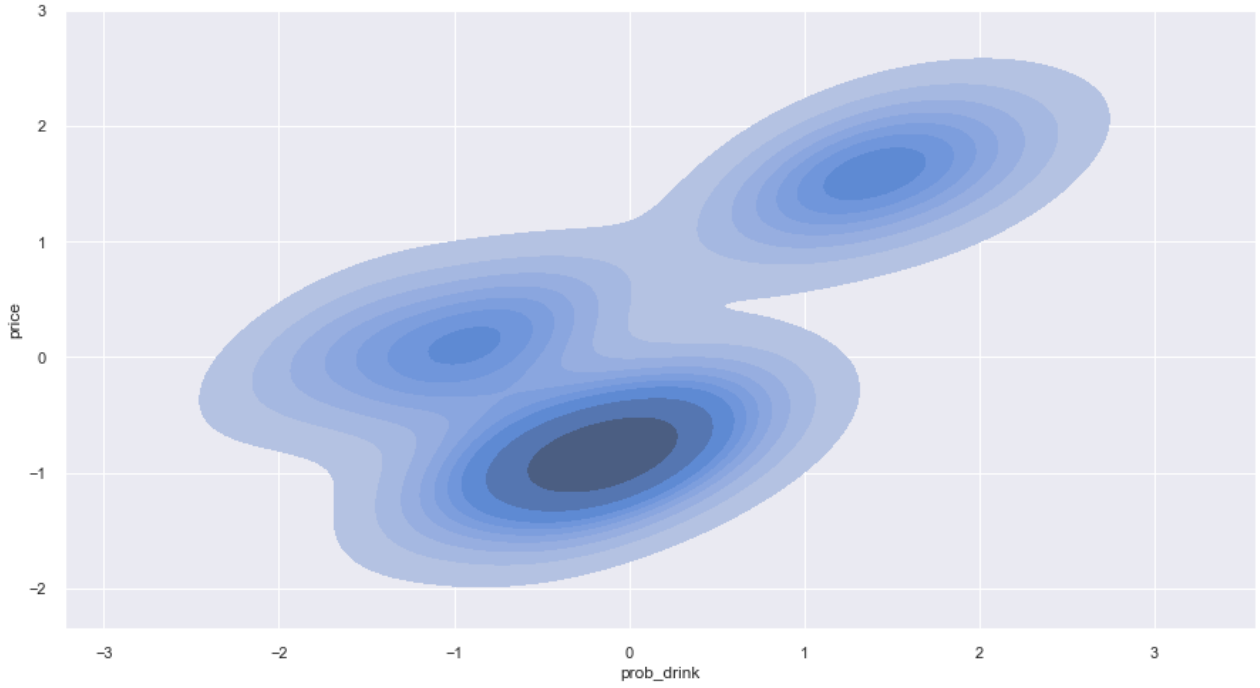
Out[195… <AxesSubplot:xlabel='prob_food', ylabel='price'>




In [198…

```
sns.kdeplot(campaigns_numerico_normalizado['prob_drink'],campaigns_numerico
```

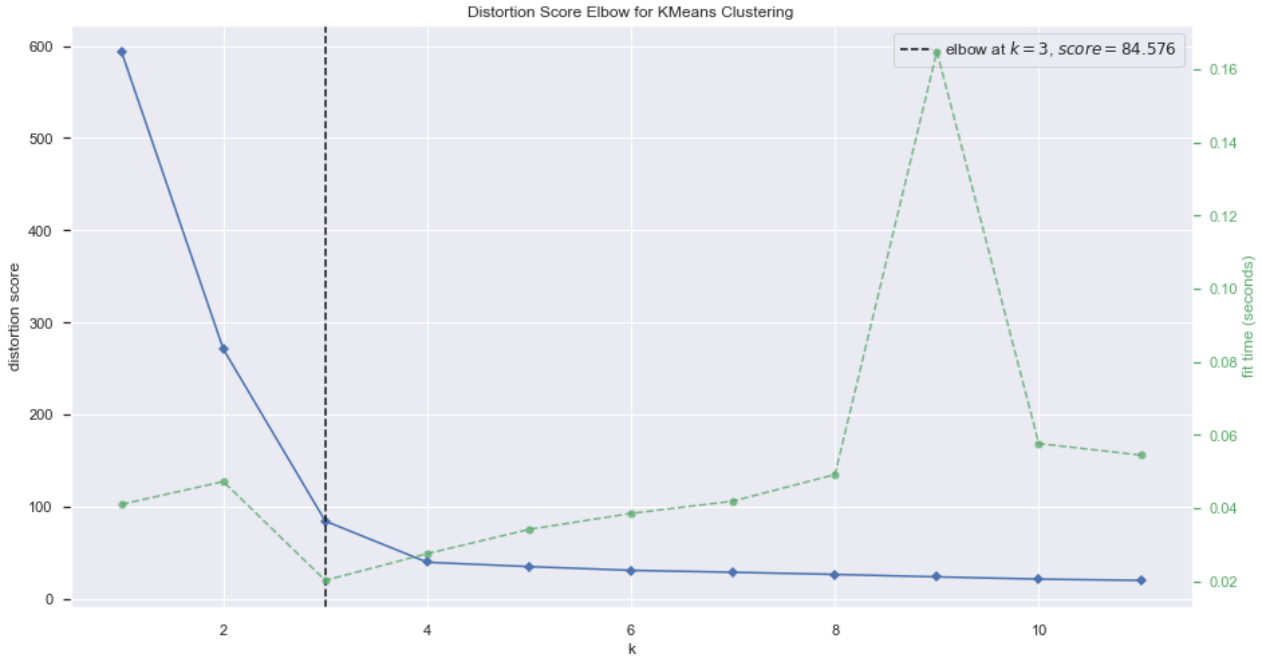
Out[198… <AxesSubplot:xlabel='prob_drink', ylabel='price'>



## KMEANS

Volvemos a realizar un K-means, donde tambien obtenemos que el numero de clusters es 3 gracias al grafico de codo.

In [199…
```
model_campaigns = KMeans()
visualizer_campaigns = KElbowVisualizer(model, k=(1,12)).fit(campaigns_nume
visualizer_campaigns.show()
```




Out[199…
```
<AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'
}, xlabel='k', ylabel='distortion score'>
```

In [200…
```
kmeans_campaigns = KMeans(n_clusters=3).fit(campaigns_numerico_normalizado)
```

In [201…
```
centroids_campaigns = kmeans_campaigns.cluster_centers_
print(centroids_campaigns)
```

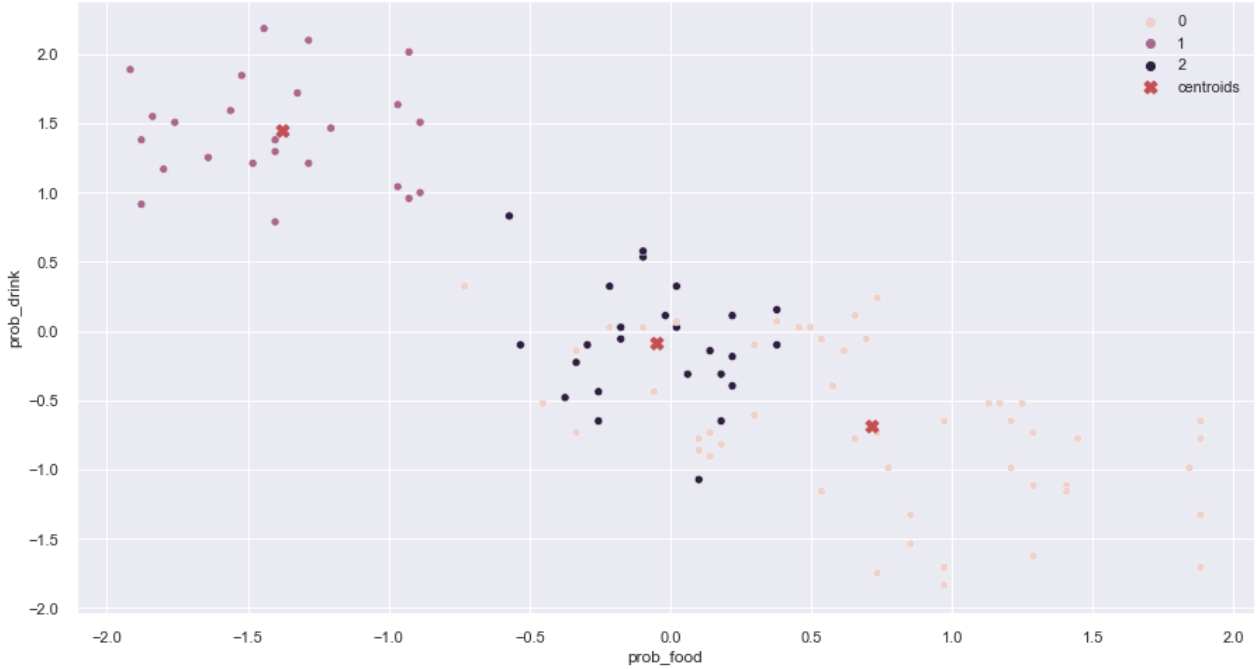
```
[[ 7.14727100e-01 -6.79219969e-01  9.48249901e-01 -4.02779915e-01
  -6.52762754e-01 -2.31111593e-33]
 [-1.37879802e+00  1.44556280e+00 -1.06848908e+00  1.58178807e+00
  -2.72616471e-01 -1.40712473e+00]
 [-5.06561843e-02 -8.71228603e-02 -8.28010720e-01 -7.76228241e-01
   1.57814198e+00  1.40712473e+00]]
```

Creamos de nuevo una variable donde almacenar las labels, para su uso mas adelante.

In [202…
```
labels_campaigns=kmeans_campaigns.labels_
```

In [204…

```
sns.scatterplot(data=campaigns_numerico_normalizado, x="prob_food", y="prob
plt.scatter(kmeans_campaigns.cluster_centers_[:,0], kmeans_campaigns.cluste
            marker="X", c="r", s=80, label="centroids")
plt.legend()
plt.show()
```




Realizamos un grafico mostrando los datos segun las campañas. En este grafico analizamos la campaña de bebidas que es el cluster 2, y la de comida el 0 y ambas se agrupan en el 0.

### 4.- Haciendo uso del dataset de entrenamiento de campaigns_to_customers_launched.csv, calcula el valor esperado de éxito para clúster de campañas mostrado a cada clúster de clientes. Este valor esperado, podría representar, de forma aproximada, por cada combinación de clusters campaña-cliente, la probabilidad de éxito al ofrecer una campaña al azar de un clúster de campañas a un cliente al azar de un clúster de clientes.

Para este ejercicio, creamos unos nuevos dataframes desde los originales. Creamos como nuevas columnas las labels hayadas anteriormente en K-means. Dejamos solo las colunmas que nos interesan que van a ser el id para poder juntarlas luego con la tabla de campaigns to customers...

In [205…

```
customers_ej4=customers
campaigns_ej4=campaigns
```

In [206…

```
customers_ej4=customers_ej4.drop(outliers.index)
```

In [207…

```
customers_ej4['labels_customers']=labels_customers

campaigns_ej4["labels_campaigns"]=labels_campaigns
```

In [208…
```
customers_ej4= customers_ej4.drop(columns=['Year_Birth','Education',
                                           'Marital_Status','Income','Kidhc
                                          'Recency', 'MntWines', 'MntFruits
                                           'MntFishProducts', 'MntSweetProc
                                           'NumDealsPurchases', 'NumWebPurc
                                          'NumStorePurchases', 'NumWebVisit
                                           'Z_Revenue', 'Response', 'Days']
```

In [209…
```
campaigns_ej4=campaigns_ej4.drop(columns=['prob_food','prob_drink','outdate
```

El objetivo de este ejercicio es obtener una tabla que para cluster de cliente nos indique como de exitosa ha sido la campaña, estas campañas estando igualmente agrupada rn clusters. Para ello utilizaremos las funciones de merge, juntando las colunmas de id que coinciden con con las de clientes y campañas en customers to campaigns.

In [210…
```
a=train.merge(customers_ej4,  left_on='customer', right_on='customer_id')
b=a.merge(campaigns_ej4,left_on='campaign', right_on='campaign_id' )
```

Queremos medir la probabilidad de exito, para ello creamos una columna que sean todo unos, que os servira como denominador a la hora de calcular la probabilidad. Por ejemplo si para campañas tenemos 4 filas pero solo 1 exitosa entonces solo 25% sera axitoso.

In [211…
```
b["uno"]=1
```

In [212…
```
b
```

Out[212…

| | Unnamed: 0 | customer | campaign | result | customer_id | labels_customers | campaign_id |
|---|---|---|---|---|---|---|---|
| **0** | 12130 | 170 | 65 | 0 | 170 | 1 | 65 |
| **1** | 3099 | 2167 | 65 | 1 | 2167 | 0 | 65 |
| **2** | 3769 | 2212 | 65 | 0 | 2212 | 1 | 65 |
| **3** | 4303 | 1635 | 65 | 0 | 1635 | 2 | 65 |
| **4** | 9543 | 730 | 65 | 0 | 730 | 1 | 65 |
| **...** | ... | ... | ... | ... | ... | ... | .. |
| **8005** | 9968 | 429 | 26 | 0 | 429 | 0 | 26 |
| **8006** | 8356 | 1847 | 26 | 0 | 1847 | 1 | 26 |
| **8007** | 4112 | 1748 | 26 | 0 | 1748 | 2 | 26 |
| **8008** | 9224 | 2069 | 26 | 0 | 2069 | 1 | 26 |
| **8009** | 12232 | 1153 | 26 | 0 | 1153 | 0 | 26 |

8010 rows × 9 columns

In [ ]:

```
```

In [217…

```
exito_campañas_train=b.groupby(["labels_customers","labels_campaigns"]).sur
```

In [218…

```
exito_campañas_train["exito"]=(exito_campañas_train.result/exito_campañas_t
```

In [219…

```
exito_campañas_train
```

Out[219…

| | labels_customers | labels_campaigns | Unnamed: 0 | customer | campaign | result | customer_i |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 0 | 6941281 | 1183208 | 58639 | 286 | 118320 |
| **1** | 0 | 1 | 3682700 | 617001 | 24091 | 96 | 61700 |
| **2** | 0 | 2 | 3211505 | 546129 | 20976 | 105 | 54612 |
| **3** | 1 | 0 | 12042237 | 2038581 | 100262 | 555 | 203858 |
| **4** | 1 | 1 | 6115579 | 1048570 | 40262 | 149 | 104857 |
| **5** | 1 | 2 | 6121949 | 1062061 | 39079 | 228 | 106206 |
| **6** | 2 | 0 | 7744114 | 1191081 | 63684 | 352 | 119108 |
| **7** | 2 | 1 | 4127316 | 653874 | 27677 | 78 | 65387 |
| **8** | 2 | 2 | 3745926 | 602830 | 24058 | 161 | 60283 |

In [220…
```
exito_campañas_train=exito_campañas_train.drop(columns=['Unnamed: 0','custo
```

In [221…
```
exito_campañas_train
```

Out[221…

| | labels_customers | labels_campaigns | exito |
|---|---|---|---|
| **0** | 0 | 0 | 27.526468 |
| **1** | 0 | 1 | 17.777778 |
| **2** | 0 | 2 | 22.198732 |
| **3** | 1 | 0 | 30.662983 |
| **4** | 1 | 1 | 16.301969 |
| **5** | 1 | 2 | 24.463519 |
| **6** | 2 | 0 | 31.205674 |
| **7** | 2 | 1 | 12.703583 |
| **8** | 2 | 2 | 28.750000 |

Obtenemos nuestra tabla, en la primera columna tenemos los cluesters de customers que estan agrupados primero los 0, luego los 1 y por ultimo los 2. Despues tenemos campains donde estan los clusters que estan dentro de customers y sus probabilidades de exito. Esta ultima es el calculo de agrupar clientes y campañas y dividirlo por el total de las campañas.

In [229…
```
#histo= exito_campañas_train.groupby("labels_customers")
```

Por ultimo en este ejercicio hemos creado una tabla para observar la distribucion de los clientes en los clusters y otra para ver la distribucion de las campañas en los clusters.

In [230…
```
z=customers_ej4.groupby("labels_customers").count()
z.head()
```

Out[230…

| | customer_id |
|---|---|
| **labels_customers** | |
| **0** | 581 |
| **1** | 1006 |
| **2** | 637 |

In [233…
```
zz=campaigns_ej4.groupby("labels_campaigns").count()
zz.head()
```

Out[233…

| labels_campaigns | campaign_id |
|---|---|
| **0** | 50 |
| **1** | 25 |
| **2** | 25 |

5.- Evalúa como de preciso son los modelos que hemos creado, contrastándolos contra el set de test. Para ello, simplemente tienes que calcular el error cuadrático medio, restando y elevando al cuadrado (cómo haciamos para detectar outliers con PCA), el resultado de cada campaña que aparece en test menos la probabilidad de esa campaña teniendo en cuenta el clúster al que pertenece el cliente y el clúster al que pertenece la campaña.

Para realizar este ejercicio repetimos las operaciones realizadas en el ejercicio anterior, para asi poder calcular el error y ver la precision.

In [234…
```
c=test.merge(customers_ej4,  left_on='customer', right_on='customer_id')
```

In [235…
```
d=c.merge(campaigns_ej4,left_on='campaign', right_on='campaign_id' )
```

grouped_df3 = d.groupby(['labels_campaigns', 'labels_customers']).mean()

In [236…
```
d["uno"]=1
```

In [237…
```
d
```

Out[237…

| | Unnamed: 0 | customer | campaign | result | customer_id | labels_customers | campaign_id |
|---|---|---|---|---|---|---|---|
| **0** | 12361 | 1167 | 45 | 0 | 1167 | 1 | 45 |
| **1** | 640 | 230 | 45 | 1 | 230 | 2 | 45 |
| **2** | 4558 | 816 | 45 | 0 | 816 | 1 | 45 |
| **3** | 4852 | 64 | 45 | 0 | 64 | 0 | 45 |
| **4** | 4078 | 1065 | 45 | 0 | 1065 | 1 | 45 |
| **...** | ... | ... | ... | ... | ... | ... | .. |
| **2667** | 5316 | 2115 | 41 | 0 | 2115 | 2 | 41 |
| **2668** | 8324 | 1421 | 41 | 0 | 1421 | 0 | 41 |
| **2669** | 10245 | 904 | 41 | 0 | 904 | 2 | 41 |
| **2670** | 9728 | 701 | 41 | 0 | 701 | 0 | 41 |
| **2671** | 2380 | 1541 | 41 | 0 | 1541 | 2 | 41 |

2672 rows × 9 columns

In [242…
```
exito_campaigns_test=d.groupby(["labels_customers","labels_campaigns"]).sur
```

In [243…
```
exito_campaigns_test["exito"]=(exito_campaigns_test.result/exito_campaigns_
```

In [244…
```
exito_campaigns_test=exito_campaigns_test.drop(columns=['Unnamed: 0','custo
```

In [245…
```
exito_campaigns_test.head()
```

Out[245…

| | labels_customers | labels_campaigns | exito |
|---|---|---|---|
| **0** | 0 | 0 | 24.924012 |
| **1** | 0 | 1 | 18.497110 |
| **2** | 0 | 2 | 14.792899 |
| **3** | 1 | 0 | 29.139073 |
| **4** | 1 | 1 | 17.799353 |

En esta nueva tabla obtenemos la probabilidad de exito por clusters de campañas y clusters de clientes. Observamos que tenemos menos clusters, nos faltan 3 filas. Creemos que esto es debido a la nuestra del test que es el 20%.

MSE

Calculamos el Mean square error, para ello indicamos que dos dataframes queremos y sus respectivas columnas. Obtenemos un error de 13.548872611458371.

In [247…
```
from sklearn.metrics import mean_squared_error
```

In [248…
```
error=mean_squared_error(exito_campañas_train["exito"],exito_campaigns_test
```

In [249…
```
error
```

Out[249… 13.548872611458371

In [119…
```
#mse_models=((exito_train["exito"] - exito_test["exito"])**2)*100
```

6.- Genera datos simulados:

6.1 Genera un cliente simulado. ¿De que clúster de campañas le ofrecerías al azar una campaña con el objetivo de maximizar el éxito?.
6.2 Genera una campaña nueva simulada. ¿A que clúster de clientes le ofrecerías la campaña con el objetivo de maximizar el éxito?.

Nota: seleccionar aleatoriamente el centroide de un kmedias es una simulación.

In [ ]:
```
Sacamos un cliente al azar utilizando la funcion random. Obtenemos el numer
```

In [250…
```
import random
np.random.seed(1)
```

In [253…
```
customers_ej4.sample(random_state=1)
```

Out[253…

| | customer_id | labels_customers |
|---|---|---|
| **1672** | 1672 | 0 |

Para el cliente 1672 se encuentra en el cluster 0 de clientes, por lo que le recomendamos la campaña 0, ya que es aquella que tiene el mayor exito con 27.5%.

In [254…
```
campaigns_ej4.sample(random_state=1)
```

Out[254...

| | campaign_id | labels_campaigns |
|---|---|---|
| **80** | 80 | 0 |

Para la campaña 80 se encuentra en el cluster 0 de campañas, por lo que le recomendamos el cluster 2, ya que es aquella que tiene el mayor exito con 31.2%.

**7.- BONUS. PUNTO EXTRA: Realiza una optimización de tus modelos. Haciendo uso de los conocimientos de diferentes técnicas de clustering que conoces, y sus diferentes parámetros. Prueba diferentes combinaciones (todas las que quieras) para clusterizar tanto las campañas y los clientes y evalúalas contra el dataset de validación. Quédate con aquella que tenga el error cuadrático más pequeño. Una vez elegida la mejor combinación, evalúala contra el dataset de test. TIPS: puedes incluso combinar el hacer un PCA primero antes de aplicar clustering.**

# CONCLUSION

Gracias a este trabajo hemos podido empezar a sumergirnos en el mundo de machine learing. Machine Learning pertenece al campo de la Inteligencia Artificial, utilizando algoritmos para identificar patrones en datos y realizar un análisis predictivo. En este caso estamos utilizando la rama de Machine Learning de aprendizaje no supervisado. En esta rama los algoritmos no tienen conocimiento previo por lo que hay que crear algoritmos para organizar los datos.

Hemos podido ver un ejemplo aplicado a la vida real que ha sido muy util y mas hoy en dia cuando la tecnologia tiene un rol vital para que las empresas evolucionen. Ha sido muy interesante ver como podemos ayuda a las empresas a analizar a sus clientes y ofrecerles las mejores ofertas adaptandose a cada uno individualmente. Sobretodo a las empresa de venta online que cuentan con grandes bases de datos de sus clientes y pueden utilizar data science para optimizar sus campañas.

En este caso hemos utilizado aprendizaje no supervisado por lo que hemos echo un analisis exploratorio en el que el objetivo era utilizar la estructura de los datos. Antes de realizar los algoritmos de clustering hemos realizado una presentacion de los datosque ha consistido en encontar NA's, ver las correlaciones en los datos, representarlos graficamente y modificarlos para poder usar codigos de clusterng correctamente (normalizar)

Despues hemos aplicado algoritmos de Machine Learning como t-sne, K-means , DBScan y PCA . PCa y db Estos ultimos han sido de gran ayuda a la hora de identificar los outliers. K-means ha sido imprescindible a la hora de crear los clusters de clientes con las campañas, que hemos utilizado posteriormente para la evalucion de dichas campañas. T-sne ha sido util para reducir los datos y poderlos representar mas claramente.

Despues de tratar los datos hemos obtenido un MSE alto por seleccionar muestras pequeña, no filtrar bien los outliers, no utilizar la muestra de validacion o errores en la probabilidad de exito.

En global esta practicva nos ha permitico aplicar todo lo aprendido durante el curso y a entender la aplicacion real de Data Science.

# Referencias

https://programmerclick.com/article/96761372751/ https://aprendeia.com/dbscan-teoria/ https://www.delftstack.com/howto/python-pandas/python-pandas-df-size-df-shape-and-df-ndim/ https://es.acervolima.com/cluster-dbscan-en-ml-agrupacion-basada-en-densidad/

In [ ]: