

✓ Examen de cinturón AML – Opción A

Alumno: Luciano Benjamin Recalde Carballo

✓ 1. Exploración y preprocesamiento de datos:


Cargamos los datos

Observando que los delimitadores están conformados por "" hemos especificado el parametro "delimiter"

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
ds_path = "data/marketing_campaign.csv"
df = pd.read_csv(ds_path, delimiter="\t")
```

Visualizamos la cabecera

```
df.head(10)
```




	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumWebVisitsMoni
0	5524	1957	Graduation	Single	58138.0	0	0	04-09-2012	58	635	...	
1	2174	1954	Graduation	Single	46344.0	1	1	08-03-2014	38	11	...	
2	4141	1965	Graduation	Together	71613.0	0	0	21-08-2013	26	426	...	
3	6182	1984	Graduation	Together	26646.0	1	0	10-02-2014	26	11	...	
4	5324	1981	PhD	Married	58293.0	1	0	19-01-2014	94	173	...	
5	7446	1967	Master	Together	62513.0	0	1	09-09-2013	16	520	...	
6	965	1971	Graduation	Divorced	55635.0	0	1	13-11-2012	34	235	...	
7	6177	1985	PhD	Married	33454.0	1	0	08-05-2013	32	76	...	
8	4855	1974	PhD	Together	30351.0	1	0	06-06-2013	19	14	...	
9	5899	1950	PhD	Together	5648.0	1	1	13-03-2014	68	28	...	:

10 rows × 29 columns

Exploramos los datos

```
df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
#   Column                Non-Null Count  Dtype
---  -
0   ID                    2240 non-null  int64
1   Year_Birth            2240 non-null  int64
2   Education             2240 non-null  object
3   Marital_Status        2240 non-null  object
4   Income                2216 non-null  float64
5   Kidhome               2240 non-null  int64
6   Teenhome              2240 non-null  int64
7   Dt_Customer           2240 non-null  object
8   Recency               2240 non-null  int64
9   MntWines              2240 non-null  int64
10  MntFruits             2240 non-null  int64
11  MntMeatProducts       2240 non-null  int64
12  MntFishProducts       2240 non-null  int64
13  MntSweetProducts      2240 non-null  int64
14  MntGoldProds          2240 non-null  int64
15  NumDealsPurchases     2240 non-null  int64
```

```

16 NumWebPurchases      2240 non-null int64
17 NumCatalogPurchases  2240 non-null int64
18 NumStorePurchases     2240 non-null int64
19 NumWebVisitsMonth     2240 non-null int64
20 AcceptedCmp3          2240 non-null int64
21 AcceptedCmp4          2240 non-null int64
22 AcceptedCmp5          2240 non-null int64
23 AcceptedCmp1          2240 non-null int64
24 AcceptedCmp2          2240 non-null int64
25 Complain              2240 non-null int64
26 Z_CostContact         2240 non-null int64
27 Z_Revenue             2240 non-null int64
28 Response              2240 non-null int64
dtypes: float64(1), int64(25), object(3)
memory usage: 507.6+ KB

```

Se observan tipos de datos que pueden ser ajustados.

A continuacion, convertimos Dt_CUstomer a tipo de dato Datetime

```

from datetime import datetime

def _convert_date(x):
    if pd.isna(x) or pd.isnull(x):
        return x
    try:
        return datetime.strptime(x, "%d-%m-%Y")
    except Exception:
        print(f"Fallo para valor: {x}")
        return x

df["Dt_Customer"] = pd.to_datetime(df["Dt_Customer"].apply(_convert_date), utc=True)
df.info()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2240 entries, 0 to 2239
Data columns (total 29 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   ID                    2240 non-null  int64
 1   Year_Birth            2240 non-null  int64
 2   Education             2240 non-null  object
 3   Marital_Status       2240 non-null  object
 4   Income               2216 non-null  float64
 5   Kidhome              2240 non-null  int64
 6   Teenhome             2240 non-null  int64
 7   Dt_Customer          2240 non-null  datetime64[ns, UTC]
 8   Recency              2240 non-null  int64
 9   MntWines             2240 non-null  int64
10   MntFruits            2240 non-null  int64
11   MntMeatProducts      2240 non-null  int64
12   MntFishProducts      2240 non-null  int64
13   MntSweetProducts     2240 non-null  int64
14   MntGoldProds         2240 non-null  int64
15   NumDealsPurchases    2240 non-null  int64
16   NumWebPurchases      2240 non-null  int64
17   NumCatalogPurchases  2240 non-null  int64
18   NumStorePurchases    2240 non-null  int64
19   NumWebVisitsMonth    2240 non-null  int64
20   AcceptedCmp3         2240 non-null  int64
21   AcceptedCmp4         2240 non-null  int64
22   AcceptedCmp5         2240 non-null  int64
23   AcceptedCmp1         2240 non-null  int64
24   AcceptedCmp2         2240 non-null  int64
25   Complain             2240 non-null  int64
26   Z_CostContact        2240 non-null  int64
27   Z_Revenue            2240 non-null  int64
28   Response             2240 non-null  int64
dtypes: datetime64[ns, UTC](1), float64(1), int64(25), object(2)
memory usage: 507.6+ KB

```

Observamos los valores unicos de las variables categóricas

```

from utils.eda import get_categoric_columns
categoric_columns = get_categoric_columns(df)
for i in categoric_columns:
    print(i)
    print(df[i].unique())

```


```
Education
['Graduation' 'PhD' 'Master' 'Basic' '2n Cycle']
Marital_Status
['Single' 'Together' 'Married' 'Divorced' 'Widow' 'Alone' 'Absurd' 'YOLO']
```

Vemos valores atipicos como Absurd y Yolo, mas adelante nos encargaremos de esos valores

Continuamos con la exploración de los datos

Visualizamos las últimas entradas

```
df.tail(10)
```



	ID	Year_Birth	Education	Marital_Status	Income	Kidhome	Teenhome	Dt_Customer	Recency	MntWines	...	NumWebVisi
2230	7004	1984	Graduation	Single	11012.0	1	0	2013-03-16 00:00:00+00:00	82	24	...	
2231	9817	1970	Master	Single	44802.0	0	0	2012-08-21 00:00:00+00:00	71	853	...	
2232	8080	1986	Graduation	Single	26816.0	0	0	2012-08-17 00:00:00+00:00	50	5	...	
2233	9432	1977	Graduation	Together	666666.0	1	0	2013-06-02 00:00:00+00:00	23	9	...	
2234	8372	1974	Graduation	Married	34421.0	1	0	2013-07-01 00:00:00+00:00	81	3	...	
2235	10870	1967	Graduation	Married	61223.0	0	1	2013-06-13 00:00:00+00:00	46	709	...	
2236	4001	1946	PhD	Together	64014.0	2	1	2014-06-10 00:00:00+00:00	56	406	...	
2237	7270	1981	Graduation	Divorced	56981.0	0	0	2014-01-25 00:00:00+00:00	91	908	...	
2238	8235	1956	Master	Together	69245.0	0	1	2014-01-24 00:00:00+00:00	8	428	...	
2239	9405	1954	PhD	Married	52869.0	1	1	2012-10-15 00:00:00+00:00	40	84	...	

10 rows × 29 columns

Verificamos los valores descriptivos

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
ID	2240.0	5592.159821	3246.662198	0.0	2828.25	5458.5	8427.75	11191.0
Year_Birth	2240.0	1968.805804	11.984069	1893.0	1959.00	1970.0	1977.00	1996.0
Income	2216.0	52247.251354	25173.076661	1730.0	35303.00	51381.5	68522.00	666666.0
Kidhome	2240.0	0.444196	0.538398	0.0	0.00	0.0	1.00	2.0
Teenhome	2240.0	0.506250	0.544538	0.0	0.00	0.0	1.00	2.0
Recency	2240.0	49.109375	28.962453	0.0	24.00	49.0	74.00	99.0
MntWines	2240.0	303.935714	336.597393	0.0	23.75	173.5	504.25	1493.0
MntFruits	2240.0	26.302232	39.773434	0.0	1.00	8.0	33.00	199.0
MntMeatProducts	2240.0	166.950000	225.715373	0.0	16.00	67.0	232.00	1725.0
MntFishProducts	2240.0	37.525446	54.628979	0.0	3.00	12.0	50.00	259.0
MntSweetProducts	2240.0	27.062946	41.280498	0.0	1.00	8.0	33.00	263.0
MntGoldProds	2240.0	44.021875	52.167439	0.0	9.00	24.0	56.00	362.0
NumDealsPurchases	2240.0	2.325000	1.932238	0.0	1.00	2.0	3.00	15.0
NumWebPurchases	2240.0	4.084821	2.778714	0.0	2.00	4.0	6.00	27.0
NumCatalogPurchases	2240.0	2.662054	2.923101	0.0	0.00	2.0	4.00	28.0
NumStorePurchases	2240.0	5.790179	3.250958	0.0	3.00	5.0	8.00	13.0
NumWebVisitsMonth	2240.0	5.316518	2.426645	0.0	3.00	6.0	7.00	20.0
AcceptedCmp3	2240.0	0.072768	0.259813	0.0	0.00	0.0	0.00	1.0
AcceptedCmp4	2240.0	0.074554	0.262728	0.0	0.00	0.0	0.00	1.0
AcceptedCmp5	2240.0	0.072768	0.259813	0.0	0.00	0.0	0.00	1.0
AcceptedCmp1	2240.0	0.064286	0.245316	0.0	0.00	0.0	0.00	1.0
AcceptedCmp2	2240.0	0.013393	0.114976	0.0	0.00	0.0	0.00	1.0
Complain	2240.0	0.009375	0.096391	0.0	0.00	0.0	0.00	1.0
Z_CostContact	2240.0	3.000000	0.000000	3.0	3.00	3.0	3.00	3.0
Z_Revenue	2240.0	11.000000	0.000000	11.0	11.00	11.0	11.00	11.0
Response	2240.0	0.149107	0.356274	0.0	0.00	0.0	0.00	1.0

Se observa que para las columnas Z_CostContact y Z_Revenue la desviacion estandar es 0, por lo tanto deben tratarse de valores constantes para todos los registros del dataset. Estas columnas pueden descartarse ya que no aportaran informacion util al modelo.

```
df.drop(['Z_Revenue', 'Z_CostContact'], axis=1, inplace=True)
```

Continuamos con el analisis

```
df.describe(include='object').T
```

	count	unique	top	freq
Education	2240	5	Graduation	1127
Marital_Status	2240	8	Married	864

Ahora reduciremos el numero de opciones para las columnas categoricas a fin de tener una mejor respuesta del modelo y simplificar la entrada.

```
from utils.eda import get_categoric_columns
categoric_columns = get_categoric_columns(df)
for i in categoric_columns:
    print(i)
    print(df[i].unique())

df['Marital_Status'].value_counts()
```

```

↗ Education
['Graduation' 'PhD' 'Master' 'Basic' '2n Cycle']
Marital_Status
['Single' 'Together' 'Married' 'Divorced' 'Widow' 'Alone' 'Absurd' 'YOLO']

```

count

Marital_Status	
Married	864
Together	580
Single	480
Divorced	232
Widow	77
Alone	3
Absurd	2
YOLO	2

dtype: int64

- En Education, reduciremos a las categorías: Pre-universitaria (undergraduate), Graduation, Master, PhD
- En Marital_Status simplificaremos a: Soltero, Acompanhado, Divorciado, Casado y Viudo (Eliminaremos las entradas YOLO y Absurd por tratarse de datos erroneos y poco recurrentes)

```

df.loc[
    (df['Education'] == 'Basic') | (df['Education'] == '2n Cycle'),
    'Education'
] = 'Undergraduate'
drop_entries = df[(df['Marital_Status'] == 'Absurd') | (df['Marital_Status'] == 'YOLO')].index
df.drop(index=drop_entries, inplace=True)
df.loc[df.Marital_Status == 'Alone', 'Marital_Status'] = "Single"
df.describe(include='object').T

```

```

↗

```

	count	unique	top	freq
Education	2236	4	Graduation	1126
Marital_Status	2236	5	Married	864

Ahora las columnas categoricas son mas claras y el modelo podra ser generado con mejor respuesta

✓ Identificación de valores nulos, duplicados y outliers

```

duplicated = df.duplicated().sum()
print(f"Registros duplicados en df: {duplicated}")

```

```

↗ Registros duplicados en df: 0

```

No se observan valores duplicados

Ahora observaremos la columna ID

```
df['ID'].duplicated().sum()
```

```

↗ 0

```

No se observan duplicados por lo tanto cada ID posee solo una entrada, por lo tanto podemos eliminar dicha columna ya que no aporta informacion importante al modelo

```
df.drop(['ID'],axis=1,inplace=True)
```

Volvemos a verificar si existen valores duplicados ahora que eliminamos la columna ID. Ya que pueden existir entradas de diferentes clientes que sean identicas.

```
duplicated = df.duplicated().sum()
print(f'Registros duplicados en df: {duplicated}')
```

↗ Registros duplicados en df: 183

Observamos que existen registros duplicados, por lo tanto eliminamos estas entradas que podrian ocasionar overfitting en nuestro modelo.

```
df = df.drop_duplicates()
duplicated = df.duplicated().sum()
print(f'Registros duplicados en df: {duplicated}')
```

↗ Registros duplicados en df: 0

Ahora buscamos valores nulos


```
from utils.eda import get_nulll_data_info
get_nulll_data_info(df)
```

↗



	datos sin NAs	en q	Na en q	Na en %	
Income	2029	24	1.17		il.
Year_Birth	2053	0	0.00		
NumDealsPurchases	2053	0	0.00		
Complain	2053	0	0.00		
AcceptedCmp2	2053	0	0.00		
AcceptedCmp1	2053	0	0.00		
AcceptedCmp5	2053	0	0.00		
AcceptedCmp4	2053	0	0.00		
AcceptedCmp3	2053	0	0.00		
NumWebVisitsMonth	2053	0	0.00		
NumStorePurchases	2053	0	0.00		
NumCatalogPurchases	2053	0	0.00		
NumWebPurchases	2053	0	0.00		
MntGoldProds	2053	0	0.00		
Education	2053	0	0.00		
MntSweetProducts	2053	0	0.00		
MntFishProducts	2053	0	0.00		
MntMeatProducts	2053	0	0.00		
MntFruits	2053	0	0.00		
MntWines	2053	0	0.00		
Recency	2053	0	0.00		
Dt_Customer	2053	0	0.00		
Teenhome	2053	0	0.00		
Kidhome	2053	0	0.00		
Marital_Status	2053	0	0.00		
Response	2053	0	0.00		

Se observan 24 valores nulos. Procederemos a imputarlos por el valor promedio. Ya que se trata del ingreso del cliente y el valor promedio es un metodo adecuado.

```
df.loc[:, 'Income'] = df['Income'].fillna(df['Income'].mean())
get_nulll_data_info(df)
```



	datos sin NAs en q	Na en q	Na en %
Year_Birth	2053	0	0.0
Education	2053	0	0.0
Complain	2053	0	0.0
AcceptedCmp2	2053	0	0.0
AcceptedCmp1	2053	0	0.0
AcceptedCmp5	2053	0	0.0
AcceptedCmp4	2053	0	0.0
AcceptedCmp3	2053	0	0.0
NumWebVisitsMonth	2053	0	0.0
NumStorePurchases	2053	0	0.0
NumCatalogPurchases	2053	0	0.0
NumWebPurchases	2053	0	0.0
NumDealsPurchases	2053	0	0.0
MntGoldProds	2053	0	0.0
MntSweetProducts	2053	0	0.0
MntFishProducts	2053	0	0.0
MntMeatProducts	2053	0	0.0
MntFruits	2053	0	0.0
MntWines	2053	0	0.0
Recency	2053	0	0.0
Dt_Customer	2053	0	0.0
Teenhome	2053	0	0.0
Kidhome	2053	0	0.0
Income	2053	0	0.0
Marital_Status	2053	0	0.0
Response	2053	0	0.0

Ahora crearemos una nueva columna para contar la cantidad de hijos de cada cliente

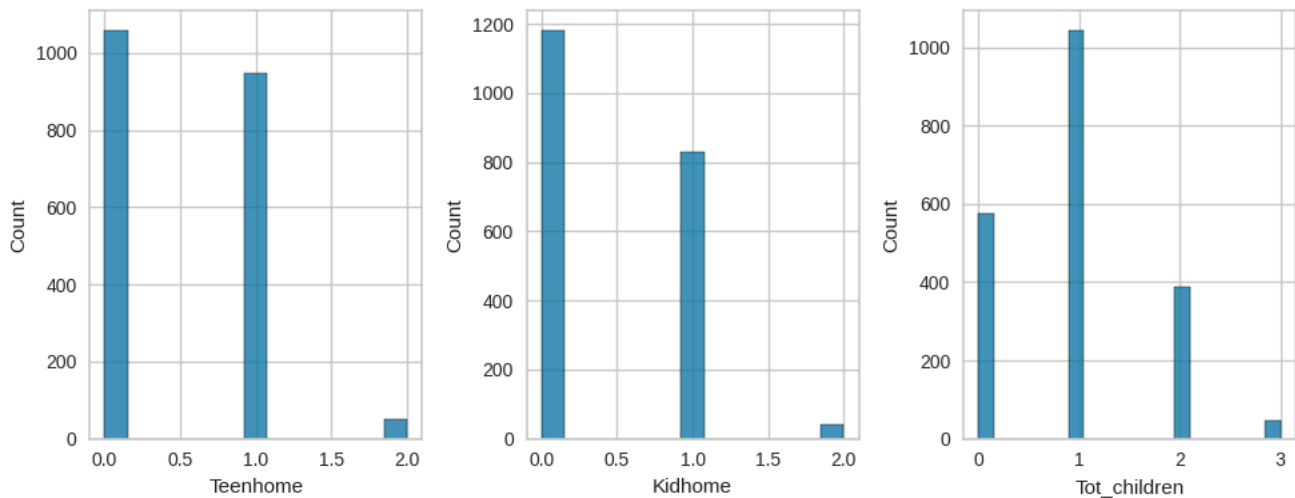
```
df.loc[:, 'Tot_children'] = df['Teenhome'] + df['Kidhome']
```

```
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(10,4))
```

```
sns.histplot(df['Teenhome'],ax=axes[0])
sns.histplot(df['Kidhome'],ax=axes[1])
sns.histplot(df['Tot_children'],ax=axes[2])
plt.tight_layout()
plt.show()
```

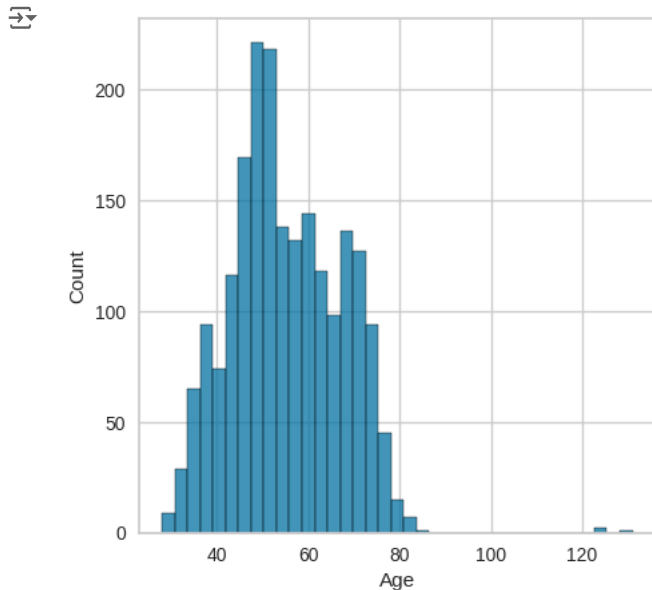
```
<ipython-input-1569-1a8f9be8041c>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view
`df.loc[:, 'Tot_children'] = df['Teenhome'] + df['Kidhome']`



Observaremos el rango etario de los clientes

```
df['Age'] = 2024-df['Year_Birth']
plt.figure(figsize=(5,5))
sns.histplot(df['Age'])
plt.show()
```



Observamos outliers que indican edades mas alla de 120 anhos, lo cual es altamente probable de que se trate de un error, eliminamos dichas entradas

```
# removemos outliers
df['Age'] = df['Age'].clip(upper=81)
```

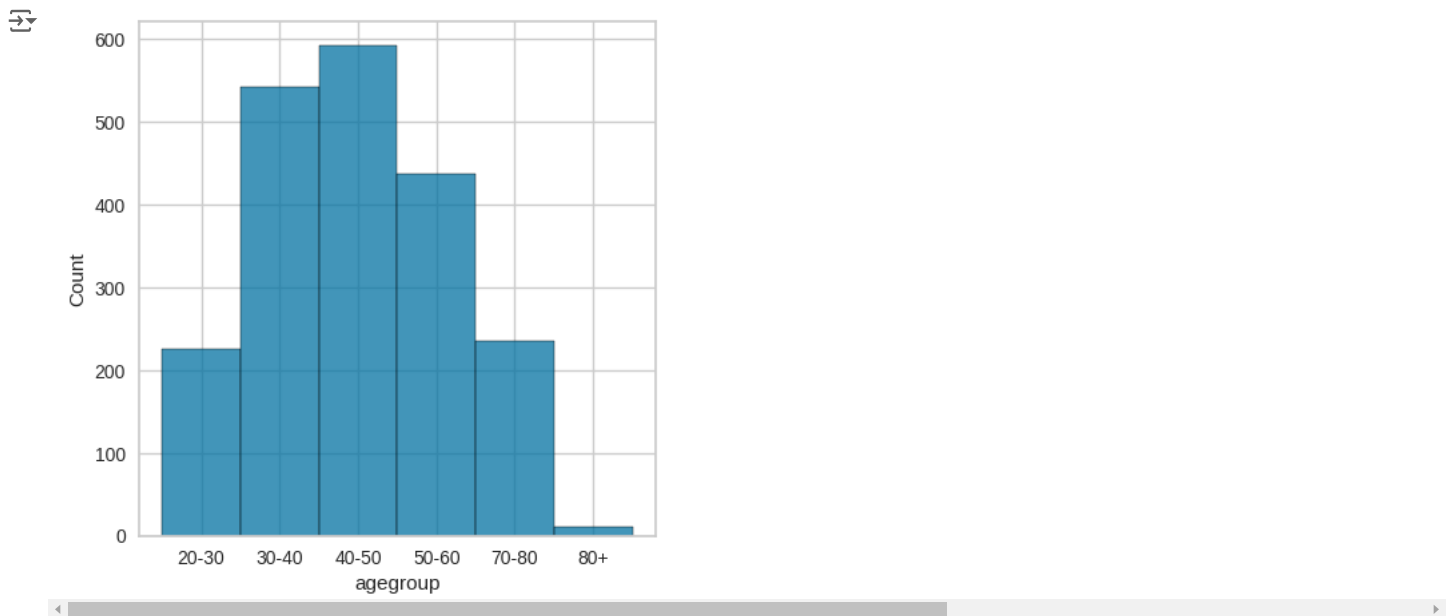
Agrupamos y volvemos a observar en un histograma

```
bins=[30,40,50,60,70,80,90]
labels = ['20-30','30-40','40-50','50-60','70-80','80+']

df['agegroup'] = pd.cut(df['Age'],bins=bins,labels=labels)
plt.figure(figsize=(5,5))
```



```
sns.histplot(df['agegroup'])
plt.show()
```



Creamos un feature con los datos totales de gastos

```
df['Tot_Expenses'] = df['MntWines'] + df['MntFruits'] + df['MntMeatProducts'] + df['MntFishProducts'] + df['MntSweetProducts'] +
```

Creamos un feature con los datos totales en cupones

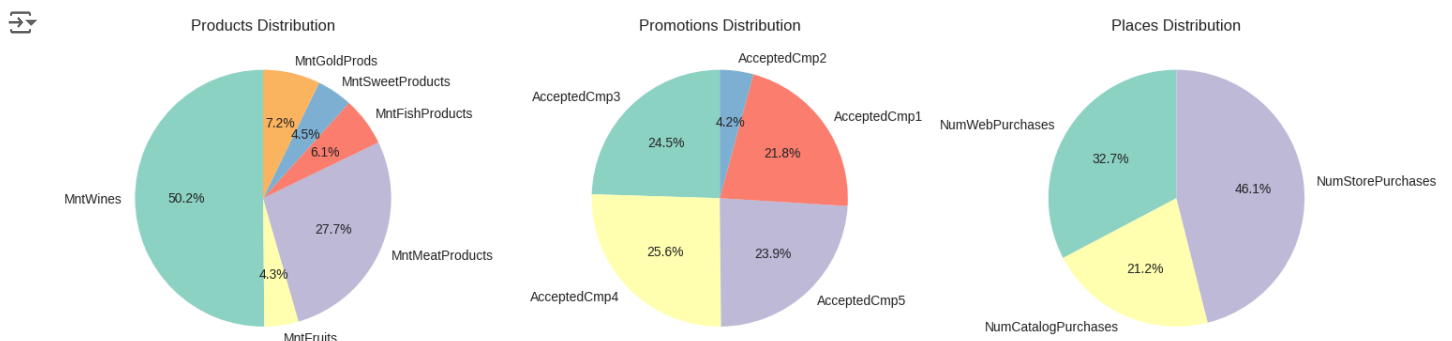
```
df['Tot_AcceptedCmp'] = df['AcceptedCmp1'] + df['AcceptedCmp2'] + df['AcceptedCmp3'] + df['AcceptedCmp4'] + df['AcceptedCmp5'] +
```

Y creamos otro feature con las compras totales por tipo de compra

```
df['Tot_Purchases'] = df['NumWebPurchases'] + df['NumCatalogPurchases'] + df['NumStorePurchases'] + df['NumDealsPurchases']
```

Generamos un grafico con la distribucion de proctos por categoria

```
from utils.eda import graph_pie
graph_pie(df)
```



Generamos Histogramas para visualizar la distribucion

```
from utils.eda import graph_histogram, get_numeric_columns
graph_histogram(df, get_numeric_columns(df), bins=15, columns number=4, figsize=(20, 20))
```





Histogramas



No se observa nada fuera de lo normal en los histogramas

Ahora verificamos los outliers

```
from utils.eda import get_outliers_data
get_outliers_data(df)
```

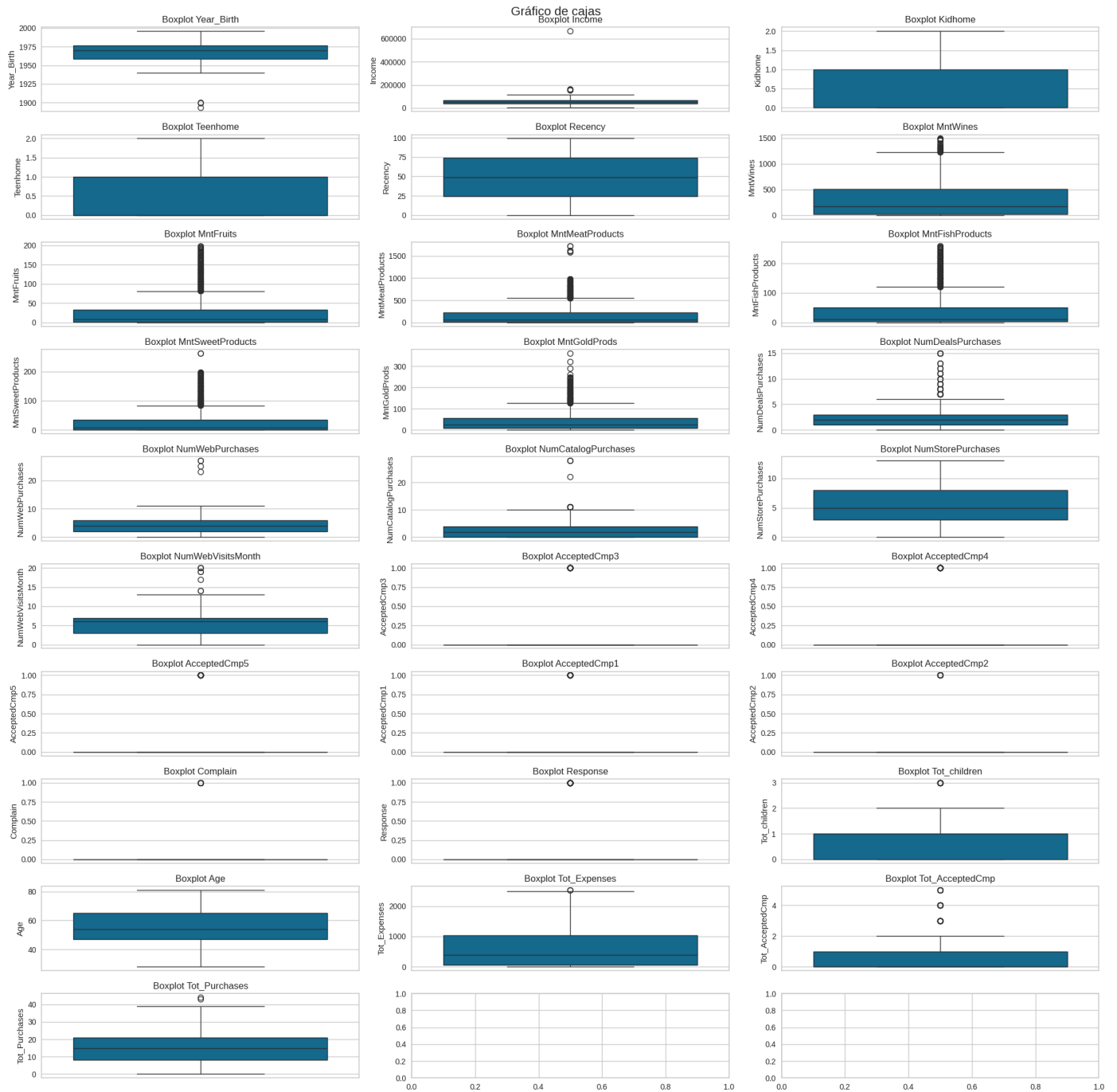


	Year_Birth	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	MntSweet
N° Outliers	3.000000	8.000000	0.0	0.0	0.0	33.000000	206.000000	173.000000	201.00000	2
% Outliers	0.146128	0.389674	0.0	0.0	0.0	1.607404	10.034096	8.426693	9.79055	
Lim. mix	1932.000000	-13158.500000	-1.5	-1.5	-51.0	-700.000000	-47.000000	-305.000000	-67.50000	.
Lim. max	2004.000000	117133.500000	2.5	2.5	149.0	1228.000000	81.000000	551.000000	120.50000	

4 rows × 28 columns

Visualizamos las gráficas de cajas de nuestras columnas


```
from utils.eda import graph_boxplot, get_numeric_columns
graph_boxplot(df, columns=get_numeric_columns(df), figsize=(20, 20))
```



Observamos algunos outliers

Ahora veremos las características descriptivas de las variables numéricas


```
df.describe()
```



	Year_Birth	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	Mn
count	2053.000000	2053.000000	2053.000000	2053.000000	2053.000000	2053.000000	2053.000000	2053.000000	2053.000000	
mean	1968.767657	52350.900444	0.446176	0.509498	49.010716	303.999513	26.209937	167.748173	37.207014	
std	11.965475	25396.446917	0.537712	0.546594	28.986804	336.849714	39.744889	228.409947	54.459371	
min	1893.000000	1730.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1959.000000	35701.000000	0.000000	0.000000	24.000000	23.000000	1.000000	16.000000	3.000000	
50%	1970.000000	52074.000000	0.000000	0.000000	49.000000	173.000000	8.000000	67.000000	12.000000	
75%	1977.000000	68274.000000	1.000000	1.000000	74.000000	505.000000	33.000000	230.000000	50.000000	
max	1996.000000	666666.000000	2.000000	2.000000	99.000000	1493.000000	199.000000	1725.000000	259.000000	

8 rows × 28 columns

```
from utils.eda import get_descriptive_statistics
get_descriptive_statistics(df)
```



	Year_Birth	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProduct
min	1893.000000	1730.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1996.000000	666666.000000	2.000000	2.000000	99.000000	1493.000000	199.000000	1725.000000	259.000000
mean	1968.767657	52350.900444	0.446176	0.509498	49.010716	303.999513	26.209937	167.748173	37.207014
std	11.965475	25396.446917	0.537712	0.546594	28.986804	336.849714	39.744889	228.409947	54.459371
median	1970.000000	52074.000000	0.000000	0.000000	49.000000	173.000000	8.000000	67.000000	12.000000
variation_coefficient	0.006078	0.485120	1.205155	1.072807	0.591438	1.108060	1.516405	1.361624	1.463660

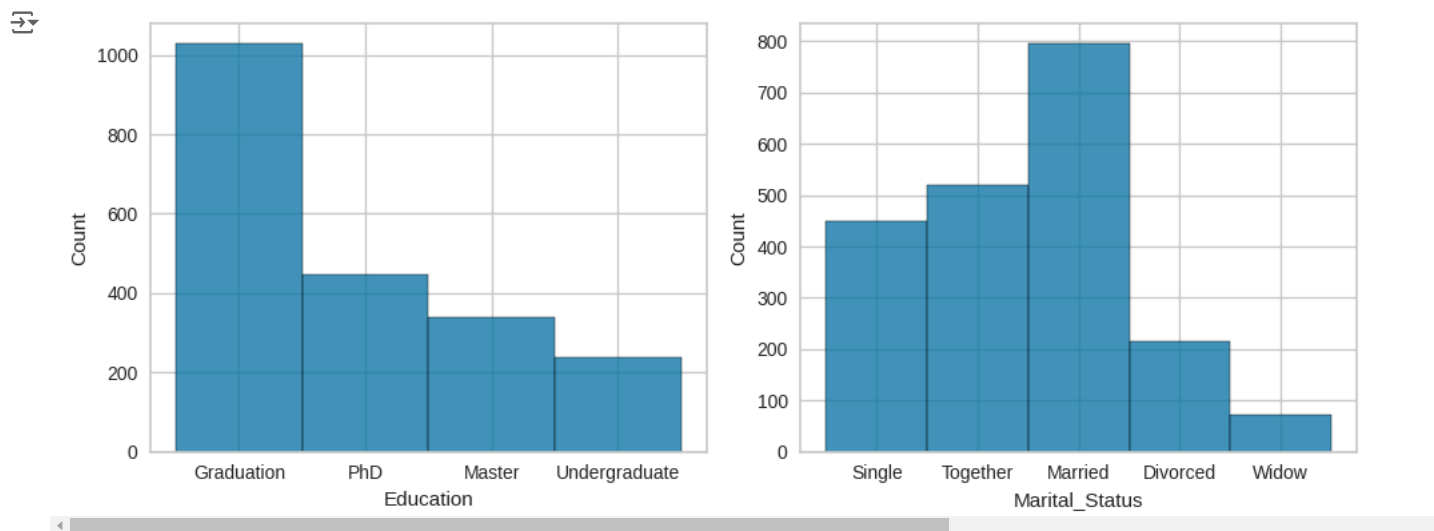
6 rows × 28 columns

Ahora visualizaremos graficos de barra columnas relevantes, las cuales han sido modificadas anteriormente

```
fig, axes = plt.subplots(nrows=1,ncols=2,figsize=(10,4))
```

```
sns.histplot(df['Education'],ax=axes[0])
sns.histplot(df['Marital_Status'],ax=axes[1])
```

```
plt.tight_layout()
plt.show()
```



Vemos que se ha reducido la cantidad de valores categoricos unicos para ambos features

Veremos las correlaciones entre las caracteristicas

```
from utils.eda import get_numeric_columns
numeric_columns = get_numeric_columns(df)
corr_matrix = df[numeric_columns].corr(method="pearson")
spearman = df[numeric_columns].corr(method="spearman")
kendall = df[numeric_columns].corr(method="kendall")
corr_matrix
```



	Year_Birth	Income	Kidhome	Teenhome	Recency	MntWines	MntFruits	MntMeatProducts	MntFishProducts	M
Year_Birth	1.000000	-0.161259	0.241456	-0.352218	-0.019727	-0.164427	-0.029775	-0.033933	-0.045933	
Income	-0.161259	1.000000	-0.425117	0.019610	-0.009385	0.561798	0.419160	0.566069	0.428075	
Kidhome	0.241456	-0.425117	1.000000	-0.045930	0.016170	-0.503243	-0.371992	-0.437269	-0.387099	
Teenhome	-0.352218	0.019610	-0.045930	1.000000	0.023400	0.006258	-0.177297	-0.259914	-0.195008	
Recency	-0.019727	-0.009385	0.016170	0.023400	1.000000	0.011055	-0.012217	0.013112	-0.002313	
MntWines	-0.164427	0.561798	-0.503243	0.006258	0.011055	1.000000	0.388262	0.555376	0.397172	
MntFruits	-0.029775	0.419160	-0.371992	-0.177297	-0.012217	0.388262	1.000000	0.541203	0.591956	
MntMeatProducts	-0.033933	0.566069	-0.437269	-0.259914	0.013112	0.555376	0.541203	1.000000	0.563551	
MntFishProducts	-0.045933	0.428075	-0.387099	-0.195008	-0.002313	0.397172	0.591956	0.563551	1.000000	
MntSweetProducts	-0.022770	0.426090	-0.368038	-0.162570	0.019437	0.379920	0.555469	0.517030	0.582155	
MntGoldProds	-0.057867	0.306417	-0.344628	-0.016658	0.009311	0.387090	0.395643	0.343533	0.414985	
NumDealsPurchases	-0.052441	-0.081372	0.214639	0.376492	0.008381	0.015482	-0.133619	-0.117171	-0.135633	
NumWebPurchases	-0.143584	0.366437	-0.366751	0.146067	-0.010427	0.537454	0.292014	0.286019	0.292966	
NumCatalogPurchases	-0.122953	0.574797	-0.501062	-0.108382	0.013303	0.627226	0.484506	0.722750	0.525971	
NumStorePurchases	-0.132286	0.513305	-0.503041	0.046459	-0.000487	0.638365	0.457024	0.471444	0.465154	
NumWebVisitsMonth	0.124699	-0.539611	0.443190	0.126418	-0.014920	-0.311116	-0.408936	-0.530549	-0.435294	
AcceptedCmp3	0.070830	-0.017234	0.009120	-0.044166	-0.034681	0.064208	0.012365	0.018092	-0.009776	
AcceptedCmp4	-0.063597	0.174290	-0.164865	0.045146	0.022911	0.361142	0.014710	0.090657	0.010281	
AcceptedCmp5	0.007447	0.327467	-0.209409	-0.186329	-0.004079	0.465964	0.217002	0.372540	0.189509	
AcceptedCmp1	0.006946	0.271078	-0.182637	-0.141724	-0.024731	0.357275	0.202926	0.317207	0.261841	
AcceptedCmp2	0.000379	0.085200	-0.085896	-0.025882	-0.009511	0.199531	-0.004983	0.045773	0.003650	
Complain	-0.025849	-0.026550	0.037606	-0.001724	0.007151	-0.037772	-0.002147	-0.021869	-0.018503	
Response	0.023652	0.129063	-0.082796	-0.157760	-0.202919	0.238826	0.129029	0.238396	0.104121	
Tot_children	-0.083701	-0.290910	0.684450	0.696854	0.028688	-0.356746	-0.396476	-0.503640	-0.420249	
Age	-0.989323	0.163219	-0.245679	0.362386	0.019520	0.169108	0.027388	0.034209	0.046996	
Tot_Expenses	-0.117346	0.650259	-0.561246	-0.136908	0.012273	0.889859	0.613635	0.840761	0.640540	
Tot_AcceptedCmp	0.015389	0.279912	-0.202179	-0.159536	-0.094286	0.482366	0.178552	0.330173	0.170169	
Tot_Purchases	-0.168619	0.549862	-0.483694	0.126908	0.003196	0.709734	0.451155	0.550293	0.470279	

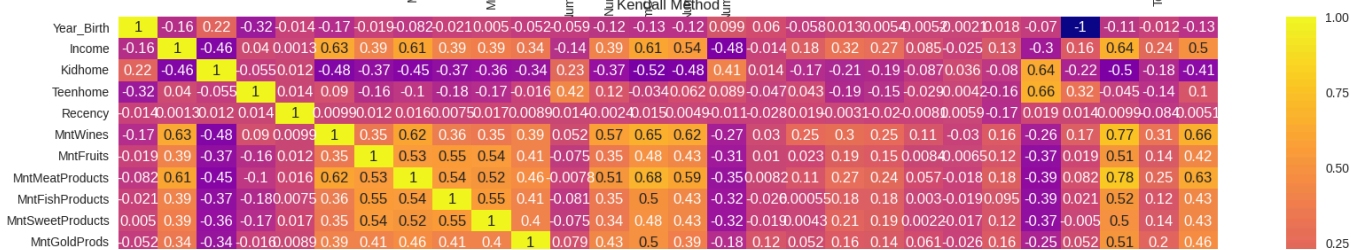
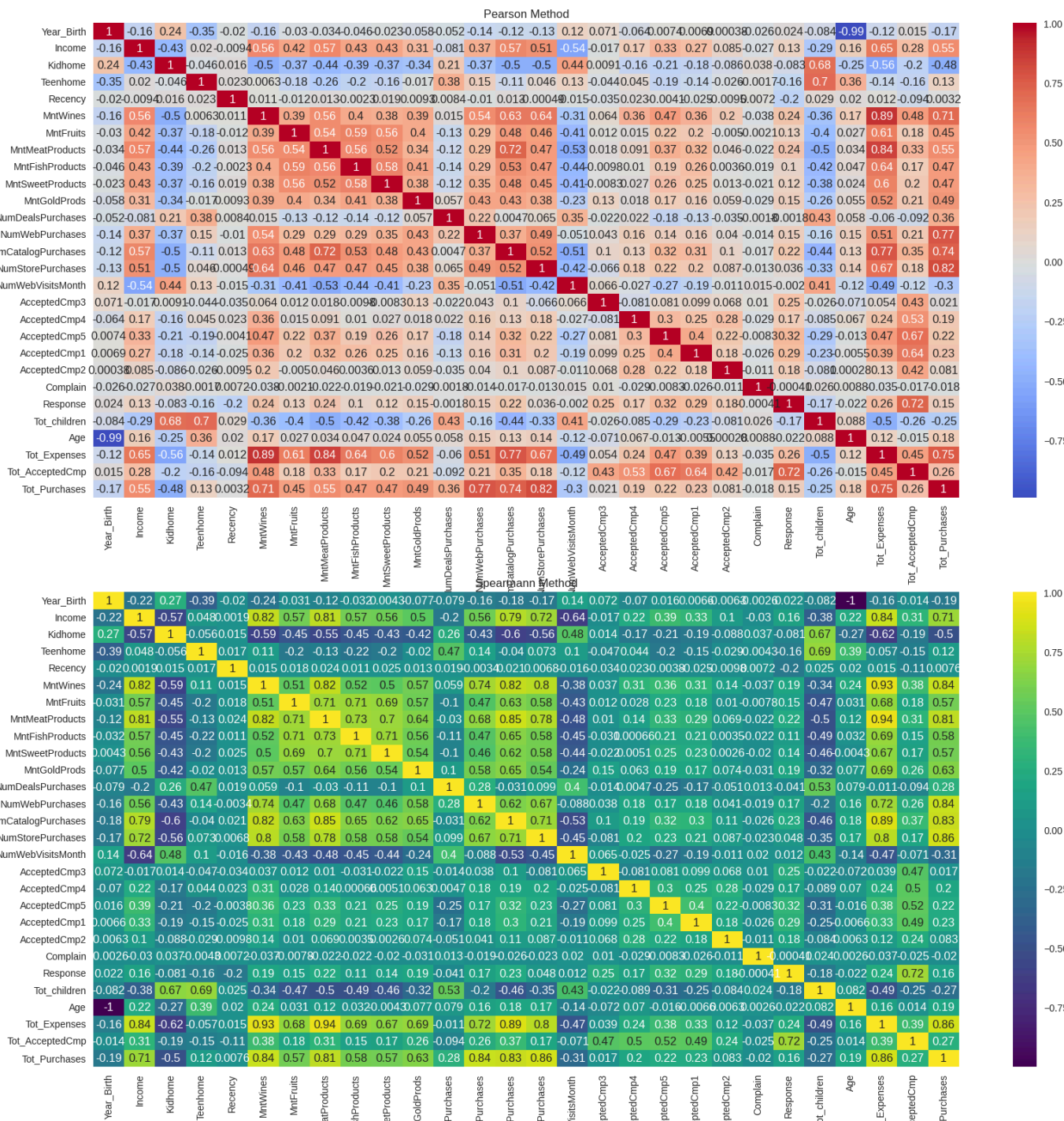
28 rows × 28 columns

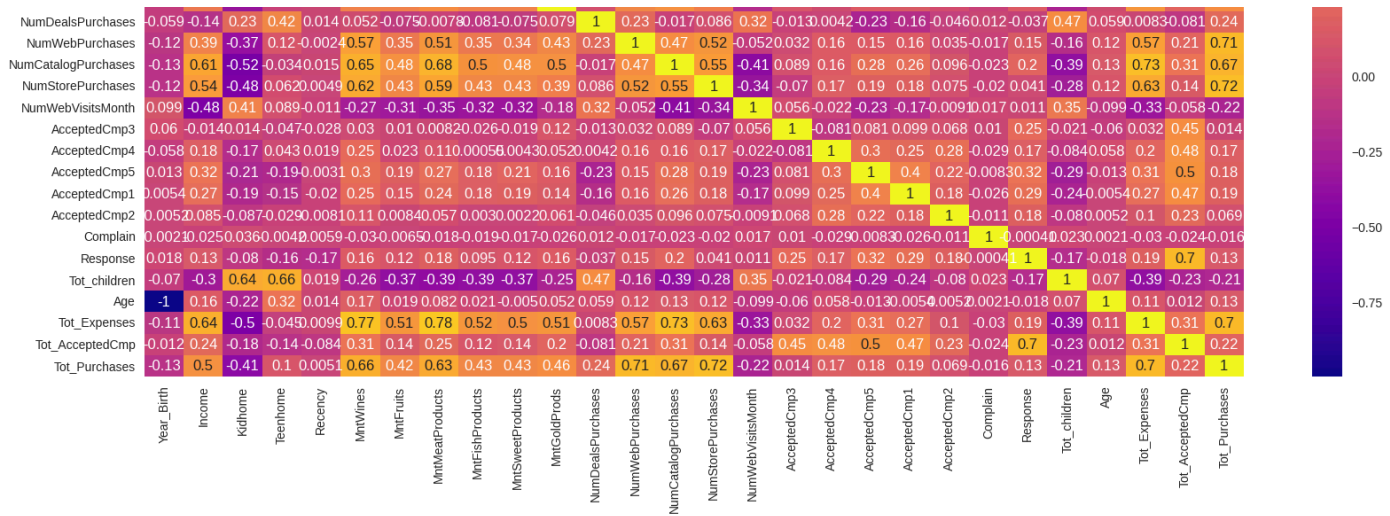
Ahora graficamos el mapa de calor de las correlaciones

```
from utils.eda import graph_correlations
graph_correlations(corr_matrix, spearmann, kendall, "Correlaciones variables de interés", figsize= (20,30))
```




Correlaciones variables de interés





Vemos que existen muchas correlaciones importantes, por ejemplo las existentes entre MntMeats MntFruits, MntSweet etc.

✓ Preprocesamos los datos

Aplicamos un LabelEncoder a las columnas de tipo object

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler, RobustScaler
df = df.drop(['Dt_Customer', 'agegroup'],axis=1)
#Convertimos datos categoricos a numericos para poder usarlos como input del modelo
LE = LabelEncoder()
for i in df:
    if df[i].dtype=='object':
        df[i] = LE.fit_transform(df[i])
```

Procedemos a normalizar los datos para luego aplicar PCA

```
from sklearn.cluster import KMeans
from yellowbrick.cluster import KElbowVisualizer
from sklearn.decomposition import PCA
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import RobustScaler

predict_columns = ['Recency', 'MntWines', 'MntFruits', 'MntMeatProducts', 'MntFishProducts',
                  'MntSweetProducts', 'MntGoldProds', 'NumDealsPurchases', 'NumWebPurchases',
                  'NumCatalogPurchases', 'NumStorePurchases', 'NumWebVisitsMonth', 'Tot_Expenses',
                  'Tot_Purchases', 'Age', 'Tot_children', 'Income']
preprocessor_scale = ColumnTransformer(
    transformers=[
        ("num", RobustScaler(), predict_columns),
    ],
    remainder="passthrough"
)
X = df[ [*predict_columns] ]

pca = Pipeline(steps=[
    ("preprocessor", preprocessor_scale),
    ("pca", PCA(n_components=0.9)),
])

X_pca = pca.fit_transform(X)
```

Graficamos la Varianza explicada acumulada por los componentes principales

```
# Variación explicada acumulada
explained_variance_cumulative = np.cumsum(pca.steps[-1][1].explained_variance_ratio_)

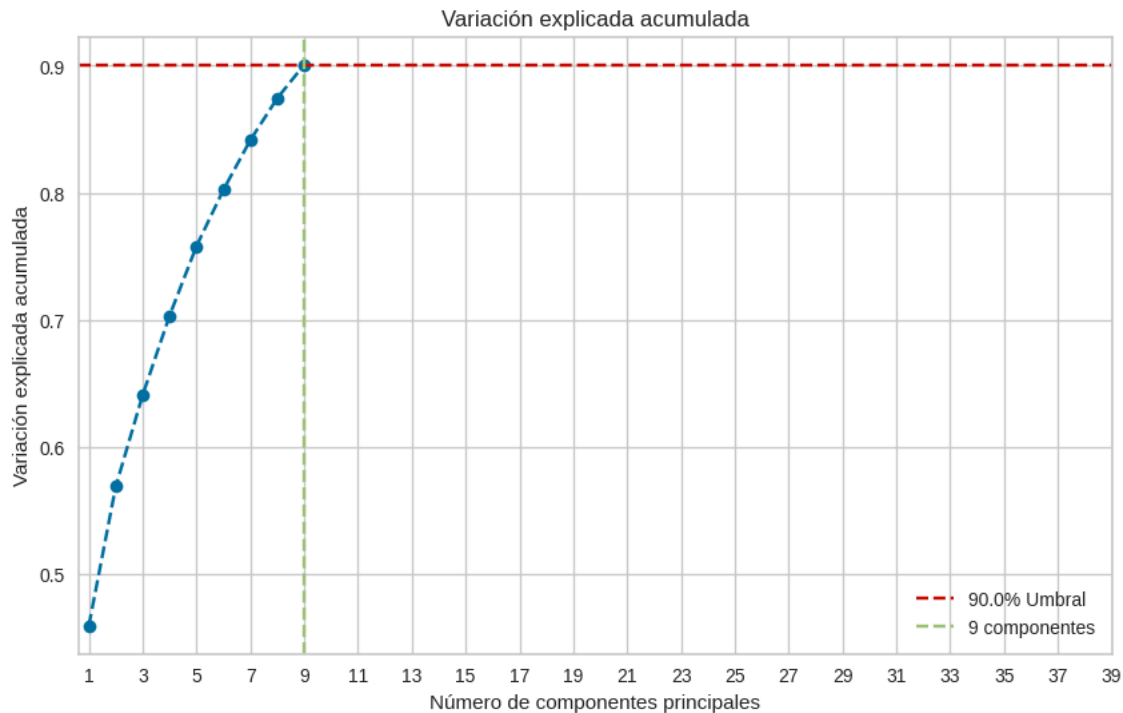
# Graficar
plt.figure(figsize=(10, 6))
plt.plot(range(1, 10), explained_variance_cumulative, marker='o', linestyle='--')
plt.title('Variación explicada acumulada')
plt.xlabel('Número de componentes principales')
plt.ylabel('Variación explicada acumulada')
plt.xticks(range(1, 41, 2)) # Mostrar solo cada dos componentes para claridad
plt.grid(True)

# Marcar el umbral del 90% como referencia
threshold = 0.90
optimal_components = np.argmax(explained_variance_cumulative >= threshold) + 1
plt.axhline(y=threshold, color='r', linestyle='--', label=f'{threshold * 100}% Umbral')
plt.axvline(x=optimal_components, color='g', linestyle='--', label=f'{optimal_components} componentes')

plt.legend()
plt.show()

# Mostrar la varianza explicada acumulada para cada componente
print("Variación explicada acumulada:")
```

```
for i, variance in enumerate(explained_variance_cumulative, 1):
    print(f"Componente {i}: {variance:.2%}")
```



Variación explicada acumulada:

Componente 1: 45.90%
 Componente 2: 56.86%
 Componente 3: 64.06%
 Componente 4: 70.32%
 Componente 5: 75.79%
 Componente 6: 80.23%
 Componente 7: 84.14%
 Componente 8: 87.44%
 Componente 9: 90.08%

- Se observa en el grafico cómo los primeros 9 componentes principales capturan la varianza explicada.
- Los 9 componentes seleccionados son suficientes para capturar al menos el 90% de la varianza de los datos.

A continuacion graficamos la ganancia de varianza

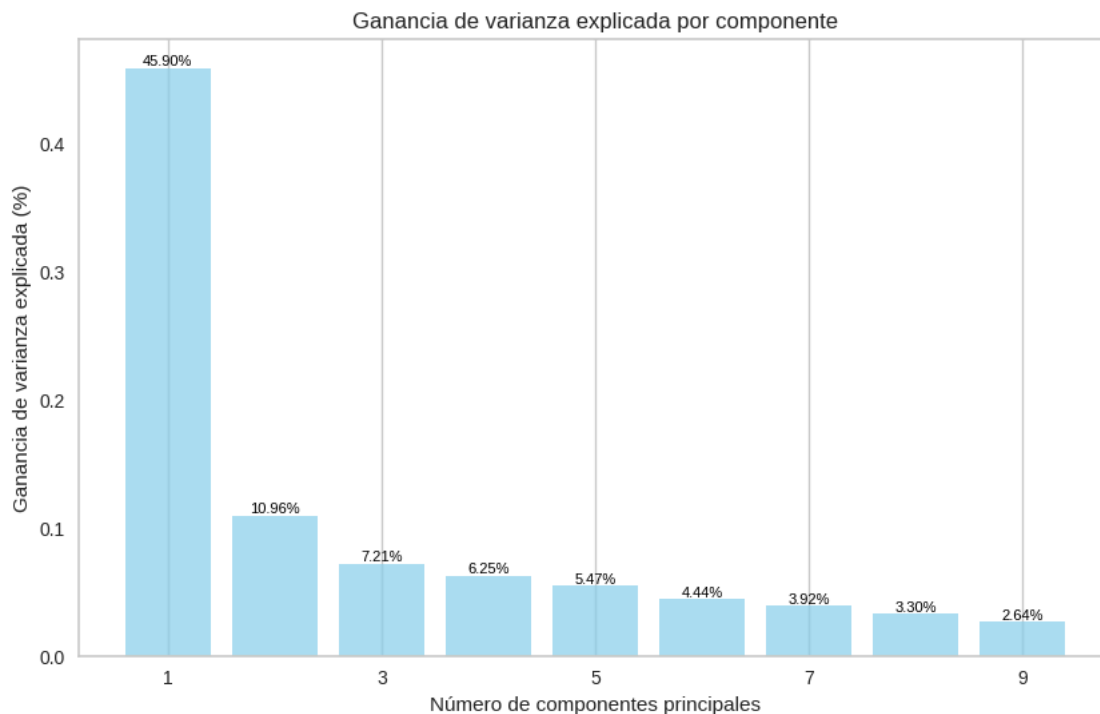
```
explained_variance_gain = np.diff(explained_variance_cumulative, prepend=0)

# Graficar la ganancia de varianza explicada
plt.figure(figsize=(10, 6))
plt.bar(range(1, 10), explained_variance_gain, color='skyblue', alpha=0.7)
plt.title('Ganancia de varianza explicada por componente')
plt.xlabel('Número de componentes principales')
plt.ylabel('Ganancia de varianza explicada (%)')
plt.xticks(range(1, 11, 2)) # Mostrar solo cada dos componentes para claridad
plt.grid(axis='y')

# Resaltar las primeras componentes principales más significativas
threshold_gain = 0.01 # Umbral de ganancia significativa (1%)
significant_components = [i for i, gain in enumerate(explained_variance_gain, 1) if gain > threshold_gain]
for component in significant_components:
    plt.text(component, explained_variance_gain[component - 1], f"{explained_variance_gain[component - 1]:.2%}",
             ha='center', va='bottom', fontsize=8, color='black')

plt.show()

# Mostrar las ganancias para cada componente
print("Ganancia de varianza explicada por componente:")
for i, gain in enumerate(explained_variance_gain, 1):
    print(f"Componente {i}: {gain:.2%}")
```



Ganancia de varianza explicada por componente:

Componente 1: 45.90%

Componente 2: 10.96%

Componente 3: 7.21%

Componente 4: 6.25%

Componente 5: 5.47%

Componente 6: 4.44%

Componente 7: 3.92%

Componente 8: 3.30%

Componente 9: 2.64%

- Se observa como los primeros 2 componentes son los de mayor aporte.
- Según los gráficos de componentes principales, el método puede mantener el 90% de la varianza acumulada solamente usando 9 componentes o features. Esta disminución en cantidad de componentes se traduce en menores tiempos de entrenamiento, y también facilita la visualización y análisis de patrones en los datos.

Ahora graficaremos La proyeccion PCA de los primeros 2 componentes principales

```
# Crear un gráfico con dos subgráficos, uno para cada imagen
fig, ax = plt.subplots(1, 2, figsize=(20, 8))

# Primer subgráfico
for i in range(0, 4):
    subset = X_pca[df["Education"] == i][:, :2] # Usar solo las primeras 2 componentes
    ax[0].scatter(subset[:, 0], subset[:, 1], label=f'{i}', alpha=0.7)

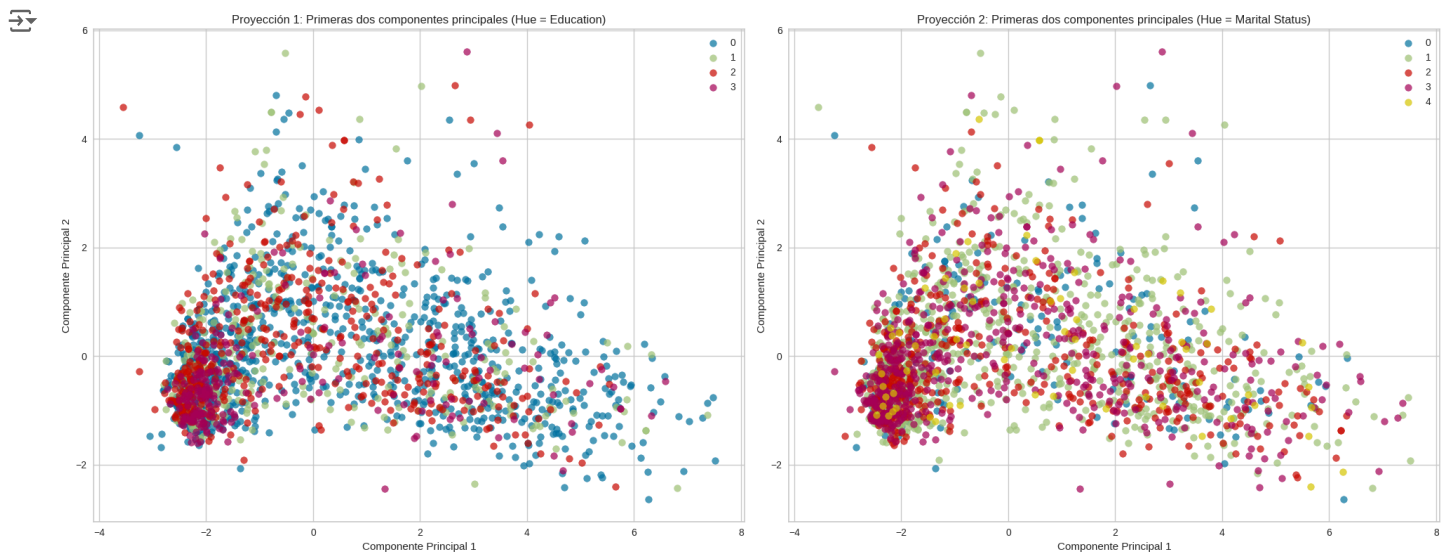
ax[0].set_title('Proyección 1: Primeras dos componentes principales (Hue = Education)')
ax[0].set_xlabel('Componente Principal 1')
ax[0].set_ylabel('Componente Principal 2')
ax[0].legend()
ax[0].grid(True)

# Segundo subgráfico
for i in range(0, 5):
    subset = X_pca[df["Marital_Status"] == i][:, :2] # Usar solo las primeras 2 componentes
    ax[1].scatter(subset[:, 0], subset[:, 1], label=f'{i}', alpha=0.7)

ax[1].set_title('Proyección 2: Primeras dos componentes principales (Hue = Marital Status)')
ax[1].set_xlabel('Componente Principal 1')
ax[1].set_ylabel('Componente Principal 2')
ax[1].legend()
ax[1].grid(True)

# Ajustar el diseño y mostrar el gráfico
plt.tight_layout()
```

```
plt.show()
```



Se puede apreciar como se distribuyen los datos en el espacio reducido de los primeros 2 componentes principales.

La proyección en 2D ayuda a entender la estructura general de los datos, aunque puede que no se capturen agrupamientos claros debido a la pérdida de información en la reducción

A continuacion observaron el espacio reducido de los primeros 3 componentes principales

```
from mpl_toolkits.mplot3d import Axes3D

# Plot the first 3 components in a 3D space
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

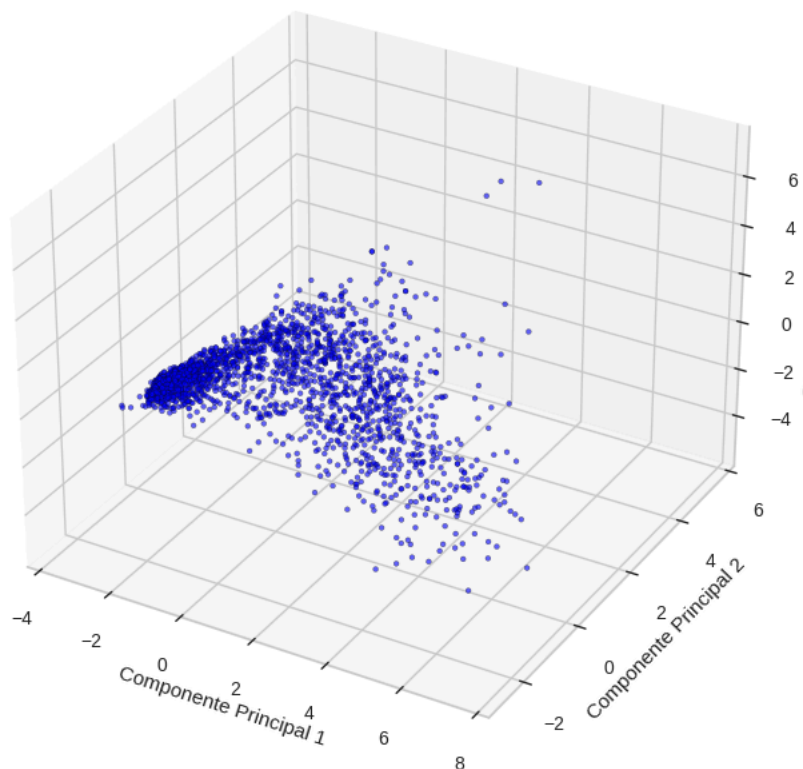
# Scatter plot with colors
ax.scatter(X_pca[:, 0], X_pca[:, 1], X_pca[:, 2], c='blue', marker='.', alpha=0.6, edgecolor='k', s=30)

# Add labels and title
ax.set_title('Proyección PCA (Primeros 3 Componentes Principales)')
ax.set_xlabel('Componente Principal 1')
ax.set_ylabel('Componente Principal 2')
ax.set_zlabel('Componente Principal 3')

# Show the plot
plt.show()
```




Proyección PCA (Primeros 3 Componentes Principales)



El espacio reducido de los 3 primeros componentes principales

Analizamos los coeficientes de cada componente principal para identificar qué variables tienen la mayor influencia en cada uno de ellos.

```
loadings = pd.DataFrame(
    pca.steps[-1][1].components_.T, # Transponer para tener variables como filas
    columns = [f'PC{i+1}' for i in range(pca.steps[-1][1].n_components_)],
    index = predict_columns
)
```

```
# Mostrar resultados
print("Coeficientes (loadings) de cada componente principal:")
print(loadings)
```

```
# Analizar las variables más influyentes
for i in range(pca.steps[-1][1].n_components_):
    most_influyente = loadings[f'PC{i+1}'].abs().idxmax()
    print(f"La variable más influyente en PC{i+1} es: {most_influyente}")
```



Coeficientes (loadings) de cada componente principal:

	PC1	PC2	PC3	PC4	PC5	\
Recency	0.001194	0.007410	0.000304	-0.012107	-0.038265	
MntWines	0.195120	0.166598	0.287442	0.034588	-0.041976	
MntFruits	0.399893	-0.164411	-0.335826	-0.078979	0.726863	
MntMeatProducts	0.353429	-0.064786	0.423431	-0.074433	0.067973	
MntFishProducts	0.378370	-0.133907	-0.206036	-0.034156	0.106629	
MntSweetProducts	0.405018	-0.134788	-0.420742	-0.441307	-0.613979	
MntGoldProds	0.273254	0.335438	-0.333836	0.770825	-0.156250	
NumDealsPurchases	-0.049365	0.696801	-0.134698	-0.379668	0.179379	
NumWebPurchases	0.145520	0.306680	0.010248	0.012931	-0.095982	
NumCatalogPurchases	0.233655	0.088656	0.250807	0.000444	-0.006001	
NumStorePurchases	0.181869	0.142045	0.119107	-0.043985	-0.002669	
NumWebVisitsMonth	-0.147318	0.157087	-0.168803	-0.029847	0.039032	
Tot_Expenses	0.231802	0.067757	0.183374	0.018706	-0.005138	
Tot_Purchases	0.179024	0.283475	0.105412	-0.071213	-0.004809	
Age	0.025442	0.105653	0.112682	-0.029770	-0.013114	
Tot_children	-0.173698	0.248402	-0.153985	-0.201905	0.078218	
Income	0.213639	0.047245	0.299003	-0.050743	-0.042124	
	PC6	PC7	PC8	PC9		

Recency	0.006594	-0.018229	0.172341	0.920804
MntWines	-0.124379	0.126796	-0.246505	0.097108
MntFruits	-0.399962	0.011270	0.016743	0.032313
MntMeatProducts	0.131646	-0.496004	0.237594	0.036820
MntFishProducts	0.820052	0.293196	-0.119485	0.009055
MntSweetProducts	-0.209234	-0.089632	0.076619	-0.020150
MntGoldProds	-0.024653	-0.125137	0.232531	-0.049833
NumDealsPurchases	0.182884	-0.263914	0.096493	-0.045058
NumWebPurchases	-0.161710	0.213337	-0.425979	0.075760
NumCatalogPurchases	0.044998	-0.125027	0.110967	0.006736
NumStorePurchases	-0.082585	0.210401	-0.227469	0.008616
NumWebVisitsMonth	0.016193	-0.100951	-0.289843	0.118724
Tot_Expenses	-0.014430	-0.040663	-0.060996	0.054389
Tot_Purchases	-0.039539	0.067493	-0.169570	0.021765
Age	-0.074243	0.592049	0.426500	0.135180
Tot_children	0.025109	0.174594	0.407969	-0.091557
Income	-0.117033	0.235196	0.242767	-0.291969

La variable más influyente en PC1 es: MntSweetProducts
 La variable más influyente en PC2 es: NumDealsPurchases
 La variable más influyente en PC3 es: MntMeatProducts
 La variable más influyente en PC4 es: MntGoldProds
 La variable más influyente en PC5 es: MntFruits
 La variable más influyente en PC6 es: MntFishProducts
 La variable más influyente en PC7 es: Age
 La variable más influyente en PC8 es: Age
 La variable más influyente en PC9 es: Recency

✓ Modelado MLP

Implementa un Perceptrón Multicapa (MLP) para predecir las compras futuras de los clientes o **su probabilidad de realizar una compra en una categoría específica de productos**.

Objetivo:

Predecir las probabilidad de compra para categoría MntSweetProducts

✓ Implementamos un MLP

```
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2
```

```
# Métricas
# Evaluación
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_auc_score,
)
```

✓ Dividimos el Dataset en Conjuntos de entrenamiento y prueba

```
X_pca = pca.fit_transform(X)

# Definimos el target
y = df['Response']

# Separamos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y, test_size=0.3, stratify=y, random_state=42
)
```

Inicializamos el modelo y le agregamos las características (capas, activadores, etc)


```

from imblearn.over_sampling import BorderlineSMOTE # Utilizamos este para tener la mejor respuesta
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.regularizers import l2
from sklearn.metrics import roc_auc_score, classification_report

# Crear variable binaria para la categoría de producto seleccionada ("MntSweetProducts")
df['MntSweetProducts_Buy'] = (df['MntSweetProducts'] > 0).astype(int)

# Características predictoras
X = df[['Recency', 'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
        'NumStorePurchases', 'NumWebVisitsMonth', 'Tot_Expenses', 'Tot_Purchases',
        'Age', 'Tot_children', 'Income']]

# Variable objetivo (probabilidad de compra en "MntSweetProducts")
y = df['MntSweetProducts_Buy']

# Normalización de las características
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dividir en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, stratify=y, random_state=42)

# Aplicar Borderline-SMOTE para balancear las clases
smote = BorderlineSMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)

# Imprimir distribución después de aplicar SMOTE
print("Distribución de clases después de Borderline-SMOTE:")
print(pd.Series(y_train_res).value_counts())

# Inicializar el modelo
model = Sequential()

# Capa de entrada y primera capa oculta
model.add(Dense(128, activation='relu', kernel_regularizer=l2(0.01), input_dim=X_train_res.shape[1]))
model.add(BatchNormalization()) # Normalización por lotes para estabilizar el entrenamiento
model.add(Dropout(0.3)) # Dropout a 30%

# Segunda capa oculta
model.add(Dense(64, activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

# Tercera capa oculta
model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

# Capa de salida (probabilidad de compra)
model.add(Dense(1, activation='sigmoid'))

# Compilación del modelo
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Resumen del modelo
model.summary()

```

↗ Distribución de clases después de Borderline-SMOTE:
 MntSweetProducts_Buy
 1 1168
 0 1168
 Name: count, dtype: int64
 /usr/local/lib/python3.10/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input`
 super().__init__(activity_regularizer=activity_regularizer, **kwargs)
 Model: "sequential_49"

Layer (type)	Output Shape	Param #
dense_151 (Dense)	(None, 128)	1,536
batch_normalization_9 (BatchNormalization)	(None, 128)	512
dropout_72 (Dropout)	(None, 128)	0
dense_152 (Dense)	(None, 64)	8,256
batch_normalization_10 (BatchNormalization)	(None, 64)	256
dropout_73 (Dropout)	(None, 64)	0
dense_153 (Dense)	(None, 32)	2,080
batch_normalization_11 (BatchNormalization)	(None, 32)	128
dropout_74 (Dropout)	(None, 32)	0
dense_154 (Dense)	(None, 1)	33

Total params: 12,801 (50.00 KB)
 Trainable params: 12,353 (48.25 KB)
 Non-trainable params: 448 (1.75 KB)

Utilizamos una construccion especifica para tener una respuesta aceptable del modelo, con BorderlineSMOTE, Dropout seteados a valores menores a 0.5 y BathNormalization

Entrenamos el modelo y seteamos los parametros de entrenamiento. Aplicamos earlyStopping y epoch de 100

```
# Entrenamiento del modelo con Early Stopping
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

history = model.fit(
    X_train_res, y_train_res,
    validation_data=(X_test, y_test),
    epochs=100,
    batch_size=32,
    verbose=1,
    callbacks=[early_stopping]
)
```

↗ Epoch 9/100
 73/73 ————— 0s 4ms/step - accuracy: 0.7349 - loss: 0.9656 - val_accuracy: 0.6786 - val_loss: 0.9137
 Epoch 10/100
 73/73 ————— 0s 4ms/step - accuracy: 0.7457 - loss: 0.9060 - val_accuracy: 0.7029 - val_loss: 0.8637
 Epoch 11/100
 73/73 ————— 0s 4ms/step - accuracy: 0.7586 - loss: 0.8463 - val_accuracy: 0.7159 - val_loss: 0.8113

```

Epoch 20/100
73/73 ————— 0s 5ms/step - accuracy: 0.7761 - loss: 0.6182 - val_accuracy: 0.6964 - val_loss: 0.6636
Epoch 21/100
73/73 ————— 1s 5ms/step - accuracy: 0.7769 - loss: 0.6181 - val_accuracy: 0.6997 - val_loss: 0.6655
Epoch 22/100
73/73 ————— 1s 6ms/step - accuracy: 0.7838 - loss: 0.6083 - val_accuracy: 0.7192 - val_loss: 0.6454
Epoch 23/100
73/73 ————— 1s 6ms/step - accuracy: 0.8035 - loss: 0.5690 - val_accuracy: 0.7305 - val_loss: 0.6353
Epoch 24/100
73/73 ————— 0s 3ms/step - accuracy: 0.7876 - loss: 0.5620 - val_accuracy: 0.7094 - val_loss: 0.6547
Epoch 25/100
73/73 ————— 0s 3ms/step - accuracy: 0.7947 - loss: 0.5595 - val_accuracy: 0.7078 - val_loss: 0.6328
Epoch 26/100
73/73 ————— 0s 3ms/step - accuracy: 0.8023 - loss: 0.5473 - val_accuracy: 0.7256 - val_loss: 0.6244
Epoch 27/100
73/73 ————— 0s 4ms/step - accuracy: 0.7829 - loss: 0.5597 - val_accuracy: 0.7695 - val_loss: 0.5791
Epoch 28/100
73/73 ————— 0s 3ms/step - accuracy: 0.8045 - loss: 0.5277 - val_accuracy: 0.7468 - val_loss: 0.5887
Epoch 29/100
73/73 ————— 0s 4ms/step - accuracy: 0.8058 - loss: 0.5377 - val_accuracy: 0.6834 - val_loss: 0.6451
Epoch 30/100
73/73 ————— 0s 4ms/step - accuracy: 0.7931 - loss: 0.5298 - val_accuracy: 0.7321 - val_loss: 0.6192
Epoch 31/100
73/73 ————— 1s 3ms/step - accuracy: 0.8145 - loss: 0.5097 - val_accuracy: 0.6948 - val_loss: 0.6484
Epoch 32/100
73/73 ————— 0s 3ms/step - accuracy: 0.8003 - loss: 0.5310 - val_accuracy: 0.7192 - val_loss: 0.6268
Epoch 33/100
73/73 ————— 0s 4ms/step - accuracy: 0.7991 - loss: 0.5415 - val_accuracy: 0.7029 - val_loss: 0.6298
Epoch 34/100
73/73 ————— 0s 4ms/step - accuracy: 0.7936 - loss: 0.5372 - val_accuracy: 0.7175 - val_loss: 0.6218
Epoch 35/100
73/73 ————— 0s 4ms/step - accuracy: 0.8131 - loss: 0.5140 - val_accuracy: 0.7078 - val_loss: 0.6351
Epoch 36/100
73/73 ————— 0s 4ms/step - accuracy: 0.8068 - loss: 0.5061 - val_accuracy: 0.7256 - val_loss: 0.6305
Epoch 37/100
73/73 ————— 0s 4ms/step - accuracy: 0.7998 - loss: 0.5339 - val_accuracy: 0.7029 - val_loss: 0.6340

```

Evaluamos nuestro modelo

```

# Evaluación del modelo
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Pérdida en el conjunto de prueba: {loss:.4f}')
print(f'Precisión en el conjunto de prueba: {accuracy:.2%}')

20/20 ————— 0s 2ms/step - accuracy: 0.7876 - loss: 0.5415
Pérdida en el conjunto de prueba: 0.5791
Precisión en el conjunto de prueba: 76.95%

```

Obtenemos buenos valores de Precision y perdida. La precision no es muy alta, pero es aceptable, la perdida se encuentra en un valor relativamente cercano a la precision

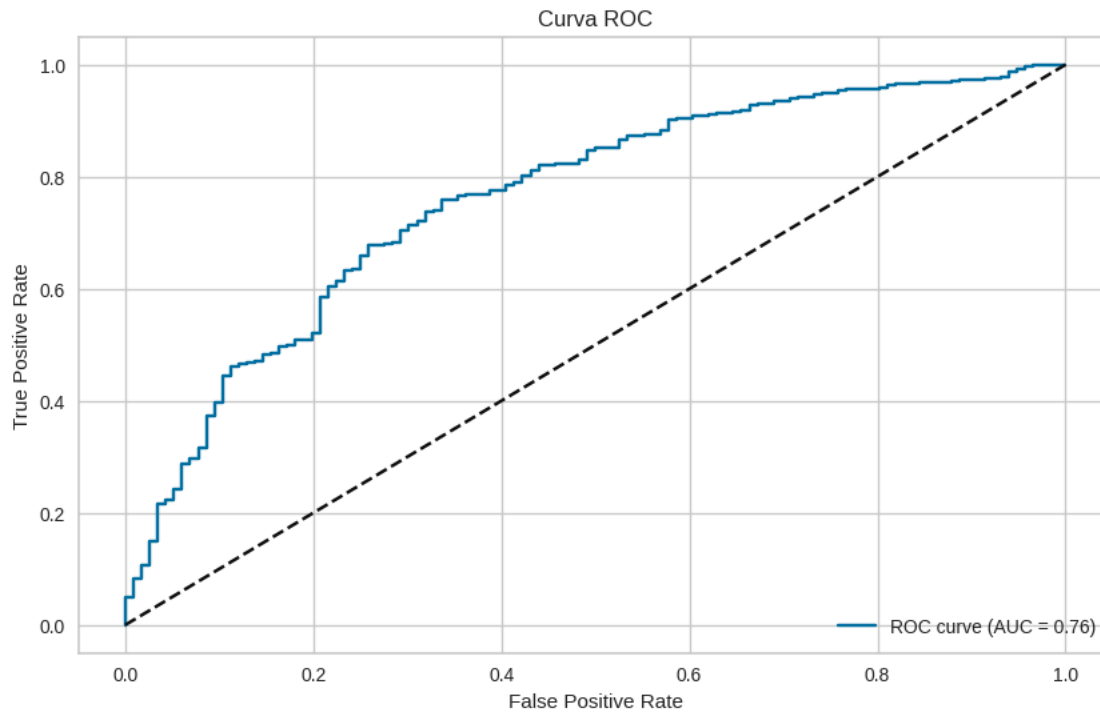
Graficamos la curva ROC

```

# Gráfica de curvas ROC
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_test_pred_prob)
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Curva ROC')
plt.legend(loc='lower right')
plt.grid(True)
plt.show()

```



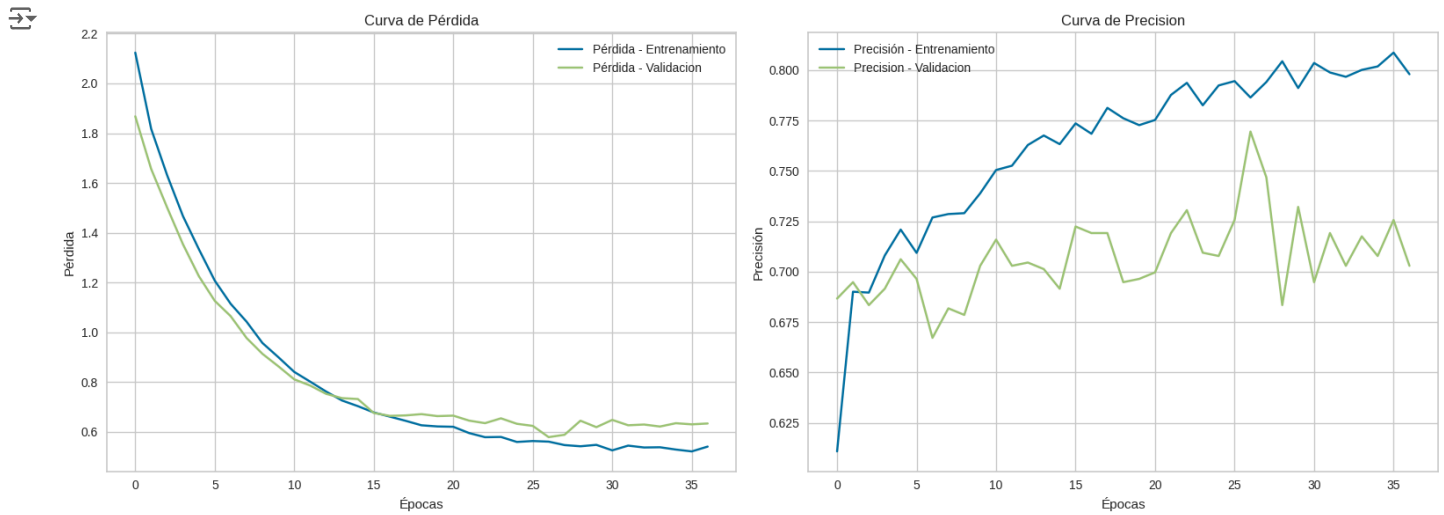
Ahora Generamos las curvas de perdida y de precision

```
fig, ax = plt.subplots(1, 2, figsize=(16, 6))

# Curva perdida
ax[0].plot(history.history['loss'], label='Pérdida - Entrenamiento')
ax[0].plot(history.history['val_loss'], label='Pérdida - Validacion')
ax[0].set_title('Curva de Pérdida')
ax[0].set_xlabel('Épocas')
ax[0].set_ylabel('Pérdida')
ax[0].legend()
ax[0].grid(True)

# Curva precision
ax[1].plot(history.history['accuracy'], label='Precisión - Entrenamiento')
ax[1].plot(history.history['val_accuracy'], label='Precision - Validacion')
ax[1].set_title('Curva de Precision')
ax[1].set_xlabel('Épocas')
ax[1].set_ylabel('Precisión')
ax[1].legend()
ax[1].grid(True)

# Combinamos plots
plt.tight_layout()
plt.show()
```



Curva de perdida

Entrenamiento

- La pérdida disminuye de manera continua y suave, lo que indica que el modelo está aprendiendo correctamente a lo largo de las épocas.

Validacion

- Al principio sigue una tendencia similar a la pérdida de entrenamiento, pero a partir de la época 10-15, se estabiliza y deja de disminuir significativamente.

Curva de precision

Entrenamiento

- Aumenta de forma consistente y llega cerca de 80%, mostrando que el modelo mejora en el entrenamiento.

Validacion

- Es inestable, con fluctuaciones importantes, lo que indica que el modelo puede tener datos de validación que no son suficientemente informativos para el modelo.

Discusion de la arquitectura del modelo MLP

Capas de Entrada:

- El tamaño de entrada se define en función de las características predictoras (11 features: `input_dim=X_train_res.shape[1]`).

Capas Ocultas:

Primera capa oculta:

- Número de Neuronas: 128 neuronas.
- Función de Activación: ReLU (Rectified Linear Unit) para introducir no linealidad.
- Regularización: L2 (con `kernel_regularizer=l2(0.01)`) para evitar el sobreajuste.
- Batch Normalization: Se aplica normalización por lotes después de la activación para estabilizar y acelerar el entrenamiento.
- Dropout: 30% de dropout (`Dropout(0.3)`) para reducir el riesgo de sobreajuste eliminando aleatoriamente conexiones.

Segunda capa oculta:

- Número de Neuronas: 64 neuronas.
- Función de Activación: ReLU.
- Regularización L2: Se mantiene la misma regularización para controlar la complejidad del modelo.

- Batch Normalization: Se aplica para mantener valores normalizados durante el entrenamiento.
- Dropout: 30% de dropout.

Tercera capa oculta

- Número de Neuronas: 32 neuronas.
- Función de Activación: ReLU.
- Regularización L2: Control del sobreajuste.
- Batch Normalization: Aplicada después de la activación.
- Dropout: 20% de dropout.

Capa de Salida:

- Número de Neuronas: 1 neurona.
- Función de Activación: sigmoid para modelar la probabilidad de compra (clasificación binaria: 1 = compra, 0 = no compra).

Diseño de modelo:

- Número de Capas Ocultas: 3 capas ocultas con 128, 64 y 32 neuronas, respectivamente. La reducción gradual del número de neuronas ayuda a evitar la complejidad innecesaria.
- Regularización:
- L2 Regularization: Agrega penalización al peso de las conexiones, reduciendo el sobreajuste y mejorando la generalización.
- Dropout: Apagado aleatorio de unidades en cada capa para reducir el riesgo de co-adaptación de neuronas.
- Batch Normalization: Normaliza los valores de salida en cada capa, lo que mejora la estabilidad y acelera la convergencia durante el entrenamiento.

Fortalezas de la Arquitectura:

- Equilibrio entre Capacidad y Regularización: La estructura asegura suficiente capacidad de aprendizaje mientras controla el sobreajuste.
- Estabilidad: La normalización por lotes mejora la estabilidad y permite un aprendizaje más rápido.
- Generalización: El uso de técnicas como Dropout y L2 regularization mejora la capacidad del modelo para generalizar en datos no vistos.

✓ Evaluacion del modelo

Evaluamos metricas de precision, recall, f1-score y support

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Realizar predicciones en los conjuntos de entrenamiento y prueba
y_train_pred = (model.predict(X_train) > 0.5).astype(int) # Convertir probabilidades a 0 o 1
y_test_pred = (model.predict(X_test) > 0.5).astype(int)

# Métricas de clasificación para el conjunto de entrenamiento
print("Métricas para el conjunto de entrenamiento:")
print(classification_report(y_train, y_train_pred))

# Métricas de clasificación para el conjunto de prueba
print("Métricas para el conjunto de prueba:")
print(classification_report(y_test, y_test_pred))
```

```
→ 45/45 ————— 0s 3ms/step
20/20 ————— 0s 1ms/step
Métricas para el conjunto de entrenamiento:
      precision    recall  f1-score   support

     0       0.50      0.74      0.60        269
     1       0.93      0.83      0.88       1168

 accuracy          0.81        1437
 macro avg       0.72      0.79      0.74        1437
 weighted avg    0.85      0.81      0.82        1437

Métricas para el conjunto de prueba:
      precision    recall  f1-score   support
```