CAPSTONE

Aplicación de modelos de predicción para identificar patrones de deserción en los datos de clientes de una entidad financiera.

INTEGRANTES:

Ana Cristina Prado
Gladys Margarita Yambay

CASO DE ESTUDIO

La deserción de clientes en una entidad financiera.

Es fundamental para la entidad financiera en estudio identificar proactivamente la probabilidad de que el cliente deserte o no de la entidad, ya que ésto no solo representa una pérdida directa de ingresos, sino que también aumenta los costos operativos, ya que para la entidad adquirir nuevos clientes es significativamente más costoso que retener a los existentes. Identificar y comprender las razones detrás de la deserción permite a quienes toman decisiones en la entidad, anticiparse y mitigar estos riesgos. Un análisis eficaz de la deserción contribuirá a la mejora continua de productos y servicios, incrementando la competitividad y rentabilidad de la entidad financiera en un mercado dinámico y exigente.

El objetivo es desarrollar un modelo predictivo para clasificar a los clientes de la entidad financiera según su probabilidad de deserción, utilizando algoritmos de machine learning como son: los árboles de decisión, random forest y regresión logística, con la finalidad de segmentar eficazmente a los clientes en función de su riesgo de abandono, optimizando así las estrategias de retención y reduciendo costos asociados a las campañas de retención.

In [8]:
```
#Install
!pip install lifelines
```

```
Requirement already satisfied: lifelines in d:\anaconda\lib\site-packages (0.2
9.0)
Requirement already satisfied: numpy<2.0,>=1.14.0 in d:\anaconda\lib\site-packag
es (from lifelines) (1.24.3)
Requirement already satisfied: scipy>=1.7.0 in d:\anaconda\lib\site-packages (fr
om lifelines) (1.11.1)
Requirement already satisfied: pandas>=2.1 in d:\anaconda\lib\site-packages (fro
m lifelines) (2.2.2)
Requirement already satisfied: matplotlib>=3.0 in d:\anaconda\lib\site-packages
(from lifelines) (3.7.2)
Requirement already satisfied: autograd>=1.5 in d:\anaconda\lib\site-packages (f
rom lifelines) (1.7.0)
Requirement already satisfied: autograd-gamma>=0.3 in d:\anaconda\lib\site-packa
ges (from lifelines) (0.5.0)
Requirement already satisfied: formulaic>=0.2.2 in d:\anaconda\lib\site-packages
(from lifelines) (1.0.2)
Requirement already satisfied: interface-meta>=1.2.0 in d:\anaconda\lib\site-pac
kages (from formulaic>=0.2.2->lifelines) (1.3.0)
Requirement already satisfied: typing-extensions>=4.2.0 in d:\anaconda\lib\site-
packages (from formulaic>=0.2.2->lifelines) (4.7.1)
Requirement already satisfied: wrapt>=1.0 in d:\anaconda\lib\site-packages (from
formulaic>=0.2.2->lifelines) (1.14.1)
Requirement already satisfied: contourpy>=1.0.1 in d:\anaconda\lib\site-packages
(from matplotlib>=3.0->lifelines) (1.0.5)
Requirement already satisfied: cycler>=0.10 in d:\anaconda\lib\site-packages (fr
om matplotlib>=3.0->lifelines) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in d:\anaconda\lib\site-package
s (from matplotlib>=3.0->lifelines) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\anaconda\lib\site-package
s (from matplotlib>=3.0->lifelines) (1.4.4)
Requirement already satisfied: packaging>=20.0 in d:\anaconda\lib\site-packages
(from matplotlib>=3.0->lifelines) (23.1)
Requirement already satisfied: pillow>=6.2.0 in d:\anaconda\lib\site-packages (f
rom matplotlib>=3.0->lifelines) (9.4.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in d:\anaconda\lib\site-pac
kages (from matplotlib>=3.0->lifelines) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in d:\anaconda\lib\site-pack
ages (from matplotlib>=3.0->lifelines) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in d:\anaconda\lib\site-packages (fr
om pandas>=2.1->lifelines) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in d:\anaconda\lib\site-packages
(from pandas>=2.1->lifelines) (2023.3)
Requirement already satisfied: six>=1.5 in d:\anaconda\lib\site-packages (from p
ython-dateutil>=2.7->matplotlib>=3.0->lifelines) (1.16.0)
```

In [9]:
```
!pip install random-survival-forest
```

```
Requirement already satisfied: random-survival-forest in d:\anaconda\lib\site-pa
ckages (0.8.2)
Requirement already satisfied: numpy in d:\anaconda\lib\site-packages (from rand
om-survival-forest) (1.24.3)
Requirement already satisfied: pandas in d:\anaconda\lib\site-packages (from ran
dom-survival-forest) (2.2.2)
Requirement already satisfied: joblib in d:\anaconda\lib\site-packages (from ran
dom-survival-forest) (1.2.0)
Requirement already satisfied: multiprocess in d:\anaconda\lib\site-packages (fr
om random-survival-forest) (0.70.14)
Requirement already satisfied: lifelines in d:\anaconda\lib\site-packages (from
random-survival-forest) (0.29.0)
Requirement already satisfied: scikit-learn in d:\anaconda\lib\site-packages (fr
om random-survival-forest) (1.5.1)
Requirement already satisfied: scipy>=1.7.0 in d:\anaconda\lib\site-packages (fr
om lifelines->random-survival-forest) (1.11.1)
Requirement already satisfied: matplotlib>=3.0 in d:\anaconda\lib\site-packages
(from lifelines->random-survival-forest) (3.7.2)
Requirement already satisfied: autograd>=1.5 in d:\anaconda\lib\site-packages (f
rom lifelines->random-survival-forest) (1.7.0)
Requirement already satisfied: autograd-gamma>=0.3 in d:\anaconda\lib\site-packa
ges (from lifelines->random-survival-forest) (0.5.0)
Requirement already satisfied: formulaic>=0.2.2 in d:\anaconda\lib\site-packages
(from lifelines->random-survival-forest) (1.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in d:\anaconda\lib\site-pa
ckages (from pandas->random-survival-forest) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in d:\anaconda\lib\site-packages (fr
om pandas->random-survival-forest) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in d:\anaconda\lib\site-packages
(from pandas->random-survival-forest) (2023.3)
Requirement already satisfied: dill>=0.3.6 in d:\anaconda\lib\site-packages (fro
m multiprocess->random-survival-forest) (0.3.6)
Requirement already satisfied: threadpoolctl>=3.1.0 in d:\anaconda\lib\site-pack
ages (from scikit-learn->random-survival-forest) (3.5.0)
Requirement already satisfied: interface-meta>=1.2.0 in d:\anaconda\lib\site-pac
kages (from formulaic>=0.2.2->lifelines->random-survival-forest) (1.3.0)
Requirement already satisfied: typing-extensions>=4.2.0 in d:\anaconda\lib\site-
packages (from formulaic>=0.2.2->lifelines->random-survival-forest) (4.7.1)
Requirement already satisfied: wrapt>=1.0 in d:\anaconda\lib\site-packages (from
formulaic>=0.2.2->lifelines->random-survival-forest) (1.14.1)
Requirement already satisfied: contourpy>=1.0.1 in d:\anaconda\lib\site-packages
(from matplotlib>=3.0->lifelines->random-survival-forest) (1.0.5)
Requirement already satisfied: cycler>=0.10 in d:\anaconda\lib\site-packages (fr
om matplotlib>=3.0->lifelines->random-survival-forest) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in d:\anaconda\lib\site-package
s (from matplotlib>=3.0->lifelines->random-survival-forest) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in d:\anaconda\lib\site-package
s (from matplotlib>=3.0->lifelines->random-survival-forest) (1.4.4)
Requirement already satisfied: packaging>=20.0 in d:\anaconda\lib\site-packages
(from matplotlib>=3.0->lifelines->random-survival-forest) (23.1)
Requirement already satisfied: pillow>=6.2.0 in d:\anaconda\lib\site-packages (f
rom matplotlib>=3.0->lifelines->random-survival-forest) (9.4.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in d:\anaconda\lib\site-pac
kages (from matplotlib>=3.0->lifelines->random-survival-forest) (3.0.9)
Requirement already satisfied: six>=1.5 in d:\anaconda\lib\site-packages (from p
ython-dateutil>=2.8.2->pandas->random-survival-forest) (1.16.0)
```

In [10]:
```
!pip install scikit-survival scikit-learn pandas
```

```
Requirement already satisfied: scikit-survival in d:\anaconda\lib\site-packages
(0.23.0)
Requirement already satisfied: scikit-learn in d:\anaconda\lib\site-packages
(1.5.1)
Requirement already satisfied: pandas in d:\anaconda\lib\site-packages (2.2.2)
Requirement already satisfied: ecos in d:\anaconda\lib\site-packages (from sciki
t-survival) (2.0.14)
Requirement already satisfied: joblib in d:\anaconda\lib\site-packages (from sci
kit-survival) (1.2.0)
Requirement already satisfied: numexpr in d:\anaconda\lib\site-packages (from sc
ikit-survival) (2.8.4)
Requirement already satisfied: numpy in d:\anaconda\lib\site-packages (from scik
it-survival) (1.24.3)
Requirement already satisfied: osqp!=0.6.0,!=0.6.1 in d:\anaconda\lib\site-packa
ges (from scikit-survival) (0.6.7.post1)
Requirement already satisfied: scipy>=1.3.2 in d:\anaconda\lib\site-packages (fr
om scikit-survival) (1.11.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in d:\anaconda\lib\site-pack
ages (from scikit-learn) (3.5.0)
Requirement already satisfied: python-dateutil>=2.8.2 in d:\anaconda\lib\site-pa
ckages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in d:\anaconda\lib\site-packages (fr
om pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in d:\anaconda\lib\site-packages
(from pandas) (2023.3)
Requirement already satisfied: qdldl in d:\anaconda\lib\site-packages (from osq
p!=0.6.0,!=0.6.1->scikit-survival) (0.1.7.post4)
Requirement already satisfied: six>=1.5 in d:\anaconda\lib\site-packages (from p
ython-dateutil>=2.8.2->pandas) (1.16.0)
```

In [11]:
```python
#Importa de librerías
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [19]:
```python
#Carga del dataset
df=pd.read_csv('data/Customer-Churn-Records.csv')
```

EXPLORACION DE DATOS

In [20]:
```python
#Visualización de datos con sus etiquetas originales
df.head()
```

Out[20]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balan
0	1	15634602	Hargrave	619	France	Female	42	2	0.
1	2	15647311	Hill	608	Spain	Female	41	1	83807.
2	3	15619304	Onio	502	France	Female	42	8	159660.
3	4	15701354	Boni	699	France	Female	39	1	0.
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.

In [21]:
```
#Descripción del dataset
df.describe()
```

Out[21]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	N
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	

In [22]:
```
#Renombramos las variables
df.rename({'RowNumber':'NroRegistro', 'CustomerId': 'IdCliente', 'Surname':'Nomb
```

In [23]:
```
#Información de tipos de datos de cada columna
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   NroRegistro              10000 non-null  int64
 1   IdCliente                10000 non-null  int64
 2   NombreCliente            10000 non-null  object
 3   ScoreCredito             10000 non-null  int64
 4   Pais                     10000 non-null  object
 5   Genero                   10000 non-null  object
 6   Edad                     10000 non-null  int64
 7   Antiguedad               10000 non-null  int64
 8   Saldo                    10000 non-null  float64
 9   NroDeProductos           10000 non-null  int64
 10  SiTieneTarjeta           10000 non-null  int64
 11  EsMiembroActivo          10000 non-null  int64
 12  SalarioEstimado          10000 non-null  float64
 13  Desertor                 10000 non-null  int64
 14  Reclamos                 10000 non-null  int64
 15  CalificacionSatisfaccion 10000 non-null  int64
 16  TipoTarjeta              10000 non-null  object
 17  PuntosGanados            10000 non-null  int64
dtypes: float64(2), int64(12), object(4)
memory usage: 1.4+ MB
```

In [24]:
```
#Agrupamos variablas cuantitativas y cualitativas
var_cuantitativas = df.select_dtypes('number').columns
var_cualitativas = df.select_dtypes('object').columns
```

In [25]:
```
#variables cuantitativas o numéricas
var_cuantitativas
```

Out[25]:
```
Index(['NroRegistro', 'IdCliente', 'ScoreCredito', 'Edad', 'Antiguedad',
       'Saldo', 'NroDeProductos', 'SiTieneTarjeta', 'EsMiembroActivo',
       'SalarioEstimado', 'Desertor', 'Reclamos', 'CalificacionSatisfaccion',
       'PuntosGanados'],
      dtype='object')
```

In [26]:
```
#variables cualitativas o categóricas
var_cualitativas
```

Out[26]:
```
Index(['NombreCliente', 'Pais', 'Genero', 'TipoTarjeta'], dtype='object')
```

El dataset contiene 10000 registros

18 datos o variables en total

14 variables numéricas

NroRegistro
IdCliente
ScoreCredito
Edad
Antiguedad
Saldo
NroDeProductos
SiTieneTarjeta
EsMiembroActivo
SalarioEstimado
Desertor
Reclamos
CalificacionSatisfaccion
PuntosGanados

4 variables categóricas

Pais
Genero
NombreCliente
TipoTarjeta

In [27]:
```
#Se obtiene el Total de registro por campo Desertor
conteo = df.groupby('Desertor').size()
print(conteo)
```

```
Desertor
0    7962
1    2038
dtype: int64
```

In [28]:
```
# Se explora totales de registros por las variables categóricas
#Genero
conteo = df.groupby('Genero').size()
print(conteo)
```

```
Genero
Female    4543
Male      5457
dtype: int64
```

In [29]:
```
#ScoreCredito
conteo = df.groupby('ScoreCredito').size()
print(conteo)
```

```
ScoreCredito
350      5
351      1
358      1
359      1
363      1
      ...
846      5
847      6
848      5
849      8
850    233
Length: 460, dtype: int64
```

In [30]:
```
#Pais
conteo= df.groupby('Pais').size()
print(conteo)
```

```
Pais
France     5014
Germany    2509
Spain      2477
dtype: int64
```

In [31]:
```
#TipoTarjeta
conteo= df.groupby('TipoTarjeta').size()
print(conteo)
```

```
TipoTarjeta
DIAMOND     2507
GOLD        2502
PLATINUM    2495
SILVER      2496
dtype: int64
```

In [32]:
```
#Edad
conteo= df.groupby('Edad').size()
print(conteo)
```

```
Edad
18    22
19    27
20    40
21    53
22    84
      ..
83     1
84     2
85     1
88     1
92     2
Length: 70, dtype: int64
```

In [33]:
```
#Antiguedad
conteo= df.groupby('Antiguedad').size()
print(conteo)
```

```
Antiguedad
0      413
1     1035
2     1048
3     1009
4      989
5     1012
6      967
7     1028
8     1025
9      984
10     490
dtype: int64
```

In [34]:
```
#Se realiza un análisis preliminar de los desertores que realizaron reclamos
#Total por Desertor-Reclamos
conteo = df[df['Desertor'] == 1].groupby('Reclamos').size()
print(conteo)
```

```
Reclamos
0       4
1    2034
dtype: int64
```

In [35]:
```
#Total por Desertor-Antiguedad
conteo = df[df['Desertor'] == 1].groupby('Antiguedad').size()
print(conteo)
```

```
Antiguedad
0      95
1     232
2     201
3     213
4     203
5     209
6     196
7     177
8     197
9     214
10    101
dtype: int64
```

In [36]:
```
#Total por Desertor-Genero
conteo = df[df['Desertor'] == 1].groupby('Genero').size()
print(conteo)
```

```
Genero
Female    1139
Male       899
dtype: int64
```

In [37]:
```
#Total por Desertor-Pais
conteo = df[df['Desertor'] == 1].groupby('Pais').size()
print(conteo)
```

```
Pais
France     811
Germany    814
Spain      413
dtype: int64
```

In [38]:
```
#Total por Desertor-TipoTarjeta
conteo = df[df['Desertor'] == 1].groupby('TipoTarjeta').size()
print(conteo)
```

```
TipoTarjeta
DIAMOND     546
GOLD        482
PLATINUM    508
SILVER      502
dtype: int64
```

In [39]:
```python
#Total por Desertor-Antiguedad
conteo = df[df['Desertor'] == 1].groupby('Antiguedad').size()
print(conteo)
```

```
Antiguedad
0      95
1     232
2     201
3     213
4     203
5     209
6     196
7     177
8     197
9     214
10    101
dtype: int64
```

LIMPIEZA

Revisión datos nulos y duplicación

In [40]:
```python
#Verificación de datos nulos
df.isna().sum()
```

Out[40]:
```
NroRegistro                0
IdCliente                  0
NombreCliente              0
ScoreCredito               0
Pais                       0
Genero                     0
Edad                       0
Antiguedad                 0
Saldo                      0
NroDeProductos             0
SiTieneTarjeta             0
EsMiembroActivo            0
SalarioEstimado            0
Desertor                   0
Reclamos                   0
CalificacionSatisfaccion   0
TipoTarjeta                0
PuntosGanados              0
dtype: int64
```

In [41]:
```python
#Verificación data duplicada
duplicados = df.duplicated()

# Verificar si hay duplicados en el dataset
if duplicados.any():
    print("El dataset tiene valores duplicados.")
else:
    print("No hay valores duplicados en el dataset.")
```

```
No hay valores duplicados en el dataset.
```

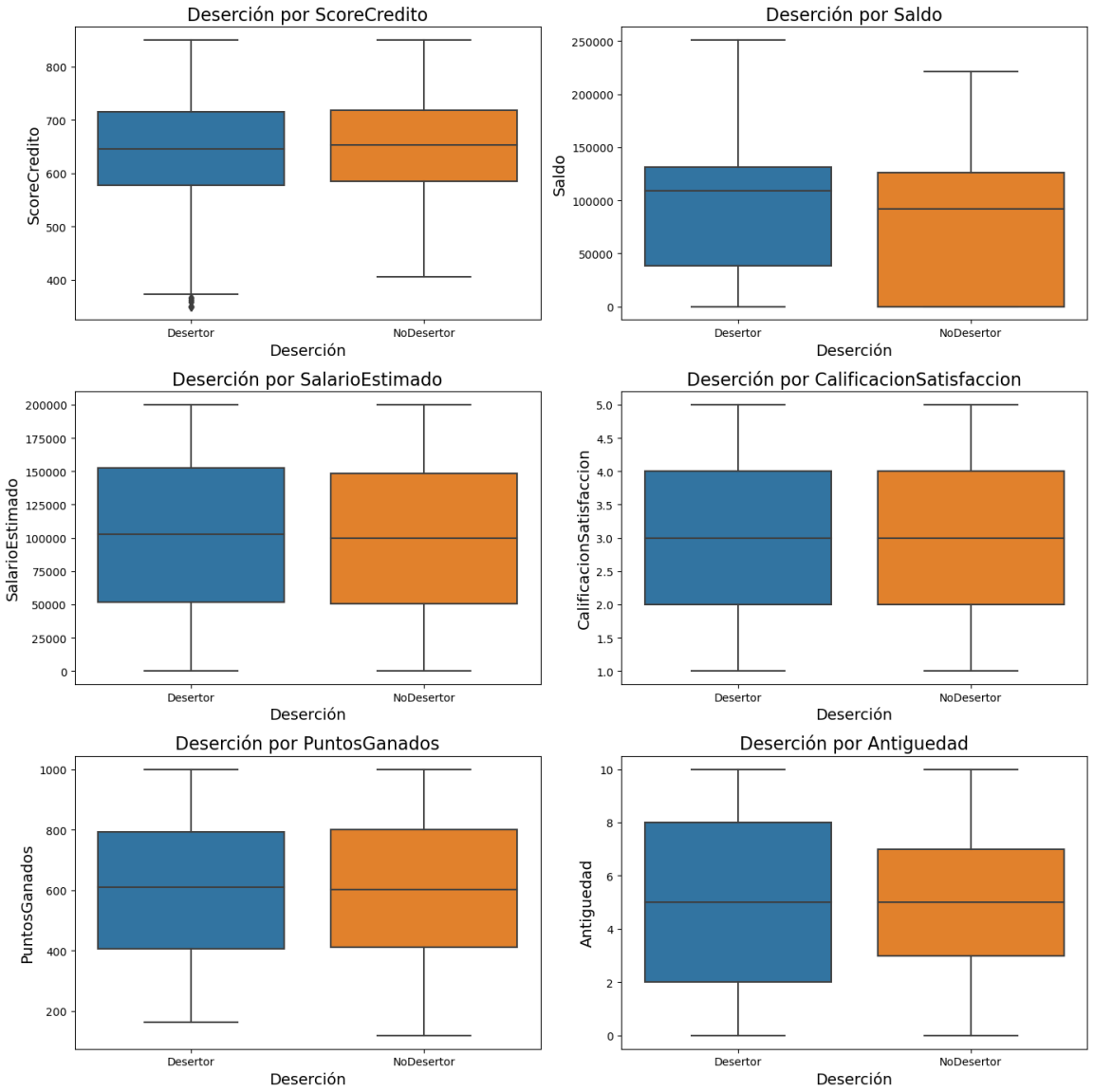
Eliminación variables no relevantes

In [42]:
```
#Se elimina variables no relevantes para continuar el análisis y posterior gener
df.drop(['NroRegistro','NombreCliente','IdCliente'], axis=1 , inplace=True)
```

PRE- PROCESAMIENTO

Revisión datos atípicos

In [43]:
```
# Crear una nueva columna con etiquetas descriptivas para Desertor
data1=df
data=df
data1['Desertor_Etiqueta'] = data['Desertor'].replace({0: 'NoDesertor', 1: 'Dese

# Crear el box plot entre Desertor_Etiqueta y CreditScore
fig, axs = plt.subplots(3, 2, figsize=(14,14))

# Primer gráfico en la cuadrícula (fila 0, columna 0)
plt1 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='ScoreCredito',data=data, ax=a
plt2 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='Saldo', data=data,ax = axs[0,
plt1 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='SalarioEstimado', data=data,a
plt2 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='CalificacionSatisfaccion',dat
plt1 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='PuntosGanados', data=data,ax
plt2 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='Antiguedad', data=data,ax = a
# Añadir títulos y etiquetas al gráfico
axs[0, 0].set_title('Deserción por ScoreCredito', fontsize=16)
axs[0, 0].set_ylabel('ScoreCredito', fontsize=14)
axs[0, 0].set_xlabel('Deserción', fontsize=14)

axs[0, 1].set_title('Deserción por Saldo', fontsize=16)
axs[0, 1].set_ylabel('Saldo', fontsize=14)
axs[0, 1].set_xlabel('Deserción', fontsize=14)

axs[1, 0].set_title('Deserción por SalarioEstimado', fontsize=16)
axs[1, 0].set_ylabel('SalarioEstimado', fontsize=14)
axs[1, 0].set_xlabel('Deserción', fontsize=14)

axs[1, 1].set_title('Deserción por CalificacionSatisfaccion', fontsize=16)
axs[1, 1].set_ylabel('CalificacionSatisfaccion', fontsize=14)
axs[1, 1].set_xlabel('Deserción', fontsize=14)

axs[2, 0].set_title('Deserción por PuntosGanados', fontsize=16)
axs[2, 0].set_ylabel('PuntosGanados', fontsize=14)
axs[2, 0].set_xlabel('Deserción', fontsize=14)

axs[2, 1].set_title('Deserción por Antiguedad', fontsize=16)
axs[2, 1].set_ylabel('Antiguedad', fontsize=14)
axs[2, 1].set_xlabel('Deserción', fontsize=14)

# Ajustar los márgenes entre gráficos
plt.tight_layout()

# Mostrar la gráfica
plt.show()
```



In [44]:
```
#Elimina variable temporal
df.drop(['Desertor_Etiqueta'], axis=1 , inplace=True)
```

Removemos los datos atípicos

In [45]:
```
# Calculamos el Quartil 1 y Quartil 3 que son aquellos que nos permiten estimar
Q1 = df.ScoreCredito.quantile(0.25)
Q3 = df.ScoreCredito.quantile(0.75)
IQR = Q3 - Q1  #rango intercuartil
# Ahora removemos aquellas observaciones que se encuentran por fuera del rango:
df = df[~((df['ScoreCredito'] < (Q1 - 1.5 * IQR)) |(df['ScoreCredito'] > (Q3 + 1
df.shape
```

Out[45]:
```
(9985, 15)
```

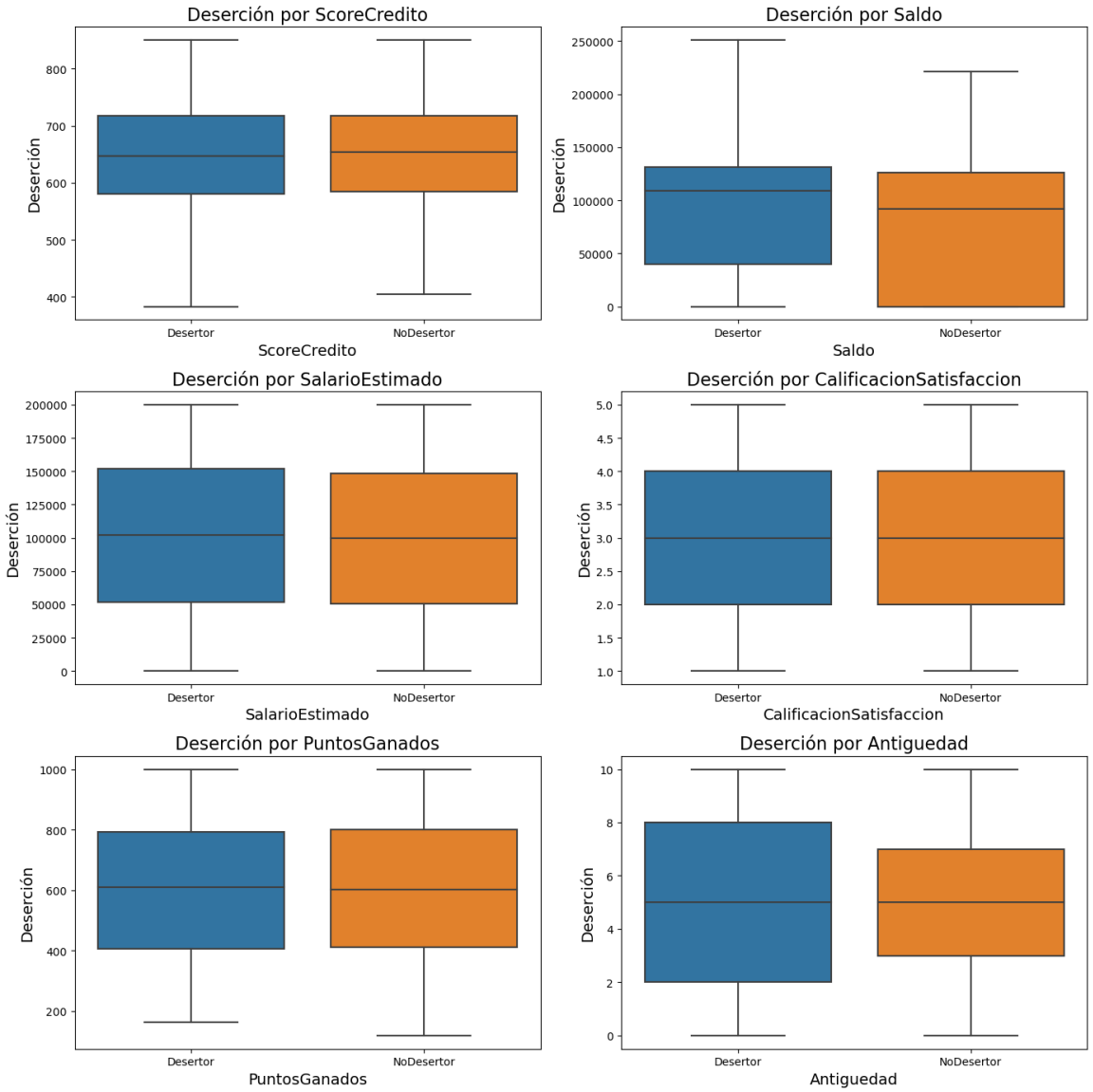
In [46]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9985 entries, 0 to 9999
Data columns (total 15 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ScoreCredito              9985 non-null   int64
 1   Pais                      9985 non-null   object
 2   Genero                    9985 non-null   object
 3   Edad                      9985 non-null   int64
 4   Antiguedad                9985 non-null   int64
 5   Saldo                     9985 non-null   float64
 6   NroDeProductos            9985 non-null   int64
 7   SiTieneTarjeta            9985 non-null   int64
 8   EsMiembroActivo           9985 non-null   int64
 9   SalarioEstimado           9985 non-null   float64
 10  Desertor                  9985 non-null   int64
 11  Reclamos                  9985 non-null   int64
 12  CalificacionSatisfaccion  9985 non-null   int64
 13  TipoTarjeta               9985 non-null   object
 14  PuntosGanados             9985 non-null   int64
dtypes: float64(2), int64(10), object(3)
memory usage: 1.2+ MB
```

Revisión sin datos atípicos

In [47]:
```
# Crear una nueva columna con etiquetas descriptivas para Exited
data1=df
data1['Desertor_Etiqueta'] = data['Desertor'].replace({0: 'NoDesertor', 1: 'Dese

# Crear el box plot entre Desertor_Etiqueta y CreditScore
fig, axs = plt.subplots(3, 2, figsize=(14,14))

# Primer gráfico en la cuadrícula (fila 0, columna 0)
plt1 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='ScoreCredito',data=data, ax=a
plt2 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='Saldo', data=data,ax = axs[0,
plt1 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='SalarioEstimado', data=data,a
plt2 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='CalificacionSatisfaccion',dat
plt1 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='PuntosGanados', data=data,ax
plt2 = sns.boxplot(x=data1['Desertor_Etiqueta'],y='Antiguedad', data=data,ax = a
# Añadir títulos y etiquetas al gráfico
axs[0, 0].set_title('Deserción por ScoreCredito', fontsize=16)
axs[0, 0].set_xlabel('ScoreCredito', fontsize=14)
axs[0, 0].set_ylabel('Deserción', fontsize=14)

axs[0, 1].set_title('Deserción por Saldo', fontsize=16)
axs[0, 1].set_xlabel('Saldo', fontsize=14)
axs[0, 1].set_ylabel('Deserción', fontsize=14)

axs[1, 0].set_title('Deserción por SalarioEstimado', fontsize=16)
axs[1, 0].set_xlabel('SalarioEstimado', fontsize=14)
axs[1, 0].set_ylabel('Deserción', fontsize=14)

axs[1, 1].set_title('Deserción por CalificacionSatisfaccion', fontsize=16)
axs[1, 1].set_xlabel('CalificacionSatisfaccion', fontsize=14)
axs[1, 1].set_ylabel('Deserción', fontsize=14)

axs[2, 0].set_title('Deserción por PuntosGanados', fontsize=16)
axs[2, 0].set_xlabel('PuntosGanados', fontsize=14)
axs[2, 0].set_ylabel('Deserción', fontsize=14)

axs[2, 1].set_title('Deserción por Antiguedad', fontsize=16)
axs[2, 1].set_xlabel('Antiguedad', fontsize=14)
axs[2, 1].set_ylabel('Deserción', fontsize=14)

# Ajustar los márgenes entre gráficos
plt.tight_layout()

# Mostrar la gráfica
plt.show()
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\3630601790.py:3: SettingWithCo
pyWarning:
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/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1['Desertor_Etiqueta'] = data['Desertor'].replace({0: 'NoDesertor', 1: 'De
sertor'})
```



TRANSFORMACION

Generación Rangos de Edad

In [48]:
```
#Se crea rangos de edad para agruparlos
df['Rangos Edad'] = pd.cut(df['Edad'], bins=[18, 30, 40, 50,60,61], labels=['18-
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\586881333.py:2: SettingWithCop
yWarning:
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/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Rangos Edad'] = pd.cut(df['Edad'], bins=[18, 30, 40, 50,60,61], labels=['1
8-30', '31-40', '41-50', '51-60', '61+'])
```

Generación Rangos Puntos Ganados

In [49]:
```
#Se crea rangos Puntos Ganados
df['Rangos PuntosGanados'] = pd.cut(df['PuntosGanados'],bins = [ 0, 200, 400, 60
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\3805444535.py:2: SettingWithCo
pyWarning:
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/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Rangos PuntosGanados'] = pd.cut(df['PuntosGanados'],bins = [ 0, 200, 400,
600, 800, 1000], labels = ['0-200', '200-400', '400-600', '600-800', '800+'])
```

Generación Rangos Score Credito

In [50]:
```
#Se crea rangos de Score Credito
df['Rangos ScoreCredito'] = pd.cut(df['ScoreCredito'],bins = [ 0, 200, 400, 600,
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\2291809383.py:2: SettingWithCo
pyWarning:
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/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Rangos ScoreCredito'] = pd.cut(df['ScoreCredito'],bins = [ 0, 200, 400, 60
0, 800, 1000], labels = ['0-200', '200-400', '400-600', '600-800', '800+'])
```

Generación Rangos Saldos

In [51]:
```
#Se crea rangos de Saldos
df['Rangos Saldos'] = pd.cut(df['Saldo'],bins = [ 0, 50000, 100000, 150000, 2000
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\1851152082.py:2: SettingWithCo
pyWarning:
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/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Rangos Saldos'] = pd.cut(df['Saldo'],bins = [ 0, 50000, 100000, 150000, 20
0000, 300000], labels = ['0-50K', '50K-100K', '100K-150K', '150K-200K','200K+'])
```

Generación Rangos Salario Estimado

In [52]:
```
#Se crea rangos de SalarioEstimado
df['Rangos SalarioEstimado'] = pd.cut(df['SalarioEstimado'],bins = [ 0, 25000, 5
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\1793399288.py:2: SettingWithCo
pyWarning:
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/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['Rangos SalarioEstimado'] = pd.cut(df['SalarioEstimado'],bins = [ 0, 25000,
50000, 75000, 100000, 125000, 150000, 175000, 200000], labels = ['0-25K', '25K-5
0K', '50K-75K', '75K-100K','100K-125K','125K-150K','150K-175K', '175K+'])
```

In [53]:
```
df
```

Out[53]:

	ScoreCredito	Pais	Genero	Edad	Antiguedad	Saldo	NroDeProductos	SiTiene1
0	619	France	Female	42	2	0.00	1	
1	608	Spain	Female	41	1	83807.86	1	
2	502	France	Female	42	8	159660.80	3	
3	699	France	Female	39	1	0.00	2	
4	850	Spain	Female	43	2	125510.82	1	
...	...	...	...	...	...	...	...	
9995	771	France	Male	39	5	0.00	2	
9996	516	France	Male	35	10	57369.61	1	
9997	709	France	Female	36	7	0.00	1	
9998	772	Germany	Male	42	3	75075.31	2	
9999	792	France	Female	28	4	130142.79	1	

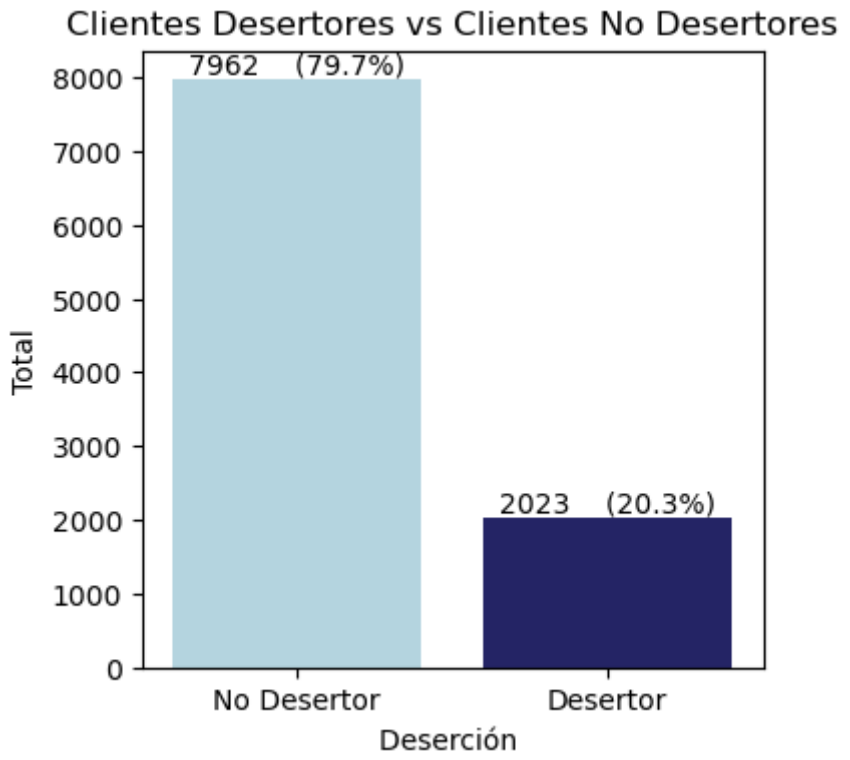
9985 rows × 21 columns

ANÁLISIS DESCRIPTIVO

ANÁLISIS DE DISTRIBUCION DE DESERCIÓN

In [54]:

```
data_rsf = df

churn_count = df['Desertor'].value_counts()
fig, ax = plt.subplots(figsize=(4, 4))
color_palette = sns.color_palette()

sns.barplot(
    x=churn_count.index,
    y=churn_count.values,
    palette=['#ADD8E6', '#191970']
).set(
    xticks=range(2),
    xticklabels=["No Desertor", "Desertor"],
    xlabel='Deserción ',
    ylabel='Total',
    title='Clientes Desertores vs Clientes No Desertores'
)

# Añadir etiquetas con cantidad y porcentaje
total_count = sum(churn_count.values)
for i, count in enumerate(churn_count.values):
    percentage = round(count / total_count * 100, 1)
    ax.text(i, count + 5, f'{count}   '+ f' ({percentage}%)', ha='center', va='b

plt.show()
```



In [55]:
```
#Elimino variables de análisis descriptivo
df.drop(['Desertor_Etiqueta'], axis=1 , inplace=True)
```

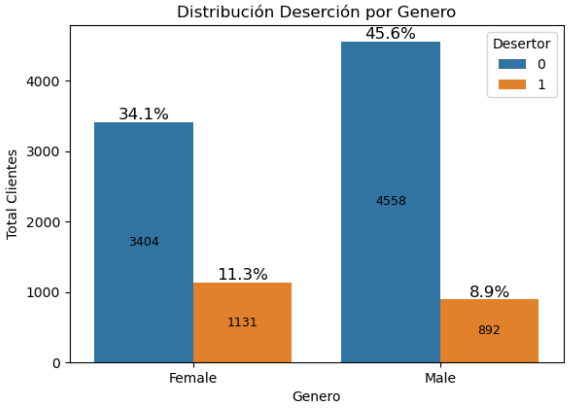
```
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\2997450132.py:2: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.drop(['Desertor_Etiqueta'], axis=1 , inplace=True)
```

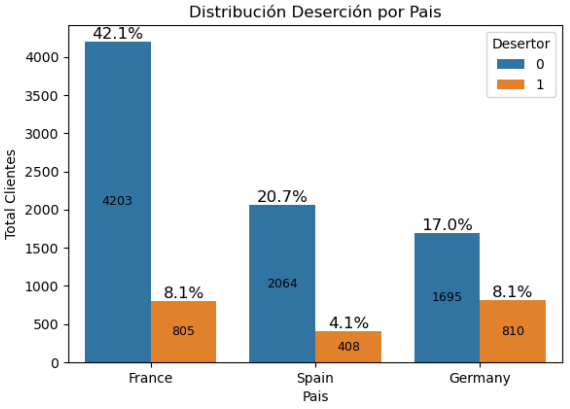
In [56]:
```
df.info()
```

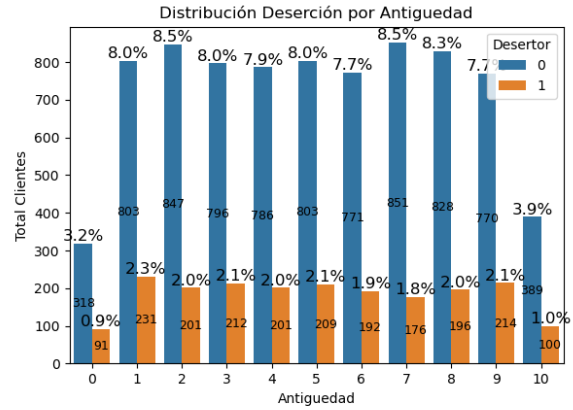
```
<class 'pandas.core.frame.DataFrame'>
Index: 9985 entries, 0 to 9999
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ScoreCredito              9985 non-null   int64
 1   Pais                      9985 non-null   object
 2   Genero                    9985 non-null   object
 3   Edad                      9985 non-null   int64
 4   Antiguedad                9985 non-null   int64
 5   Saldo                     9985 non-null   float64
 6   NroDeProductos            9985 non-null   int64
 7   SiTieneTarjeta            9985 non-null   int64
 8   EsMiembroActivo           9985 non-null   int64
 9   SalarioEstimado           9985 non-null   float64
 10  Desertor                  9985 non-null   int64
 11  Reclamos                  9985 non-null   int64
 12  CalificacionSatisfaccion  9985 non-null   int64
 13  TipoTarjeta               9985 non-null   object
 14  PuntosGanados             9985 non-null   int64
 15  Rangos Edad               9552 non-null   category
 16  Rangos PuntosGanados      9985 non-null   category
 17  Rangos ScoreCredito       9985 non-null   category
 18  Rangos Saldos             6373 non-null   category
 19  Rangos SalarioEstimado    9985 non-null   category
dtypes: category(5), float64(2), int64(10), object(3)
memory usage: 1.5+ MB
```

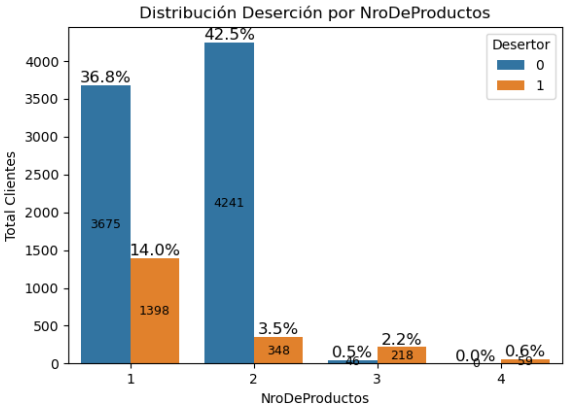
In [43]:

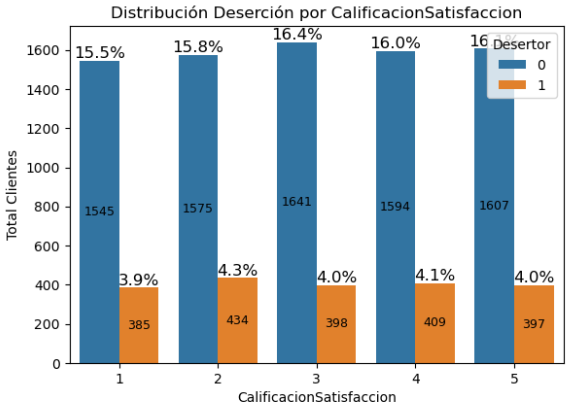
```
#countplot_list = df.select_dtypes(include=['object', 'category','number']).colu
#Revisión de distribución de deserción por variable
countplot_list = ['Genero','Pais', 'Antiguedad','NroDeProductos','CalificacionSa
                  'EsMiembroActivo','TipoTarjeta','SiTieneTarjeta','Rangos Edad'
                  'Rangos ScoreCredito','Rangos PuntosGanados','Rangos Saldos',
                  'Rangos SalarioEstimado','Reclamos']

plt.figure(figsize=(12,30))
for i, col in enumerate(countplot_list):
    plt.subplot(7, 2, i+1)
    ax = sns.countplot(data=df, x=col, hue='Desertor')
    ax.set_ylabel('Total Clientes')
    # Calcular porcentaje para cada contenedor
    total = len(df)  # número total de registros

    for container in ax.containers:
        labels_porcentaje = [f'{(v.get_height() / total) * 100:.1f}%' for v in c

        #count = int(v.get_height())
        labels_total=[f'{v.get_height():.0f}' for v in container]

        ax.bar_label(container, labels=labels_porcentaje, label_type='edge', fon
                     color='black')
        ax.bar_label(container, labels=labels_total, label_type='center', fontsi
                     color='black')

    plt.title(f'Distribución Deserción por {col}')

plt.tight_layout()
plt.xticks(rotation = 45)
plt.show()
```

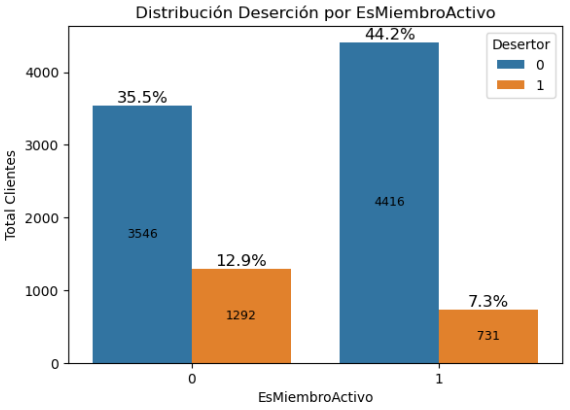

Distribución Deserción por Genero
Desertor
0
1
Total Clientes
Genero
Female
Male
34.1%
3404
11.3%
1131
45.6%
4558
8.9%
892

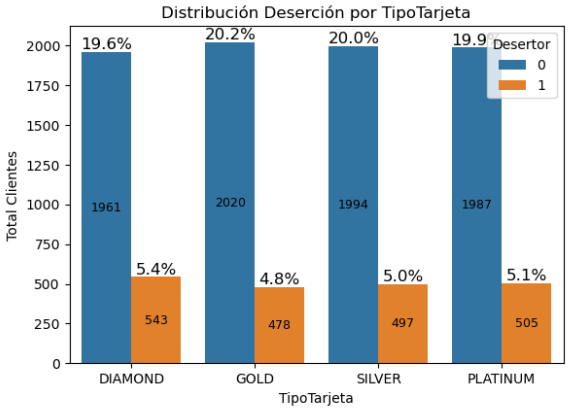

Distribución Deserción por Pais
Desertor
0
1
Total Clientes
Pais
France
Spain
Germany
42.1%
4203
8.1%
805
20.7%
2064
4.1%
408
17.0%
1695
8.1%
810

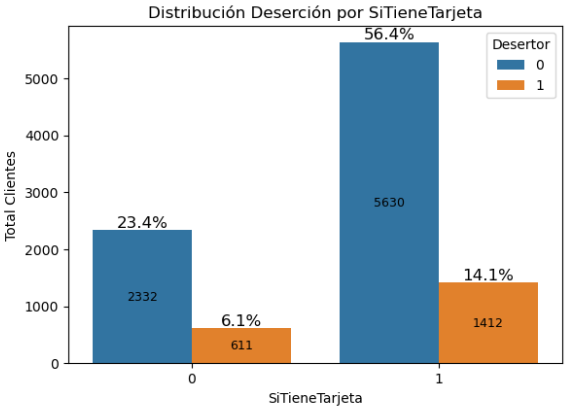

Distribución Deserción por Antiguedad
Desertor
0
1
Total Clientes
Antiguedad

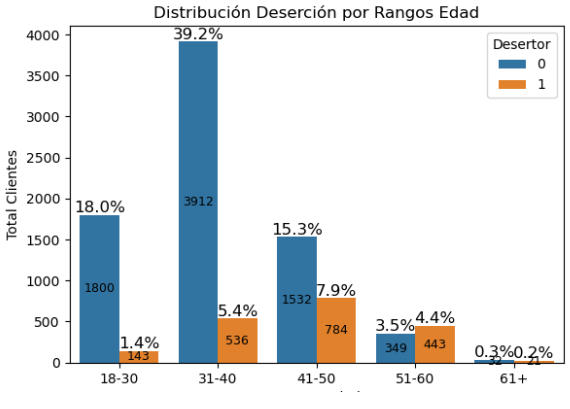

Distribución Deserción por NroDeProductos
Desertor
0
1
Total Clientes
NroDeProductos
36.8%
3675
14.0%
1398
42.5%
4241
3.5%
348
0.5%
2.2%
218
0.0%
0.6%

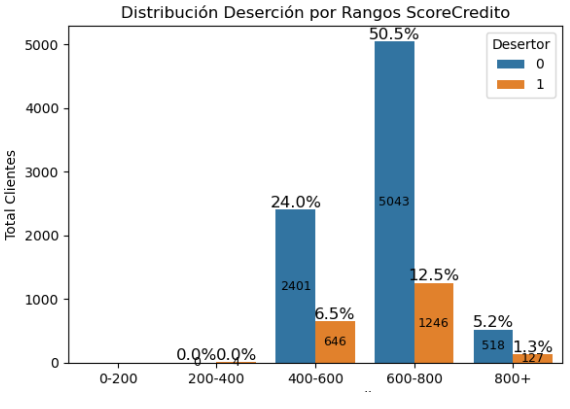

Distribución Deserción por CalificacionSatisfaccion
Desertor
0
1
Total Clientes
CalificacionSatisfaccion
15.5%
1545
3.9%
385
15.8%
1575
4.3%
434
16.4%
1641
4.0%
398
16.0%
1594
4.1%
409
1607
4.0%
397

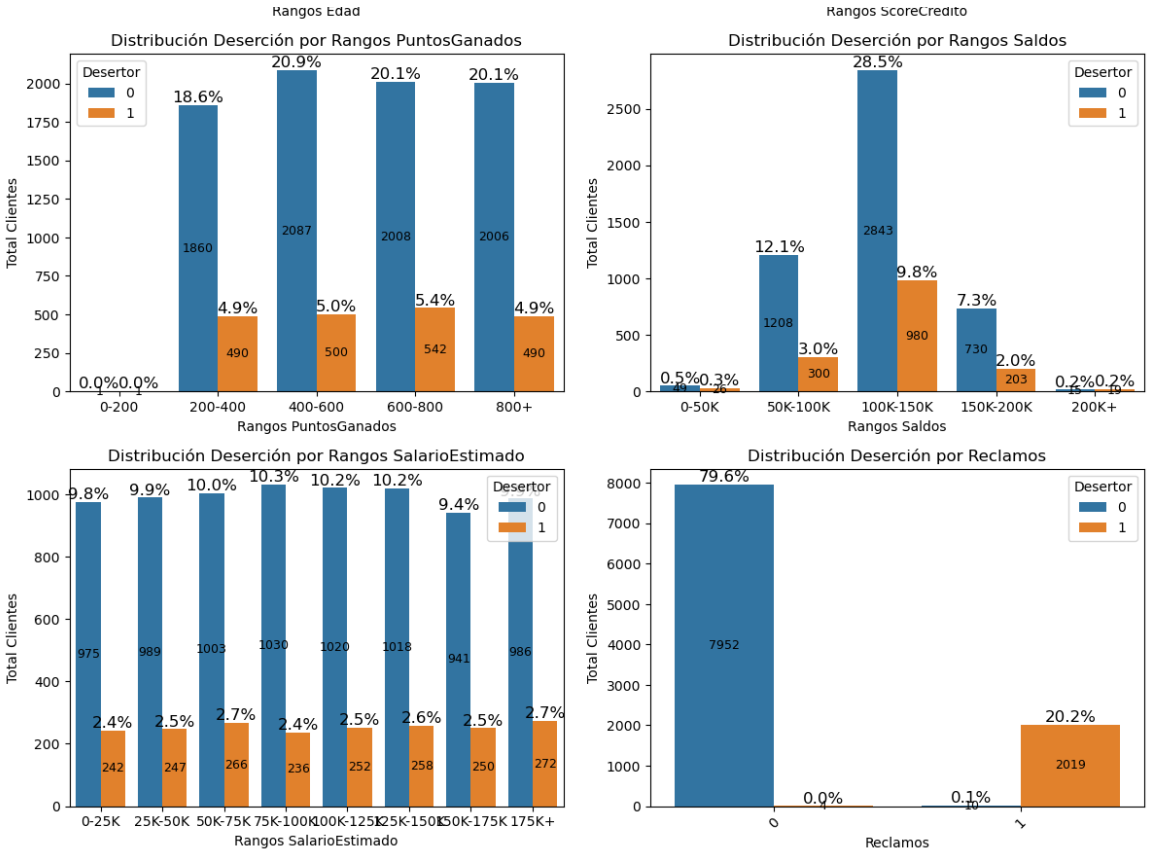

Distribución Deserción por EsMiembroActivo
Desertor
0
1
Total Clientes
EsMiembroActivo
35.5%
3546
12.9%
1292
44.2%
4416
7.3%
731


Distribución Deserción por TipoTarjeta
Desertor
0
1
Total Clientes
TipoTarjeta
DIAMOND
GOLD
SILVER
PLATINUM
19.6%
1961
5.4%
543
20.2%
2020
4.8%
478
20.0%
1994
5.0%
497
1987
5.1%
505


Distribución Deserción por SiTieneTarjeta
Desertor
0
1
Total Clientes
SiTieneTarjeta
23.4%
2332
6.1%
611
56.4%
5630
14.1%
1412


Distribución Deserción por Rangos Edad
Desertor
0
1
Total Clientes
18-30
31-40
41-50
51-60
61+
18.0%
1800
1.4%
143
39.2%
3912
5.4%
536
15.3%
1532
7.9%
784
3.5%
349
4.4%
443
0.3%
0.2%


Distribución Deserción por Rangos ScoreCredito
Desertor
0
1
Total Clientes
0-200
200-400
400-600
600-800
800+
0.0%
0.0%
24.0%
2401
6.5%
646
50.5%
5043
12.5%
1246
5.2%
518
1.3%
127



In [62]:
```
#Elimino variables de análisis descriptivo
df.drop(['Rangos Edad','Rangos Saldos','Rangos PuntosGanados', 'Rangos SalarioEs
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\862932869.py:2: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.drop(['Rangos Edad','Rangos Saldos','Rangos PuntosGanados', 'Rangos Salario
Estimado','Rangos ScoreCredito'], axis=1 , inplace=True)
```

In [63]:
```
from sklearn.preprocessing import LabelEncoder

# Crear una instancia de LabelEncoder
label_encoder = LabelEncoder()

# Aplicar LabelEncoder solo a columnas categóricas
for column in df.select_dtypes(include=['object']).columns:
    df[column] = label_encoder.fit_transform(df[column])
```

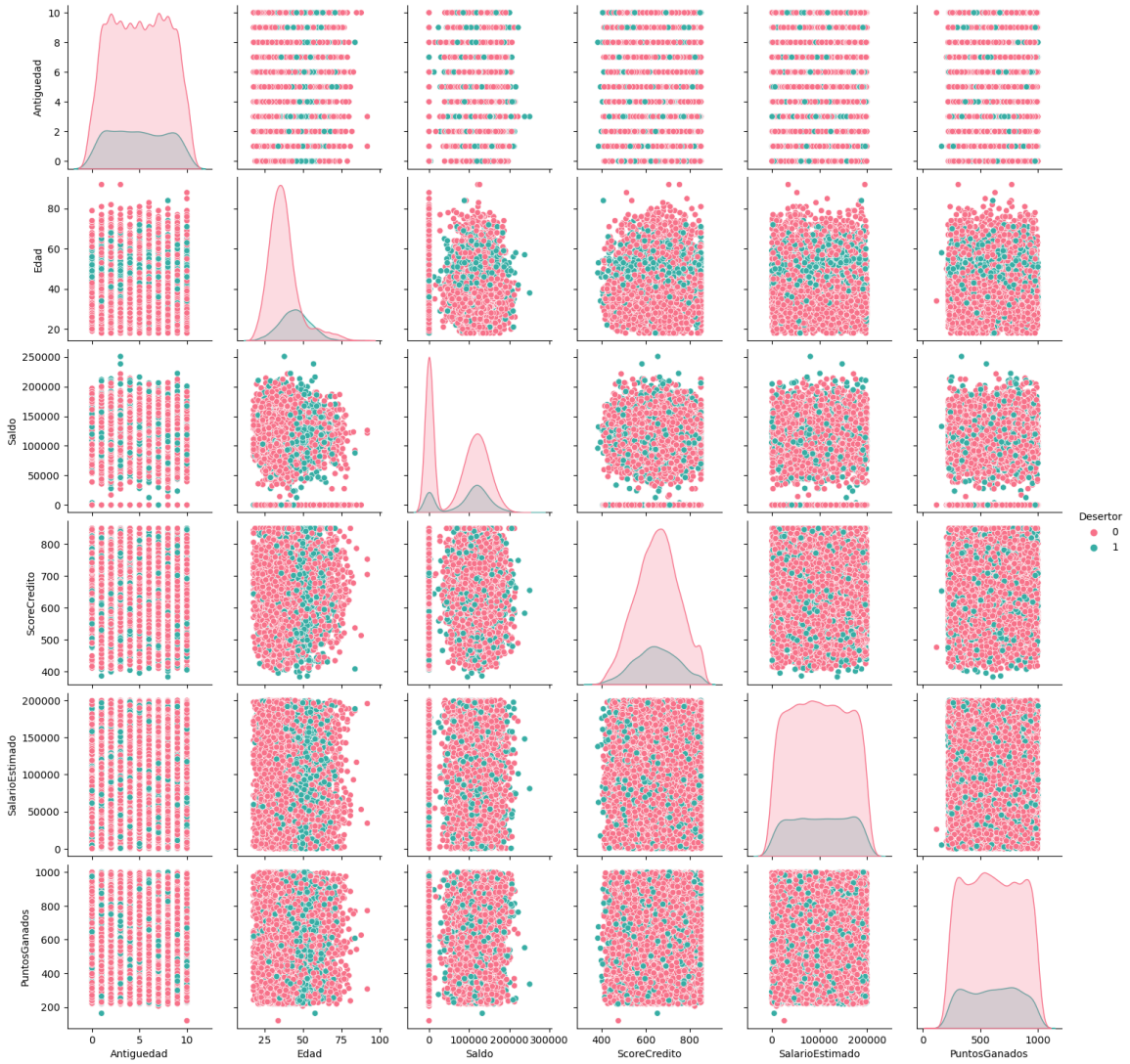
```
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\693833941.py:8: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column] = label_encoder.fit_transform(df[column])
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\693833941.py:8: SettingWithCop
yWarning:
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/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column] = label_encoder.fit_transform(df[column])
C:\Users\Admin\AppData\Local\Temp\ipykernel_17984\693833941.py:8: SettingWithCop
yWarning:
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/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[column] = label_encoder.fit_transform(df[column])
```

ANÁLISIS DE DISPERSIÓN

In [46]:

```
#Relación entre variables-gráfico
warnings.filterwarnings("ignore", category=FutureWarning)
plt.figure(figsize=(8, 14))
sns.pairplot(df[['Antiguedad','Edad', 'Saldo','ScoreCredito','SalarioEstimado','
plt.show()
```

```
D:\Anaconda\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure l
ayout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
<Figure size 800x1400 with 0 Axes>
```

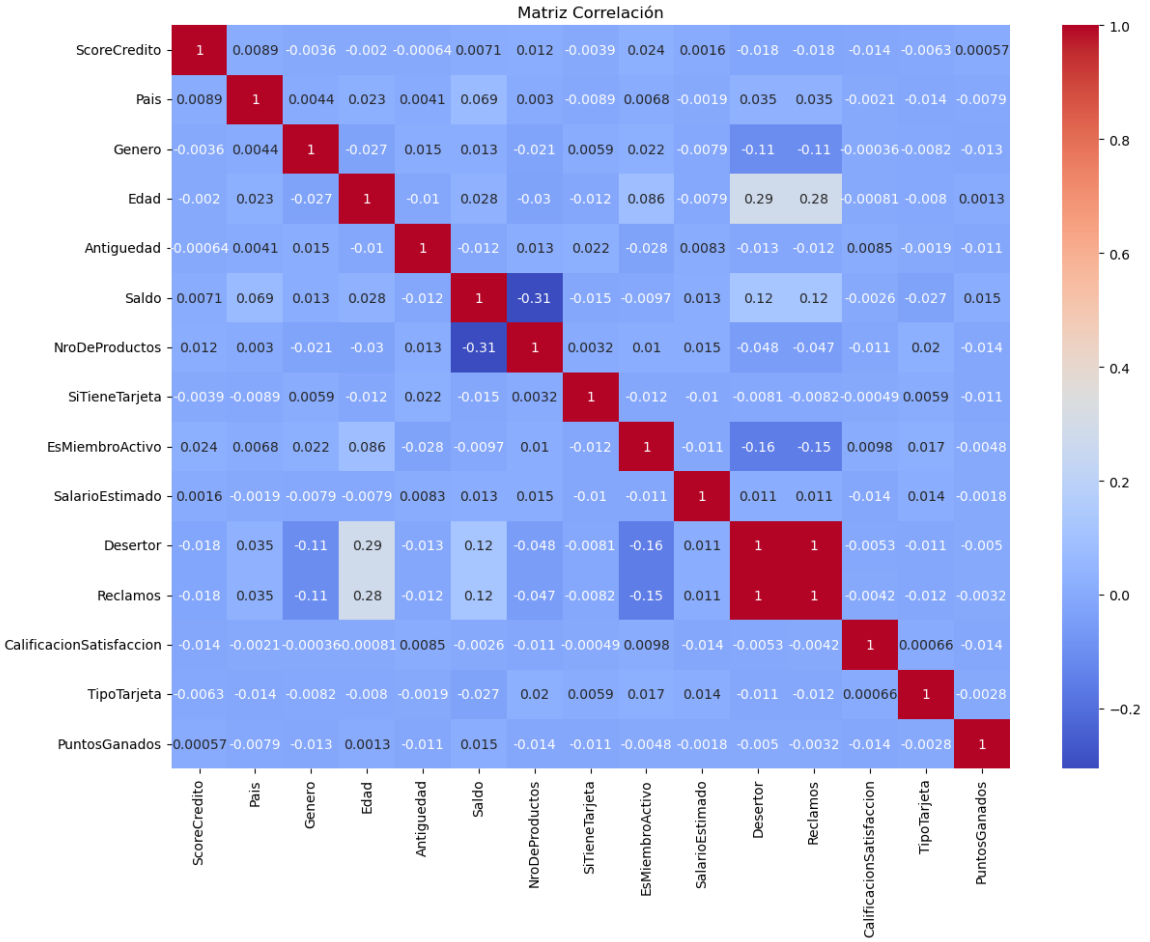


MATRIZ DE CORRELACION

In [47]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 9985 entries, 0 to 9999
Data columns (total 15 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   ScoreCredito              9985 non-null   int64  
 1   Pais                      9985 non-null   int32  
 2   Genero                    9985 non-null   int32  
 3   Edad                      9985 non-null   int64  
 4   Antiguedad                9985 non-null   int64  
 5   Saldo                     9985 non-null   float64
 6   NroDeProductos            9985 non-null   int64  
 7   SiTieneTarjeta            9985 non-null   int64  
 8   EsMiembroActivo           9985 non-null   int64  
 9   SalarioEstimado           9985 non-null   float64
 10  Desertor                  9985 non-null   int64  
 11  Reclamos                  9985 non-null   int64  
 12  CalificacionSatisfaccion  9985 non-null   int64  
 13  TipoTarjeta               9985 non-null   int32  
 14  PuntosGanados             9985 non-null   int64  
dtypes: float64(2), int32(3), int64(10)
memory usage: 1.4 MB
```

In [48]:
```
#Se genera la matriz de correlación con Reclamos
# Seleccionar todas las columnas numéricas
numeric_df = df.select_dtypes(include=['int32','int64', 'float64'])

matriz_correlacion =numeric_df.corr()
plt.figure(figsize=(14, 10))
sns.heatmap(matriz_correlacion, annot=True, cmap='coolwarm')
plt.title('Matriz Correlación')
plt.show()
```

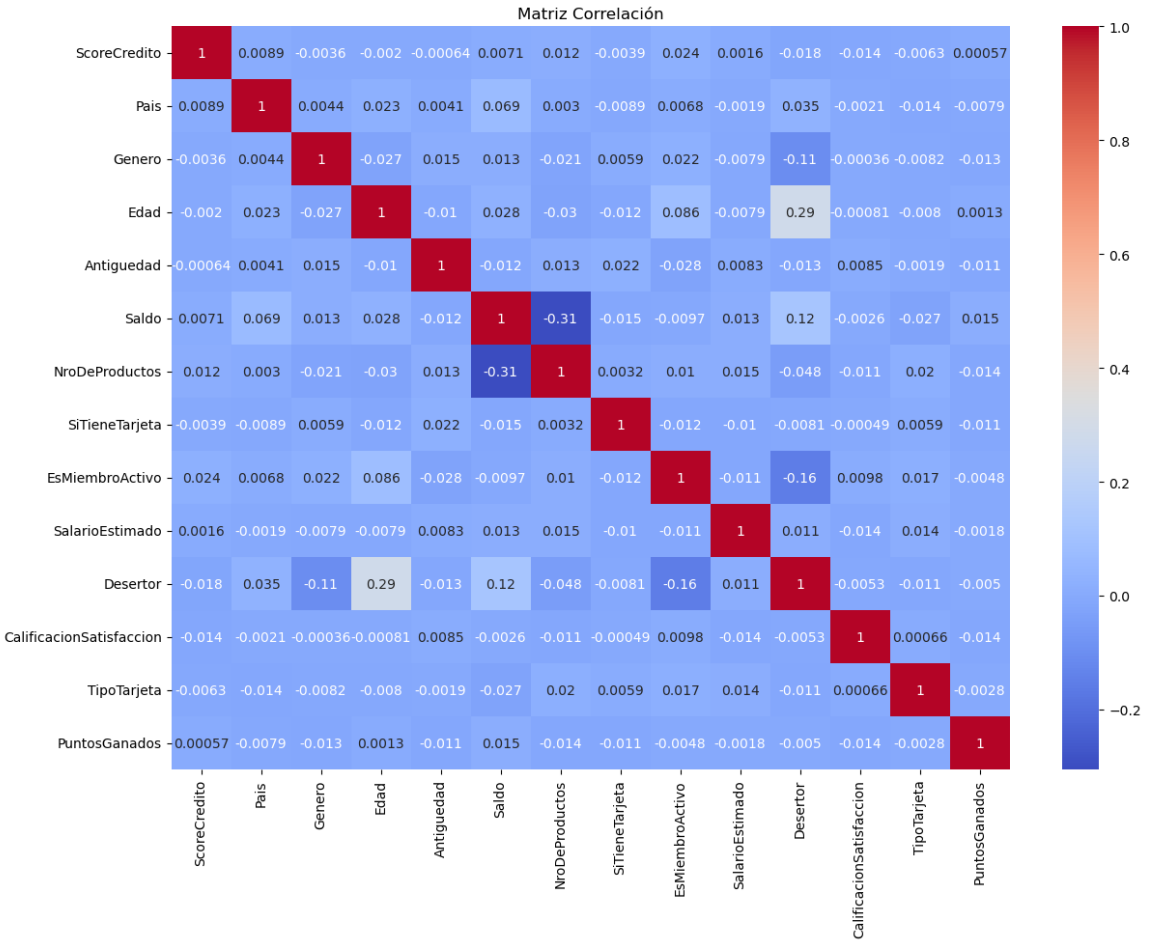


In [49]:
```
#Eliminamos la variable Reclamos por ser altamente correlacionada con Deserción
df.drop(['Reclamos'], axis=1 , inplace=True)
```

```
C:\Users\Admin\AppData\Local\Temp\ipykernel_9276\2826562776.py:2: SettingWithCop
yWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  df.drop(['Reclamos'], axis=1 , inplace=True)
```

RECLAMOS alta colineadad, produce sobreajuste

In [50]:
```
#Se genera la matriz de correlación sin Reclamos
matriz_correlacion = numeric_df.drop(columns=['Reclamos']).corr()
plt.figure(figsize=(14, 10))
sns.heatmap(matriz_correlacion, annot=True, cmap='coolwarm')
plt.title('Matriz Correlación')
plt.show()
```



La matriz de correlación revela varias relaciones entre las variables del conjunto de datos, las cuales se indican a continuación.

• Edad y Desertor: Existe una correlación positiva considerable (0.29), lo que indica que los clientes de mayor edad tienden a desertar de la entidad financiera con mayor frecuencia.

• Saldo y Desertor: Hay una correlación positiva (0.12), sugiriendo que los clientes con mayores saldos tienden a desertar de la entidad financiera en mayor proporción.

• Es Miembro Activo y Desertor: Existe una correlación negativa moderada (-0.16), lo que sugiere que los miembros activos tienen menos probabilidades de desertar de la entidad financiera.

Las correlaciones de la matriz indican que las variables como la edad, el saldo, y si es miembro activo tienen mayor impacto en el análisis de deserción de este dataset.

MODELAMIENTO

MODELOS DE CLASIFICACIÓN : REGRESIÓN LOGÍSTICA, ARBOLES DE DECISIÓN, RANDOM FOREST

MÉTRICAS

Evaluaremos con las métricas: **Recall,Precisión,F1-score,Accuracy**

In [58]:
```
#Librerías
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,accuracy_score, precision_scor
from sklearn import tree
from sklearn import model_selection
```

In [59]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
```

In [60]:
```
# Definir variables predictoras (X) y dependientes (y)
X=df.drop(columns=['Desertor'])
y=df['Desertor']
```

In [54]:
```
# Dividir en conjunto de entrenamiento y test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

In [55]:
```
# Crear y entrenar el modelo
model_log = LogisticRegression()
model_tree = DecisionTreeClassifier(random_state = 42)
model_rfo = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [56]:
```
#Entrenamos el modelo con los datos de entranamiento
model_log.fit(X_train,y_train)
model_tree.fit(X_train,y_train)
model_rfo.fit(X_train, y_train)
```

```
D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  n_iter_i = _check_optimize_result(
```

Out[56]:
```
▾ RandomForestClassifier i ?

RandomForestClassifier(random_state=42)
```

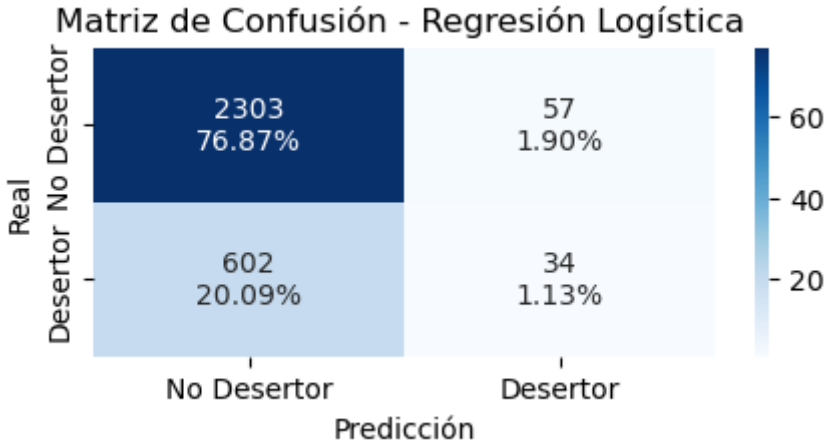
In [57]:
```
#Score modelos
trainlog_score = model_log.score(X_train, y_train)
print(f"Train Regresion Logística Score: {trainlog_score}")
testlog_score = model_log.score(X_test, y_test)
print(f"Test Regresion Logística Score : {testlog_score}")

traintree_score = model_tree.score(X_train, y_train)
print(f"Train Arbol Decisión Score     : {traintree_score}")
testtree_score = model_tree.score(X_test, y_test)
print(f"Test Arbol Decisión Score      : {testtree_score}")

trainrfo_score = model_rfo.score(X_train, y_train)
print(f"Train Random Forest Score      : {trainrfo_score}")
testrfo_score = model_rfo.score(X_test, y_test)
print(f"Test Random Forest Score       : {testrfo_score}")
```

```
Train Regresion Logística Score: 0.7942481041636857
Test Regresion Logística Score : 0.7800400534045394
Train Arbol Decisión Score     : 1.0
Test Arbol Decisión Score      : 0.7767022696929239
Train Random Forest Score      : 1.0
Test Random Forest Score       : 0.8477970627503337
```

In [58]:
```python
# Hacer predicciones en el conjunto de prueba
y_predlog = model_log.predict(X_test)
y_predlog_proba = model_log.predict_proba(X_test)[:, 1]

y_predtree = model_tree.predict(X_test)
y_predtree_proba = model_tree.predict_proba(X_test)[:, 1]

y_predrfo = model_rfo.predict(X_test)
y_predrfo_proba = model_rfo.predict_proba(X_test)[:, 1]
```

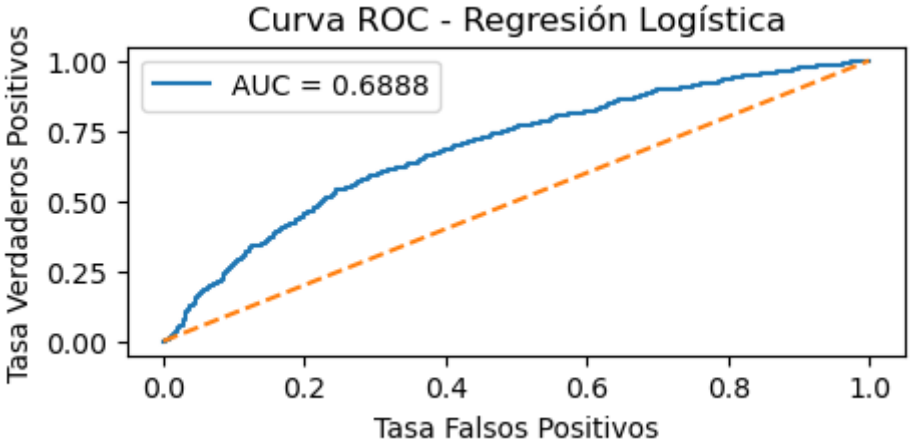
In [59]:
```python
# Matriz de confusión- Modelo Regresión Logística
print('Modelo Regresión Logística')
cm=confusion_matrix(y_test, y_predlog)
cm_percentcm = cm.astype('float') / cm.sum() * 100
labels = np.asarray([f'{v}\n{p:.2f}%' for v, p in zip(cm.flatten(),
                                                      cm_percentcm.flatten())]).
# Crear un gráfico de la matriz de confusión
plt.figure(figsize=(5, 2))  # Tamaño del gráfico
sns.heatmap(cm_percentcm, annot=labels , fmt='', cmap='Blues', cbar=True,
            annot_kws={"size": 10},
            xticklabels=['No Desertor', 'Desertor'],
            yticklabels=['No Desertor', 'Desertor'])

# Añadir etiquetas y título
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión - Regresión Logística ')

# Mostrar el gráfico
plt.show()
# Reporte de clasificación
print(classification_report(y_test, y_predlog, digits= 4))

# Calcular y mostrar el AUC-ROC
auc_log = roc_auc_score(y_test, y_predlog_proba)
print(f'AUC-ROC: {auc_log:.4f}')

# Graficar la curva ROC
fpr_log, tpr_log, thresholds = roc_curve(y_test, y_predlog_proba)
plt.figure(figsize=(5,2))
plt.plot(fpr_log, tpr_log, label=f'AUC = {auc_log:.4f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curva ROC - Regresión Logística')
plt.legend()
plt.show()
```

```
Modelo Regresión Logística
```



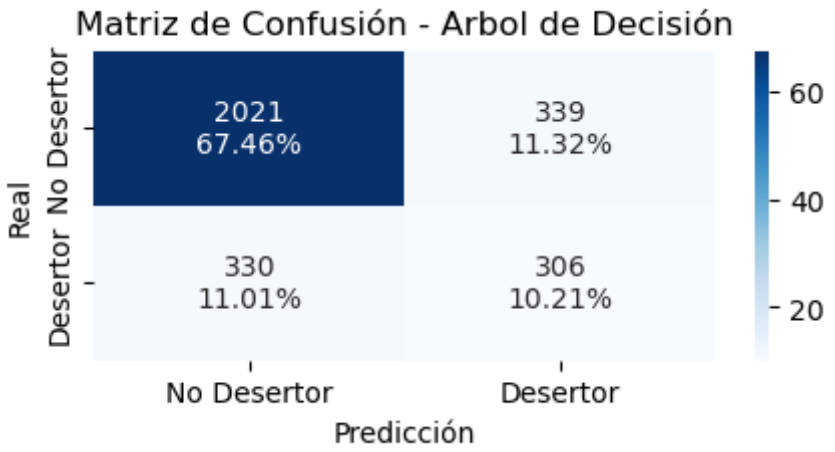
	precision	recall	f1-score	support
0	0.7928	0.9758	0.8748	2360
1	0.3736	0.0535	0.0935	636
accuracy			0.7800	2996
macro avg	0.5832	0.5147	0.4842	2996
weighted avg	0.7038	0.7800	0.7090	2996

AUC-ROC: 0.6888



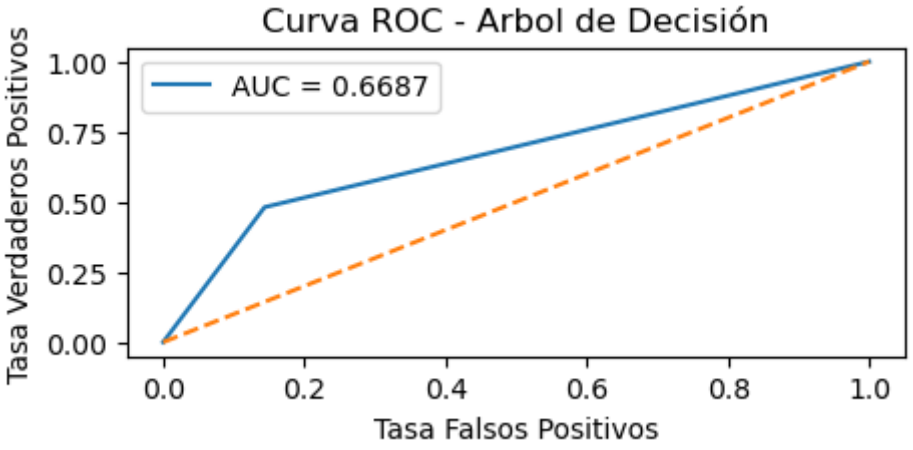
In [60]:
```
# Matriz de confusión- Modelo Arbol de Decisión
print('Modelo Arbol de Decisión')
cm=confusion_matrix(y_test, y_predtree)
cm_percentcm = cm.astype('float') / cm.sum() * 100
labels = np.asarray([f'{v}\n{p:.2f}%' for v, p in zip(cm.flatten(), cm_percentcm

# Crear un gráfico de la matriz de confusión
plt.figure(figsize=(5, 2))  # Tamaño del gráfico
sns.heatmap(cm_percentcm, annot=labels , fmt='', cmap='Blues', cbar=True,
            annot_kws={"size": 10},
            xticklabels=['No Desertor', 'Desertor'],
            yticklabels=['No Desertor', 'Desertor'])

# Añadir etiquetas y título
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión - Arbol de Decisión')

# Mostrar el gráfico
plt.show()


# Reporte de clasificación
print(classification_report(y_test, y_predtree, digits= 4))

# Calcular y mostrar el AUC-ROC
auc_tree = roc_auc_score(y_test, y_predtree_proba)
print(f'AUC-ROC: {auc_tree:.4f}')

# Graficar la curva ROC
fpr_tree, tpr_tree, thresholds = roc_curve(y_test, y_predtree_proba)
plt.figure(figsize=(5,2))
plt.plot(fpr_tree, tpr_tree, label=f'AUC = {auc_tree:.4f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curva ROC - Arbol de Decisión')
plt.legend()
plt.show()
```

```
Modelo Arbol de Decisión
```



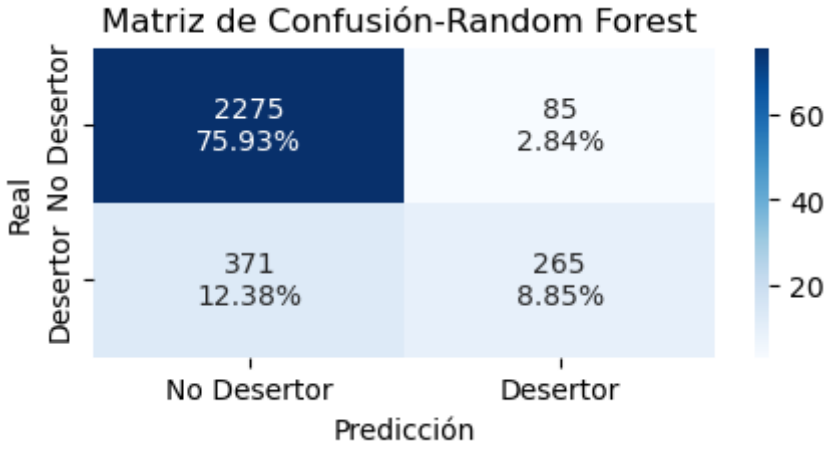
	precision	recall	f1-score	support
0	0.8596	0.8564	0.8580	2360
1	0.4744	0.4811	0.4778	636
accuracy			0.7767	2996
macro avg	0.6670	0.6687	0.6679	2996
weighted avg	0.7779	0.7767	0.7773	2996

AUC-ROC: 0.6687



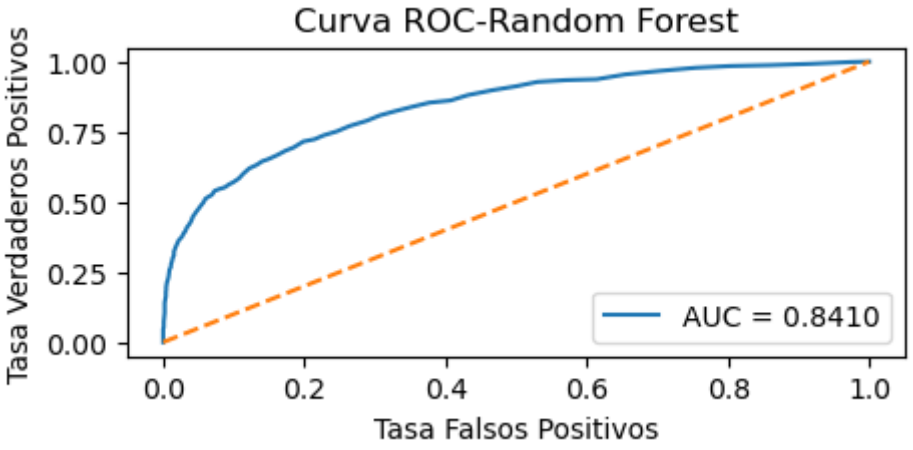
In [61]:
```
# Matriz de confusión- Modelo Random Forest
print('Modelo de Random Forest')
cm=confusion_matrix(y_test, y_predrfo)
cm_percentcm = cm.astype('float') / cm.sum() * 100
labels = np.asarray([f'{v}\n{p:.2f}%' for v, p in zip(cm.flatten(), cm_percentcm

# Crear un gráfico de la matriz de confusión
plt.figure(figsize=(5, 2))  # Tamaño del gráfico
sns.heatmap(cm_percentcm, annot=labels , fmt='', cmap='Blues', cbar=True,
            annot_kws={"size": 10},
            xticklabels=['No Desertor', 'Desertor'],
            yticklabels=['No Desertor', 'Desertor'])

# Añadir etiquetas y título
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión-Random Forest ')

# Mostrar el gráfico
plt.show()


# Reporte de clasificación
print(classification_report(y_test, y_predrfo, digits= 4))


# Calcular y mostrar el AUC-ROC
auc_rfo = roc_auc_score(y_test, y_predrfo_proba)
print(f'AUC-ROC: {auc_rfo:.4f}')


# Graficar la curva ROC
fpr_rfo, tpr_rfo, thresholds = roc_curve(y_test, y_predrfo_proba)
plt.figure(figsize=(5,2))
plt.plot(fpr_rfo, tpr_rfo, label=f'AUC = {auc_rfo:.4f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curva ROC-Random Forest')
plt.legend()
plt.show()
```

```
Modelo de Random Forest
```



	precision	recall	f1-score	support
0	0.8598	0.9640	0.9089	2360
1	0.7571	0.4167	0.5375	636
accuracy			0.8478	2996
macro avg	0.8085	0.6903	0.7232	2996
weighted avg	0.8380	0.8478	0.8301	2996

AUC-ROC: 0.8410



In [62]:
```
from sklearn.metrics import f1_score, recall_score, roc_auc_score, precision_sco

# Calcular las métricas para el modelo de Regresión Logística
results = {}
results['Modelo'] = 'Regresion Logística'
results['f1_score'] = f1_score(y_test, y_predlog, average='weighted')
results['recall'] = recall_score(y_test, y_predlog, average='weighted')
results['accuracy'] = roc_auc_score(y_test, y_predlog_proba)
results['precision'] = precision_score(y_test, y_predlog, average='weighted')

# Calcular las métricas para el modelo de Árbol de Decisión
results_ad = {}
results_ad['Modelo'] = 'Arbol Decisión'
results_ad['f1_score'] = f1_score(y_test, y_predtree, average='weighted')
results_ad['recall'] = recall_score(y_test, y_predtree, average='weighted')
results_ad['accuracy'] = roc_auc_score(y_test, y_predtree_proba)
results_ad['precision'] = precision_score(y_test, y_predtree, average='weighted'

# Calcular las métricas para el modelo de Bosque Aleatorio
results_rf = {}
results_rf['Modelo'] = 'Random Forest'
results_rf['f1_score'] = f1_score(y_test, y_predrfo, average='weighted')
results_rf['recall'] = recall_score(y_test, y_predrfo, average='weighted')
results_rf['accuracy'] = roc_auc_score(y_test, y_predrfo_proba)
results_rf['precision'] = precision_score(y_test, y_predrfo, average='weighted')

# Crear un DataFrame con los resultados de los modelos
metrics = pd.DataFrame([results, results_ad, results_rf], index=[0, 1, 2])
#metrics = pd.DataFrame([results_ad, results_rf], index=[0, 1])
metrics= metrics.round(4)

# Mostrar el DataFrame con las métricas
print(metrics)

# Transformar el DataFrame para que sea más fácil de graficar
dfm = metrics.melt('Modelo', var_name='Métrica', value_name='Valores')

# Configurar el estilo del gráfico
sns.set_theme(style='whitegrid', rc={'figure.figsize':(7, 5)})  # Ajusta el tama
```

```
                Modelo  f1_score  recall  accuracy  precision
0  Regresion Logística    0.7090  0.7800    0.6888     0.7038
1       Arbol Decisión    0.7773  0.7767    0.6687     0.7779
2        Random Forest    0.8301  0.8478    0.8410     0.8380
```

MODELAMIENTO CON BALANCEO RANDOM UNDER SAMPLER

In [63]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in d:\anaconda\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in d:\anaconda\lib\site-packages
(from imblearn) (0.12.3)
Requirement already satisfied: numpy>=1.17.3 in d:\anaconda\lib\site-packages (f
rom imbalanced-learn->imblearn) (1.24.3)
Requirement already satisfied: scipy>=1.5.0 in d:\anaconda\lib\site-packages (fr
om imbalanced-learn->imblearn) (1.11.1)
Requirement already satisfied: scikit-learn>=1.0.2 in d:\anaconda\lib\site-packa
ges (from imbalanced-learn->imblearn) (1.5.1)
Requirement already satisfied: joblib>=1.1.1 in d:\anaconda\lib\site-packages (f
rom imbalanced-learn->imblearn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\anaconda\lib\site-pack
ages (from imbalanced-learn->imblearn) (3.5.0)
```

In [64]:
```
from imblearn.under_sampling import RandomUnderSampler

# Aplicar UnderSampler solo al conjunto de entrenamiento
UnderSampler = RandomUnderSampler(random_state=42)
X_train_balanced, y_train_balanced = UnderSampler.fit_resample(X_train, y_train)

# Ver las proporciones después de aplicar
from collections import Counter
print("Distribución antes de RandomUnderSampler:", Counter(y_train))
print("Distribución después de RandomUnderSampler:", Counter(y_train_balanced))
```

```
Distribución antes de RandomUnderSampler: Counter({0: 5602, 1: 1387})
Distribución después de RandomUnderSampler: Counter({0: 1387, 1: 1387})
```

In [65]:
```
#Entrenamos el modelo con los datos de entranamiento balanceado
model_log.fit(X_train_balanced,y_train_balanced)
model_tree.fit(X_train_balanced,y_train_balanced)
model_rfo.fit(X_train_balanced, y_train_balanced)
```

```
D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:469: Convergence
Warning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressio
n
  n_iter_i = _check_optimize_result(
```

Out[65]:

▾ RandomForestClassifier i ?

```
RandomForestClassifier(random_state=42)
```

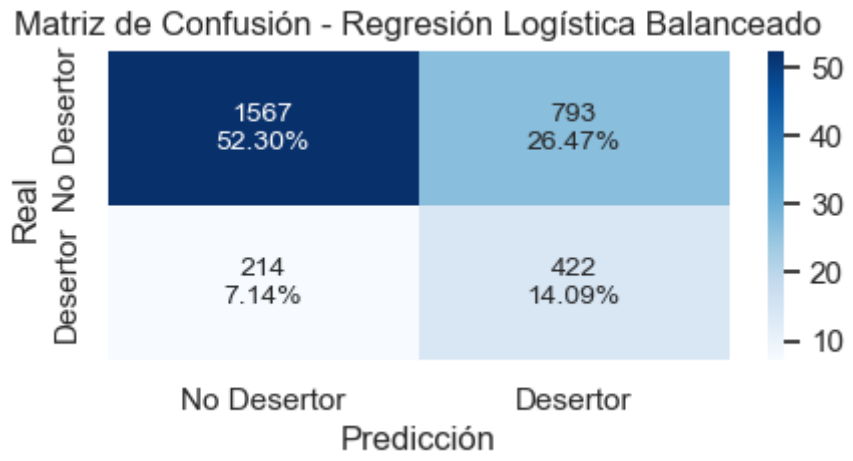
In [66]:
```
#Score modelos balanceados
trainlog_score = model_log.score(X_train_balanced, y_train_balanced)
print(f"Train Regresion Logística Balanceado Score: {trainlog_score}")
testlog_score = model_log.score(X_test, y_test)
print(f"Test Regresion Logística Score : {testlog_score}")

traintree_score = model_tree.score(X_train_balanced, y_train_balanced)
print(f"Train Arbol Decisión Balanceado Score     : {traintree_score}")
testtree_score = model_tree.score(X_test, y_test)
print(f"Test Arbol Decisión Score      : {testtree_score}")

trainrfo_score = model_rfo.score(X_train_balanced, y_train_balanced)
print(f"Train Random Forest Balanceado Score      : {trainrfo_score}")
testrfo_score = model_rfo.score(X_test, y_test)
print(f"Test Random Forest Score       : {testrfo_score}")
```

```
Train Regresion Logística Balanceado Score: 0.6658255227108868
Test Regresion Logística Score : 0.6638851802403204
Train Arbol Decisión Balanceado Score     : 1.0
Test Arbol Decisión Score      : 0.6822429906542056
Train Random Forest Balanceado Score      : 1.0
Test Random Forest Score       : 0.7746995994659546
```

In [67]:
```python
# Hacer predicciones en el conjunto de prueba
y_predlog = model_log.predict(X_test)
y_predlog_proba = model_log.predict_proba(X_test)[:, 1]

y_predtree = model_tree.predict(X_test)
y_predtree_proba = model_tree.predict_proba(X_test)[:, 1]

y_predrfo = model_rfo.predict(X_test)
y_predrfo_proba = model_rfo.predict_proba(X_test)[:, 1]
```

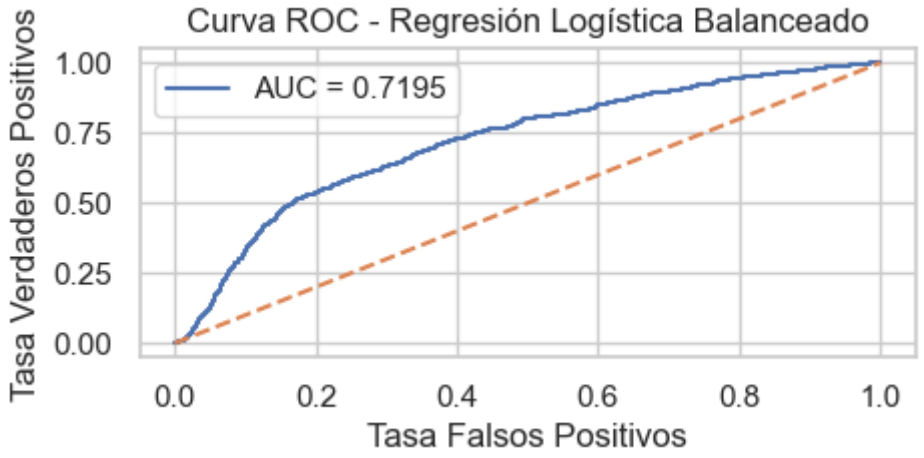
In [68]:
```python
# Matriz de confusión- Modelo Regresión Logística Balanceado
print('Modelo Regresión Logística Balanceado')
cm=confusion_matrix(y_test, y_predlog)
cm_percentcm = cm.astype('float') / cm.sum() * 100
labels = np.asarray([f'{v}\n{p:.2f}%' for v, p in zip(cm.flatten(), cm_percentcm

# Crear un gráfico de la matriz de confusión
plt.figure(figsize=(5, 2))  # Tamaño del gráfico
sns.heatmap(cm_percentcm, annot=labels , fmt='', cmap='Blues', cbar=True,
            annot_kws={"size": 10},
            xticklabels=['No Desertor', 'Desertor'],
            yticklabels=['No Desertor', 'Desertor'])

# Añadir etiquetas y título
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión - Regresión Logística Balanceado')

# Mostrar el gráfico
plt.show()


# Reporte de clasificación
print(classification_report(y_test, y_predlog, digits= 4))

# Calcular y mostrar el AUC-ROC
auc_log = roc_auc_score(y_test, y_predlog_proba)
print(f'AUC-ROC: {auc_log:.4f}')

# Graficar la curva ROC
fpr_log, tpr_log, thresholds = roc_curve(y_test, y_predlog_proba)
plt.figure(figsize=(5,2))
plt.plot(fpr_log, tpr_log, label=f'AUC = {auc_log:.4f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curva ROC - Regresión Logística Balanceado')
plt.legend()
plt.show()
```

```
Modelo Regresión Logística Balanceado
```



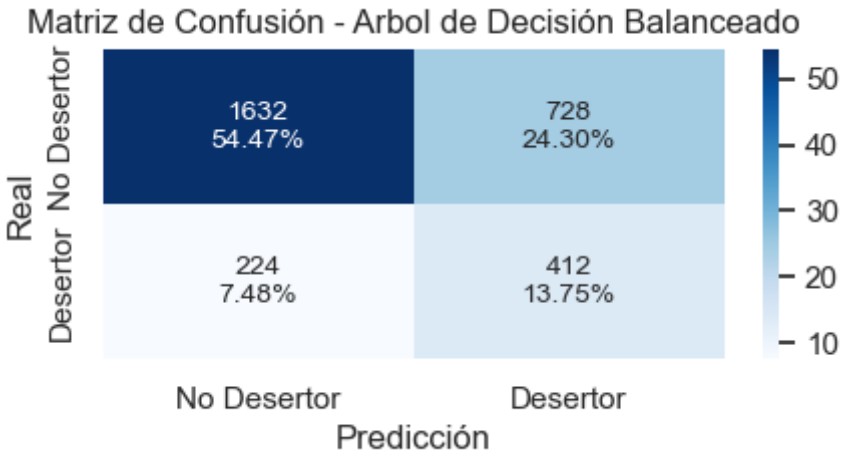
	precision	recall	f1-score	support
0	0.8798	0.6640	0.7568	2360
1	0.3473	0.6635	0.4560	636
accuracy			0.6639	2996
macro avg	0.6136	0.6638	0.6064	2996
weighted avg	0.7668	0.6639	0.6930	2996

AUC-ROC: 0.7195



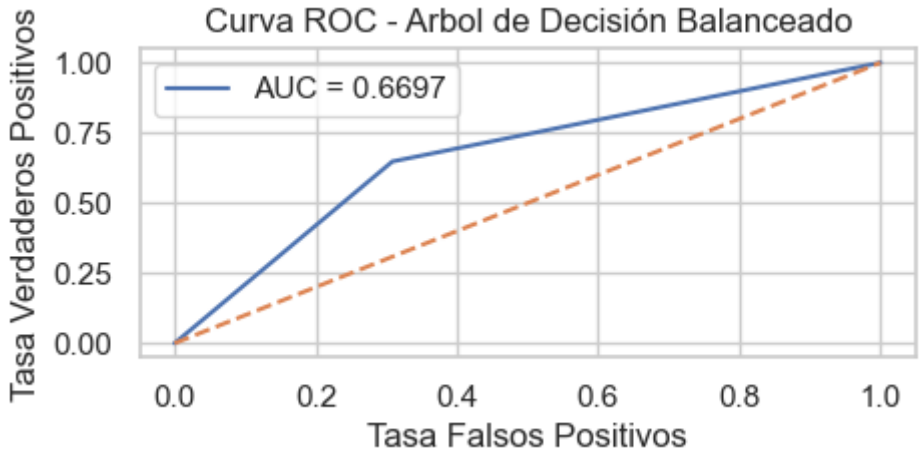
In [69]:
```
# Matriz de confusión- Modelo Arbol de Decisión Balanceado
print('Modelo Arbol de Decisión Balanceado')
cm=confusion_matrix(y_test, y_predtree)
cm_percentcm = cm.astype('float') / cm.sum() * 100
labels = np.asarray([f'{v}\n{p:.2f}%' for v, p in zip(cm.flatten(), cm_percentcm

# Crear un gráfico de la matriz de confusión
plt.figure(figsize=(5, 2))  # Tamaño del gráfico
sns.heatmap(cm_percentcm, annot=labels , fmt='', cmap='Blues', cbar=True,
            annot_kws={"size": 10},
            xticklabels=['No Desertor', 'Desertor'],
            yticklabels=['No Desertor', 'Desertor'])

# Añadir etiquetas y título
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión - Arbol de Decisión Balanceado')

# Mostrar el gráfico
plt.show()


# Reporte de clasificación
print(classification_report(y_test, y_predtree, digits= 4))

# Calcular y mostrar el AUC-ROC
auc_tree = roc_auc_score(y_test, y_predtree_proba)
print(f'AUC-ROC: {auc_tree:.4f}')

# Graficar la curva ROC
fpr_tree, tpr_tree, thresholds = roc_curve(y_test, y_predtree_proba)
plt.figure(figsize=(5,2))
plt.plot(fpr_tree, tpr_tree, label=f'AUC = {auc_tree:.4f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curva ROC - Arbol de Decisión Balanceado')
plt.legend()
plt.show()
```

```
Modelo Arbol de Decisión Balanceado
```



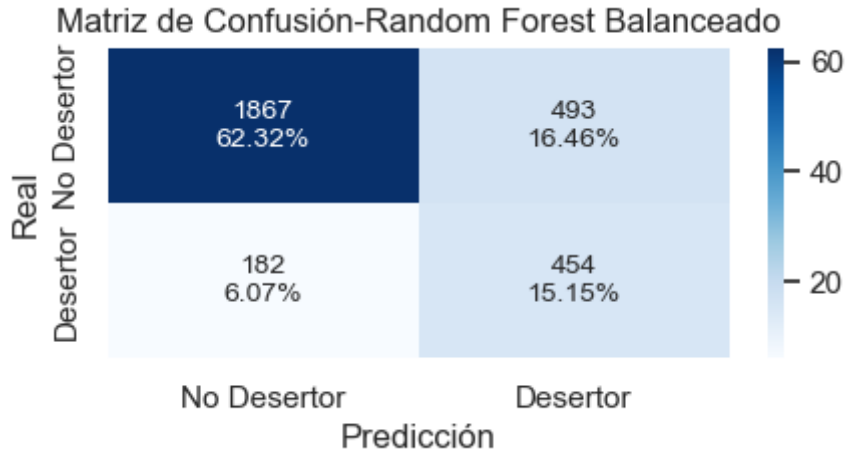
	precision	recall	f1-score	support
0	0.8793	0.6915	0.7742	2360
1	0.3614	0.6478	0.4640	636
accuracy			0.6822	2996
macro avg	0.6204	0.6697	0.6191	2996
weighted avg	0.7694	0.6822	0.7083	2996

AUC-ROC: 0.6697



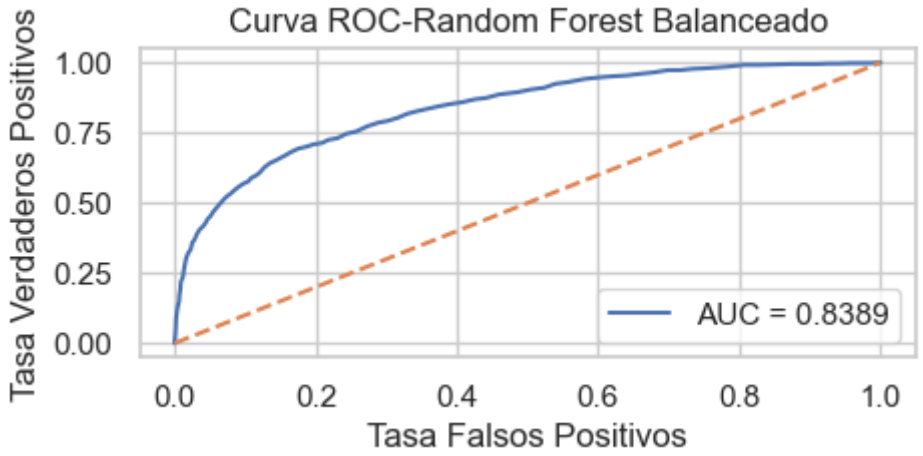
In [70]:
```
# Matriz de confusión- Modelo Random Forest Balanceado
print('Modelo de Random Forest Balanceado')
cm=confusion_matrix(y_test, y_predrfo)
cm_percentcm = cm.astype('float') / cm.sum() * 100
labels = np.asarray([f'{v}\n{p:.2f}%' for v, p in zip(cm.flatten(), cm_percentcm

# Crear un gráfico de la matriz de confusión
plt.figure(figsize=(5, 2))  # Tamaño del gráfico
sns.heatmap(cm_percentcm, annot=labels , fmt='', cmap='Blues', cbar=True,
            annot_kws={"size": 10},
            xticklabels=['No Desertor', 'Desertor'],
            yticklabels=['No Desertor', 'Desertor'])

# Añadir etiquetas y título
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión-Random Forest Balanceado')

# Mostrar el gráfico
plt.show()


# Reporte de clasificación
print(classification_report(y_test, y_predrfo, digits= 4))


# Calcular y mostrar el AUC-ROC
auc_rfo = roc_auc_score(y_test, y_predrfo_proba)
print(f'AUC-ROC: {auc_rfo:.4f}')


# Graficar la curva ROC
fpr_rfo, tpr_rfo, thresholds = roc_curve(y_test, y_predrfo_proba)
plt.figure(figsize=(5,2))
plt.plot(fpr_rfo, tpr_rfo, label=f'AUC = {auc_rfo:.4f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curva ROC-Random Forest Balanceado')
plt.legend()
plt.show()
```

```
Modelo de Random Forest Balanceado
```



	precision	recall	f1-score	support
0	0.9112	0.7911	0.8469	2360
1	0.4794	0.7138	0.5736	636
accuracy			0.7747	2996
macro avg	0.6953	0.7525	0.7102	2996
weighted avg	0.8195	0.7747	0.7889	2996

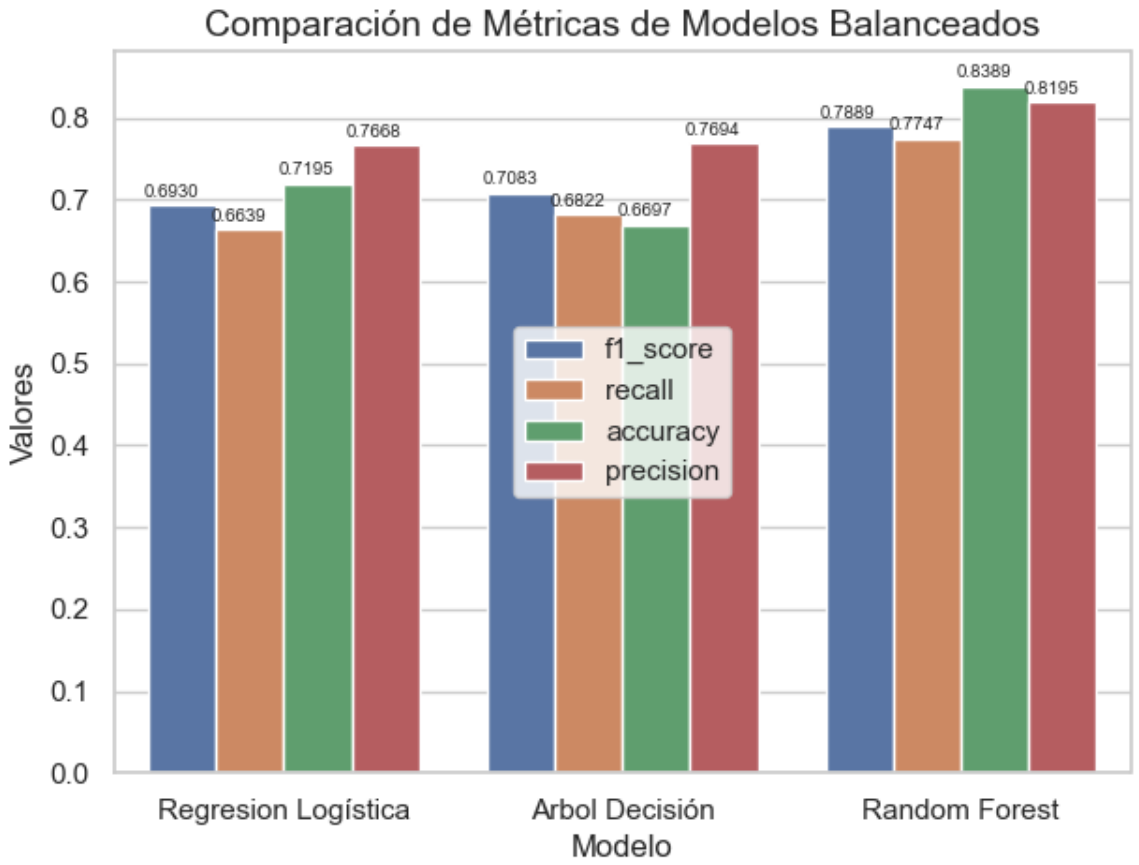
AUC-ROC: 0.8389



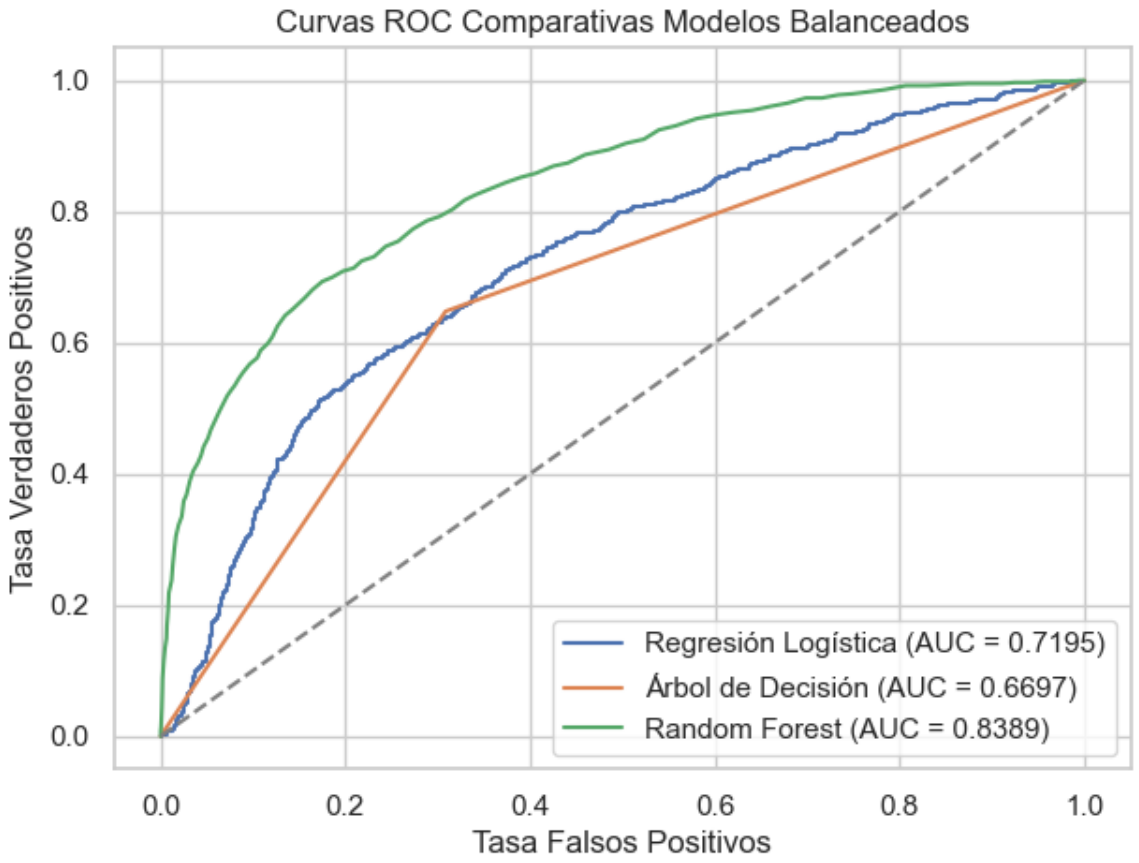
COMPARACION DE METRICAS ENTRE MODELOS REGRESION, ARBOL DE DECISION, RANDOM FOREST - BALANCEADOS

In [71]:
```
from sklearn.metrics import f1_score, recall_score, roc_auc_score, precision_sco

# Calcular las métricas para el modelo de Regresión Logística
results = {}
results['Modelo'] = 'Regresion Logística'
results['f1_score'] = f1_score(y_test, y_predlog, average='weighted')
results['recall'] = recall_score(y_test, y_predlog, average='weighted')
results['accuracy'] = roc_auc_score(y_test, y_predlog_proba)
results['precision'] = precision_score(y_test, y_predlog, average='weighted')

# Calcular las métricas para el modelo de Árbol de Decisión
results_ad = {}
results_ad['Modelo'] = 'Arbol Decisión'
results_ad['f1_score'] = f1_score(y_test, y_predtree, average='weighted')
results_ad['recall'] = recall_score(y_test, y_predtree, average='weighted')
results_ad['accuracy'] = roc_auc_score(y_test, y_predtree_proba)
results_ad['precision'] = precision_score(y_test, y_predtree, average='weighted'

# Calcular las métricas para el modelo de Bosque Aleatorio
results_rf = {}
results_rf['Modelo'] = 'Random Forest'
results_rf['f1_score'] = f1_score(y_test, y_predrfo, average='weighted')
results_rf['recall'] = recall_score(y_test, y_predrfo, average='weighted')
results_rf['accuracy'] = roc_auc_score(y_test, y_predrfo_proba)
results_rf['precision'] = precision_score(y_test, y_predrfo, average='weighted')

# Crear un DataFrame con los resultados de los modelos
metrics = pd.DataFrame([results, results_ad, results_rf], index=[0, 1, 2])
metrics= metrics.round(4)

# Mostrar el DataFrame con las métricas
print(metrics)

# Transformar el DataFrame para que sea más fácil de graficar
dfm = metrics.melt('Modelo', var_name='Métrica', value_name='Valores')

# Configurar el estilo del gráfico
sns.set_theme(style='whitegrid', rc={'figure.figsize':(7, 5)})  # Ajusta el tama

# Crear el gráfico de barras
ax = sns.barplot(data=dfm, x="Modelo", y="Valores", hue='Métrica', palette='deep


# Anotar los valores en las barras
for p in ax.patches:
    ax.annotate(f'{p.get_height():.4f}',
                (p.get_x() + p.get_width() / 3., p.get_height()),
                ha='center',va='center',
                xytext=(0,5),
                textcoords='offset points',
                fontsize=7)
#Título
plt.title('Comparación de Métricas de Modelos Balanceados', fontsize=14)

# Mostrar el gráfico
ax.legend(loc='center')
plt.show()
```

```
                Modelo  f1_score  recall  accuracy  precision
0  Regresion Logística    0.6930  0.6639    0.7195     0.7668
1       Arbol Decisión    0.7083  0.6822    0.6697     0.7694
2        Random Forest    0.7889  0.7747    0.8389     0.8195
```



In [72]:
```python
# Graficar las curvas ROC de los tres modelos
plt.figure(figsize=(7, 5))

# Curva ROC Regresión Logística
plt.plot(fpr_log, tpr_log, label=f'Regresión Logística (AUC = {auc_log:.4f})')

# Curva ROC Árbol de Decisión
plt.plot(fpr_tree, tpr_tree, label=f'Árbol de Decisión (AUC = {auc_tree:.4f})')

# Curva ROC Random Forest
plt.plot(fpr_rfo, tpr_rfo, label=f'Random Forest (AUC = {auc_rfo:.4f})')

# Línea diagonal de referencia (clasificador aleatorio)
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

# Etiquetas y título
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curvas ROC Comparativas Modelos Balanceados')

# Leyenda y mostrar gráfico
plt.legend()
plt.show()
```



APLICACION DE HIPERPARAMETROS - GRIDSEARCHCV PARA MEJORAMIENTO DE MODELO

In [73]:
```
pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in d:\anaconda\lib\site-packages
(1.5.1)Note: you may need to restart the kernel to use updated packages.

Requirement already satisfied: numpy>=1.19.5 in d:\anaconda\lib\site-packages (f
rom scikit-learn) (1.24.3)
Requirement already satisfied: scipy>=1.6.0 in d:\anaconda\lib\site-packages (fr
om scikit-learn) (1.11.1)
Requirement already satisfied: joblib>=1.2.0 in d:\anaconda\lib\site-packages (f
rom scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in d:\anaconda\lib\site-pack
ages (from scikit-learn) (3.5.0)
```

In [74]:
```
from sklearn.model_selection import GridSearchCV
```

Hiperparametros Regresión Logística Balanceado

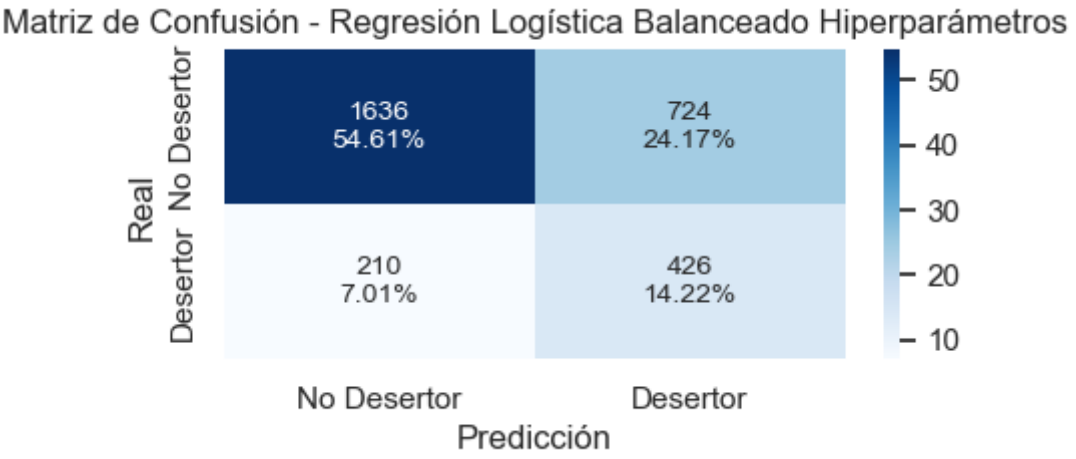
In [75]:

```
from sklearn.linear_model import LogisticRegression
#Hiperparametros
#'C'='1' #Representa un equilibrio moderado entre ajuste y regularización
#Solver='liblinear' #Se utiliza para conjunto de datos pequeños o medianos
#penalty='l2' #Se utiliza cuando un conjunto de datos con muchas característica

param_grid = {
    'C': [1],  # Diferentes valores para el parámetro de regularización
    'penalty': ['l1'],  # Tipos de penalización
    'solver': ['liblinear']  # Algoritmos de optimización que soportan l1 y l2
}

# Configurar GridSearchCV
grid_search_log = GridSearchCV(estimator=model_log, param_grid=param_grid,
                               cv=5, scoring='precision', verbose=1, n_jobs=-1)

grid_search_log.fit(X_train_balanced, y_train_balanced)

best_model_log= grid_search_log.best_estimator_
best_model_log


y_predlog = best_model_log.predict(X_test)
y_predlog_proba = best_model_log.predict_proba(X_test)[:, 1]
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
```

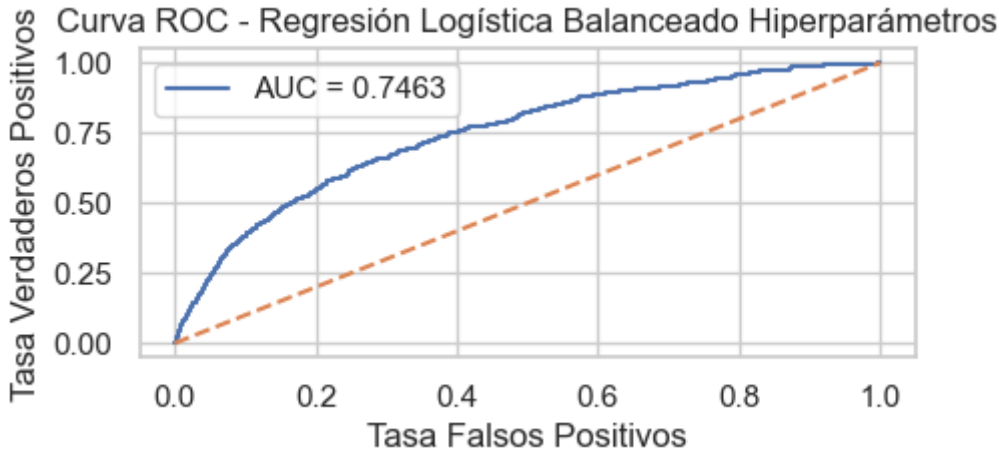
In [76]:
```
# Matriz de confusión- Modelo Regresión Logística Balanceado Hiperparámetro
print('Modelo Regresión Logística Balanceado Hiperparámretos')
cm=confusion_matrix(y_test, y_predlog)
cm_percentcm = cm.astype('float') / cm.sum() * 100
labels = np.asarray([f'{v}\n{p:.2f}%' for v, p in zip(cm.flatten(), cm_percentcm

# Crear un gráfico de la matriz de confusión
plt.figure(figsize=(5, 2))  # Tamaño del gráfico
sns.heatmap(cm_percentcm, annot=labels , fmt='', cmap='Blues', cbar=True,
            annot_kws={"size": 10},
            xticklabels=['No Desertor', 'Desertor'],
            yticklabels=['No Desertor', 'Desertor'])

# Añadir etiquetas y título
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión - Regresión Logística Balanceado Hiperparámetros'

# Mostrar el gráfico
plt.show()


# Reporte de clasificación
print(classification_report(y_test, y_predlog, digits= 4))

# Calcular y mostrar el AUC-ROC
auc_log = roc_auc_score(y_test, y_predlog_proba)
print(f'AUC-ROC: {auc_log:.4f}')

# Graficar la curva ROC
fpr_log, tpr_log, thresholds = roc_curve(y_test, y_predlog_proba)
plt.figure(figsize=(5,2))
plt.plot(fpr_log, tpr_log, label=f'AUC = {auc_log:.4f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curva ROC - Regresión Logística Balanceado Hiperparámetros')
plt.legend()
plt.show()
```

```
Modelo Regresión Logística Balanceado Hiperparámretos
```



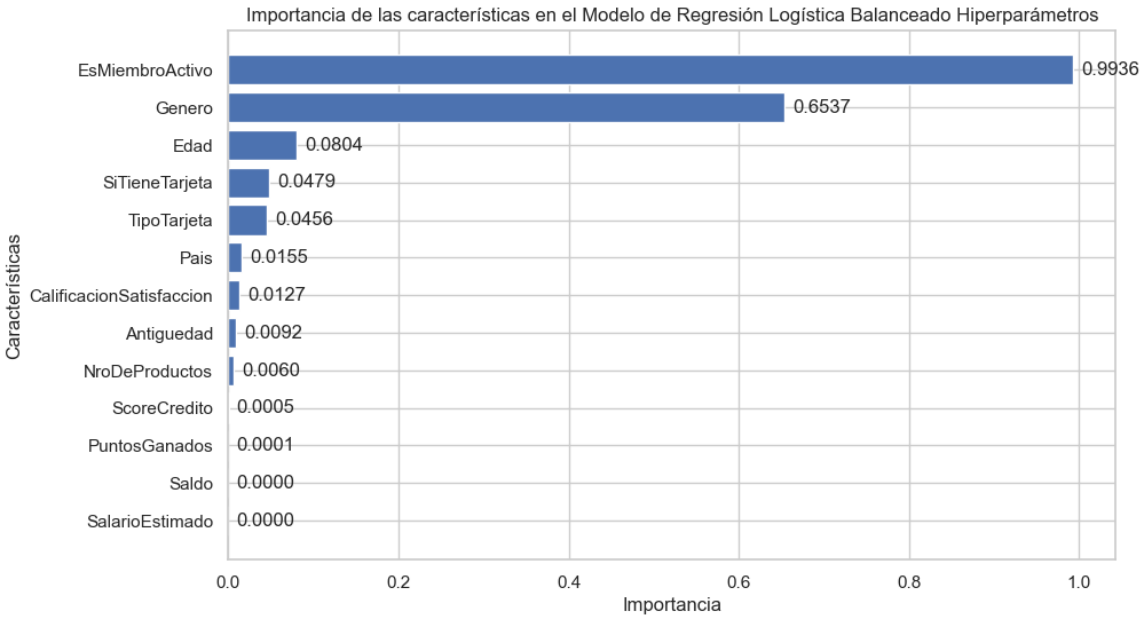
	precision	recall	f1-score	support
0	0.8862	0.6932	0.7779	2360
1	0.3704	0.6698	0.4770	636
accuracy			0.6883	2996
macro avg	0.6283	0.6815	0.6275	2996
weighted avg	0.7767	0.6883	0.7141	2996

AUC-ROC: 0.7463



Importancia de las Características Regresión Logística Balanceado Hiperparámetros

In [77]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Obtener la importancia de las características (los coeficientes)
feature_importance = np.abs(best_model_log.coef_[0])

# Crear un DataFrame
features = X.columns
importance_df = pd.DataFrame({'Feature': features, 'Importance': feature_importa

# Ordenar las características por importancia
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Graficar la importancia de las características
plt.figure(figsize=(10, 6))
bars = plt.barh(importance_df['Feature'], importance_df['Importance'])

# Añadir anotaciones a las barras (el valor total de cada barra)
for bar in bars:
    plt.annotate(f'{bar.get_width():.4f}',
                 xy=(bar.get_width(), bar.get_y() + bar.get_height() / 2),
                 xytext=(5, 0),  # Desplazamiento de la anotación (x, y)
                 textcoords="offset points",
                 ha='left', va='center')

# Etiquetas y título
plt.xlabel('Importancia')
plt.ylabel('Características')
plt.title('Importancia de las características en el Modelo de Regresión Logístic

# Invertir el eje y para que la característica más importante aparezca arriba
plt.gca().invert_yaxis()

# Mostrar el gráfico
plt.show()
```



Hiperparametros Arbol de Decisión Balanceado

In [78]:
```
from sklearn.tree import DecisionTreeClassifier

param_grid_tree={'max_depth':[None,10,20],'min_samples_split':[2,5,10]}
model_tree = DecisionTreeClassifier()
grid_search_tree = GridSearchCV(model_tree,param_grid_tree,cv=5, scoring='precis
grid_search_tree.fit(X_train_balanced,y_train_balanced)
print("Mejores hiperparámetros para Arbol de decision", grid_search_tree.best_pa
```

```
Mejores hiperparámetros para Arbol de decision {'max_depth': 10, 'min_samples_sp
lit': 10}
```

In [79]:
```
best_model_tree= grid_search_tree.best_estimator_
best_model_tree
```

Out[79]:

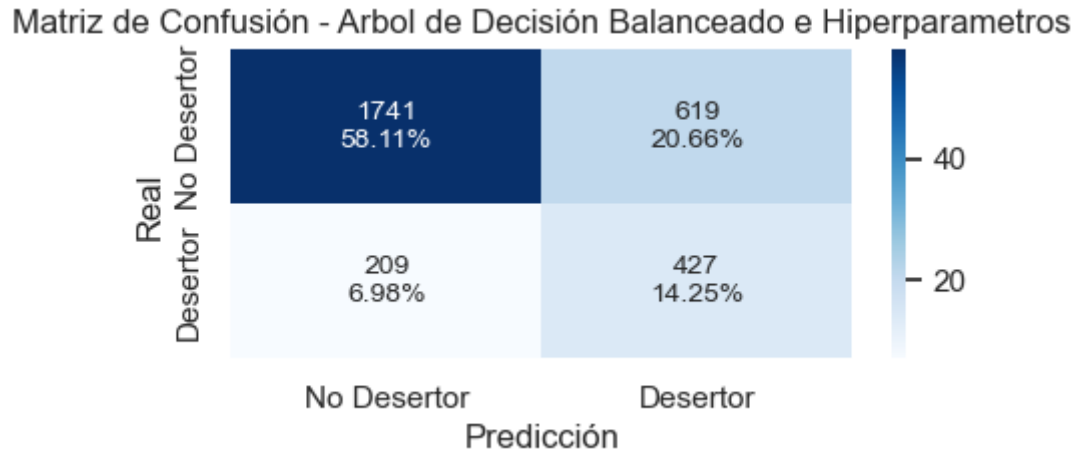
▾ DecisionTreeClassifier

```
DecisionTreeClassifier(max_depth=10, min_samples_split=10)
```

In [80]:
```
y_predtree = best_model_tree.predict(X_test)
y_predtree_proba = best_model_tree.predict_proba(X_test)[:, 1]
```

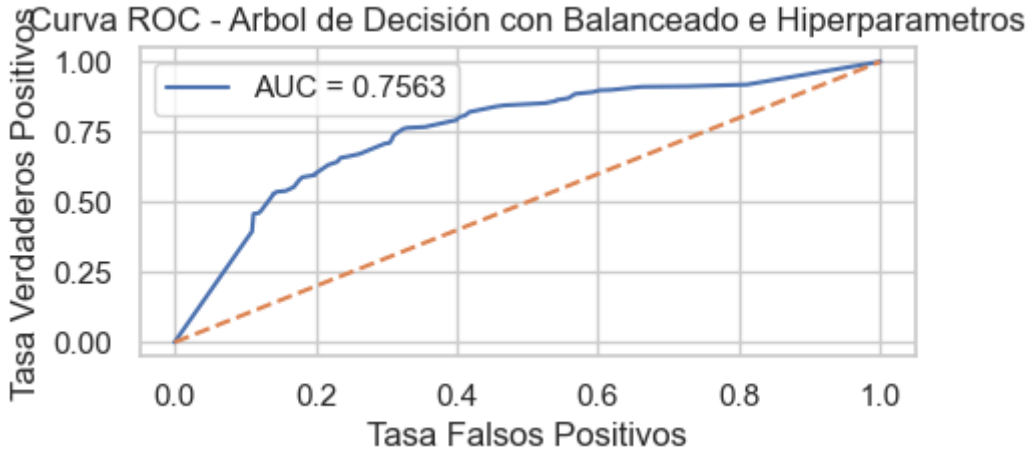
In [81]:
```
# Matriz de confusión- Modelo Arbol de Decisión Hiperparametros
print('Modelo Arbol de Decisión Balanceado e Hiperparametros')
cm=confusion_matrix(y_test, y_predtree)
cm_percentcm = cm.astype('float') / cm.sum() * 100
labels = np.asarray([f'{v}\n{p:.2f}%' for v, p in zip(cm.flatten(), cm_percentcm

# Crear un gráfico de la matriz de confusión
plt.figure(figsize=(5, 2))  # Tamaño del gráfico
sns.heatmap(cm_percentcm, annot=labels , fmt='', cmap='Blues', cbar=True,
            annot_kws={"size": 10},
            xticklabels=['No Desertor', 'Desertor'],
            yticklabels=['No Desertor', 'Desertor'])

# Añadir etiquetas y título
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión - Arbol de Decisión Balanceado e Hiperparametros'

# Mostrar el gráfico
plt.show()


# Reporte de clasificación
print(classification_report(y_test, y_predtree, digits= 4))

# Calcular y mostrar el AUC-ROC
auc_tree = roc_auc_score(y_test, y_predtree_proba)
print(f'AUC-ROC: {auc_tree:.4f}')

# Graficar la curva ROC
fpr_tree, tpr_tree, thresholds = roc_curve(y_test, y_predtree_proba)
plt.figure(figsize=(5,2))
plt.plot(fpr_tree, tpr_tree, label=f'AUC = {auc_tree:.4f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curva ROC - Arbol de Decisión con Balanceado e Hiperparametros')
plt.legend()
plt.show()
```

Modelo Arbol de Decisión Balanceado e Hiperparametros



	precision	recall	f1-score	support
0	0.8928	0.7377	0.8079	2360
1	0.4082	0.6714	0.5077	636
accuracy			0.7236	2996
macro avg	0.6505	0.7045	0.6578	2996
weighted avg	0.7899	0.7236	0.7442	2996

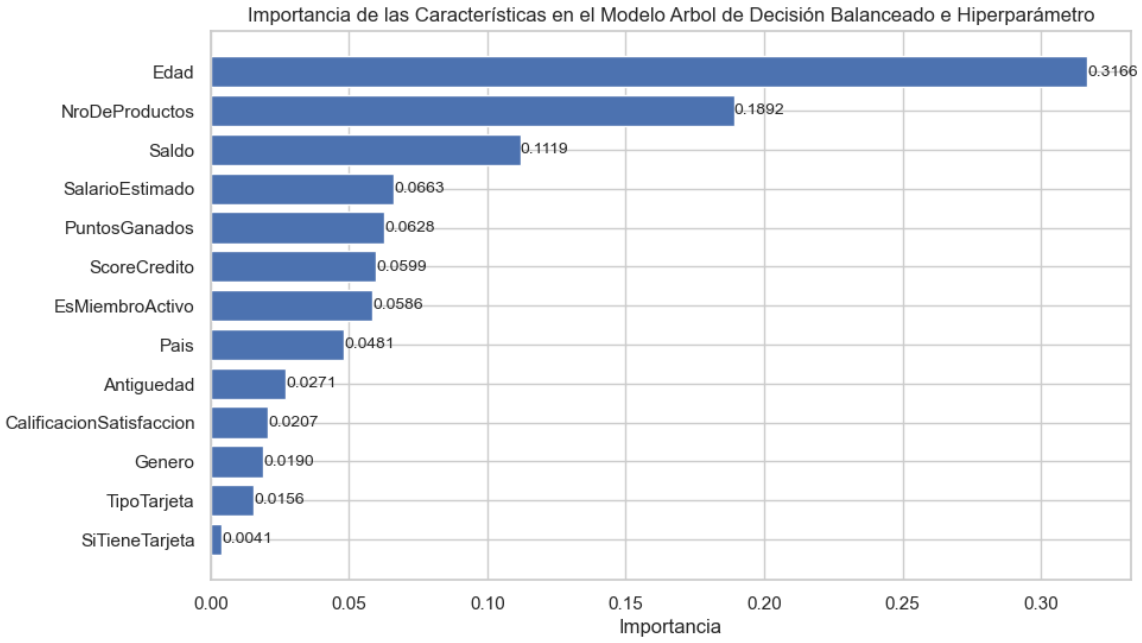
AUC-ROC: 0.7563



Importancia de las Características Arbol de Decisión Balanceado Hiperparámetros

In [82]:
```
# Obtener la importancia de las características

importances =best_model_tree.feature_importances_


# Crear un DataFrame para visualizar la importancia de las características
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Mostrar la tabla de importancia de las características
print(feature_importance_df)

# Graficar la importancia de las características
plt.figure(figsize=(10, 6))
ax = plt.barh(feature_importance_df['Feature'], feature_importance_df['Importanc
plt.xlabel('Importancia')
plt.title('Importancia de las Características en el Modelo Arbol de Decisión Bal
plt.gca().invert_yaxis()

# Añadir anotaciones con los valores en cada barra
for index, value in enumerate(feature_importance_df['Importance']):
    plt.text(value, index, f'{value:.4f}', va='center', ha='left', fontsize=10)

plt.show()
```

```
                   Feature  Importance
3                     Edad    0.316636
6            NroDeProductos    0.189198
5                    Saldo    0.111936
9            SalarioEstimado    0.066300
12             PuntosGanados    0.062833
0               ScoreCredito    0.059880
8            EsMiembroActivo    0.058629
1                     Pais    0.048147
4                Antiguedad    0.027051
10  CalificacionSatisfaccion    0.020723
2                    Genero    0.018977
11              TipoTarjeta    0.015636
7             SiTieneTarjeta    0.004053
```



Hiperparametros Random Forest con GridSearchCV con validación cruzada (precision)

*Búsqueda de mejores parámetros

In [83]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import ParameterGrid
#from joblib import Parallel, delayed, cpu_count
from sklearn.model_selection import RepeatedKFold
from joblib import cpu_count


# Grid de hiperparámetros evaluados
# ==============================================================================
param_grid = {
    'n_estimators': [150],
    'max_features': [5, 7, 9],
    'max_depth'   : [None, 3, 10, 20],
    'criterion'   : ['gini', 'entropy']
}

# Búsqueda por grid search con validación cruzada
# ==============================================================================
grid = GridSearchCV(
        estimator  = RandomForestClassifier(),
        param_grid = param_grid,
        scoring    = 'precision',
        n_jobs     = cpu_count() - 1,
        cv         = RepeatedKFold(n_splits=5, n_repeats=3, random_state=42),
        refit      = True,
        verbose    = 0,
        return_train_score = True
       )

grid.fit(X=X_train_balanced, y=y_train_balanced)

# Resultados
# ==============================================================================
resultados = pd.DataFrame(grid.cv_results_)
resultados.filter(regex='(param*|mean_t|std_t)') \
    .drop(columns='params') \
    .sort_values('mean_test_score', ascending = False) \
    .head(4)
```

Out[83]:

	param_criterion	param_max_depth	param_max_features	param_n_estimators	mean_test_sc
12	entropy	None	5	150	0.790
9	gini	20	5	150	0.789
6	gini	10	5	150	0.789
22	entropy	20	7	150	0.788

In [84]:
```
# Mejores hiperparámetros encontrados por validación cruzada
# ==============================================================================
print("----------------------------------------")
print("Mejores hiperparámetros encontrados por (cv)")
print("----------------------------------------")
print(grid.best_params_, ":", grid.best_score_, grid.scoring)
```

```
-------------------------------------------
Mejores hiperparámetros encontrados por (cv)
-------------------------------------------
{'criterion': 'entropy', 'max_depth': None, 'max_features': 5, 'n_estimators': 1
50} : 0.7900871900950539 precision
```

!pip install scikit-optimize

In [85]:
```
best_model_rfo= grid.best_estimator_
best_model_rfo
```

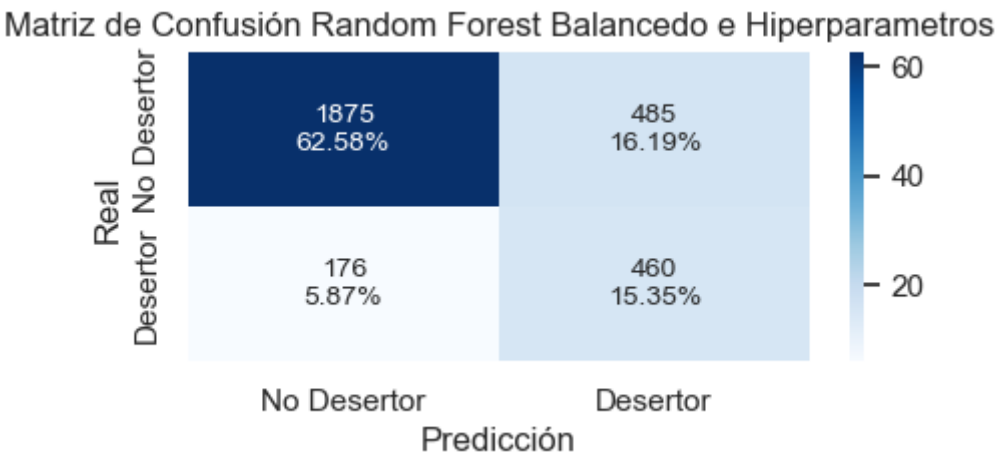
Out[85]:
```
RandomForestClassifier(criterion='entropy', max_features=5, n_estimator
s=150)
```

*Aplicación del modelo

In [86]:
```
y_predrfo = best_model_rfo.predict(X_test)
y_predrfo_proba = best_model_rfo.predict_proba(X_test)[:, 1]
```

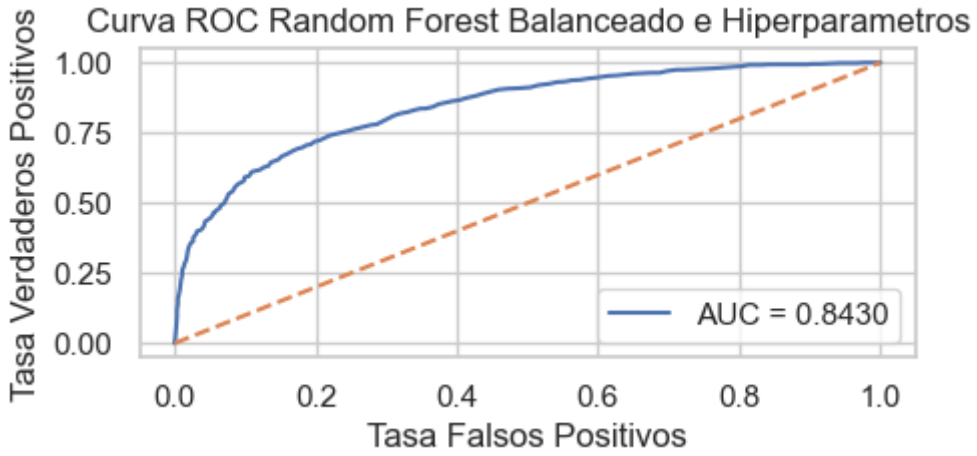
In [87]:
```
# Matriz de confusión- Modelo Random Forest
print('Modelo de Random Forest Balanceado e Hiperparametros')
cm=confusion_matrix(y_test, y_predrfo)
cm_percentcm = cm.astype('float') / cm.sum() * 100
labels = np.asarray([f'{v}\n{p:.2f}%' for v, p in zip(cm.flatten(), cm_percentcm

# Crear un gráfico de la matriz de confusión
plt.figure(figsize=(5, 2))  # Tamaño del gráfico
sns.heatmap(cm_percentcm, annot=labels , fmt='', cmap='Blues', cbar=True,
            annot_kws={"size": 10},
            xticklabels=['No Desertor', 'Desertor'],
            yticklabels=['No Desertor', 'Desertor'])


# Añadir etiquetas y título
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.title('Matriz de Confusión Random Forest Balancedo e Hiperparametros')

# Mostrar el gráfico
plt.show()


# Reporte de clasificación
print(classification_report(y_test, y_predrfo, digits= 4))


# Calcular y mostrar el AUC-ROC
auc_rfo = roc_auc_score(y_test, y_predrfo_proba)
print(f'AUC-ROC: {auc_rfo:.4f}')


# Graficar la curva ROC
fpr_rfo, tpr_rfo, thresholds = roc_curve(y_test, y_predrfo_proba)
plt.figure(figsize=(5,2))
plt.plot(fpr_rfo, tpr_rfo, label=f'AUC = {auc_rfo:.4f}')
plt.plot([0, 1], [0, 1], linestyle='--')
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curva ROC Random Forest Balanceado e Hiperparametros')
plt.legend()
plt.show()
```

```
Modelo de Random Forest Balanceado e Hiperparametros
```



```
              precision    recall  f1-score   support

           0     0.9142    0.7945    0.8501      2360
           1     0.4868    0.7233    0.5819       636

    accuracy                         0.7794      2996
   macro avg     0.7005    0.7589    0.7160      2996
weighted avg     0.8235    0.7794    0.7932      2996
```

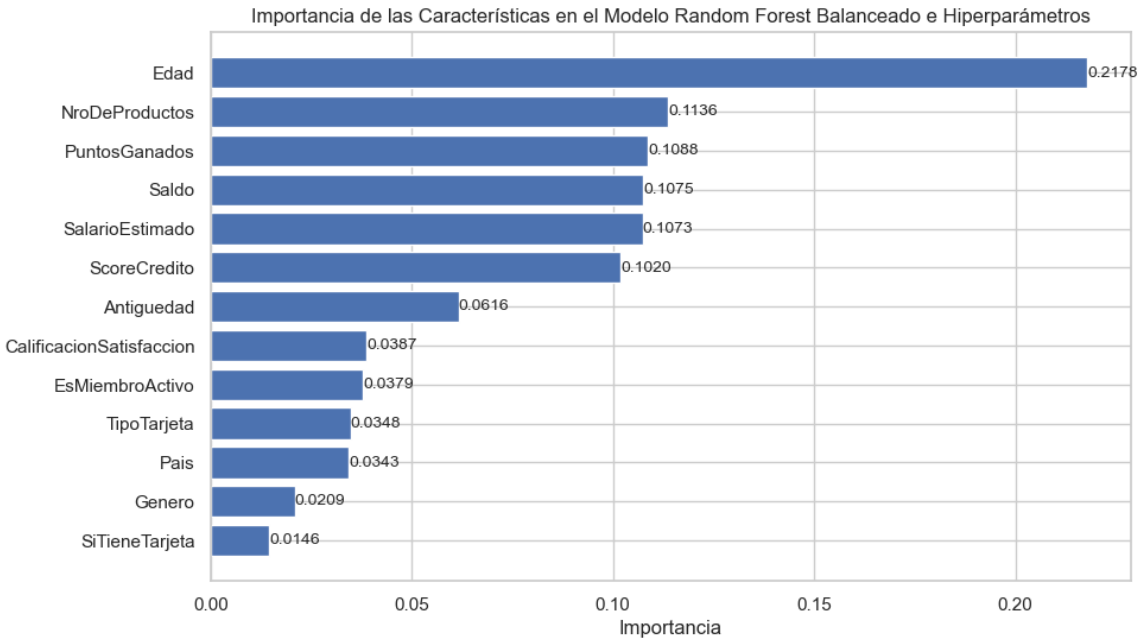
AUC-ROC: 0.8430



Importancia de las Características Random Forest Balanceado Hiperparámetros

In [88]:
```
# Obtener la importancia de las características

importances =best_model_rfo.feature_importances_


# Crear un DataFrame para visualizar la importancia de las características
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
    'Importance': importances
}).sort_values(by='Importance', ascending=False)

# Mostrar la tabla de importancia de las características
print(feature_importance_df)

# Graficar la importancia de las características
plt.figure(figsize=(10, 6))
ax = plt.barh(feature_importance_df['Feature'], feature_importance_df['Importanc
plt.xlabel('Importancia')
plt.title('Importancia de las Características en el Modelo Random Forest Balance
plt.gca().invert_yaxis()

# Añadir anotaciones con los valores en cada barra
for index, value in enumerate(feature_importance_df['Importance']):
    plt.text(value, index, f'{value:.4f}', va='center', ha='left', fontsize=10)

plt.show()
```

	Feature	Importance
3	Edad	0.217813
6	NroDeProductos	0.113633
12	PuntosGanados	0.108752
5	Saldo	0.107537
9	SalarioEstimado	0.107346
0	ScoreCredito	0.101950
4	Antiguedad	0.061645
10	CalificacionSatisfaccion	0.038725
8	EsMiembroActivo	0.037877
11	TipoTarjeta	0.034829
1	Pais	0.034339
2	Genero	0.020948
7	SiTieneTarjeta	0.014605

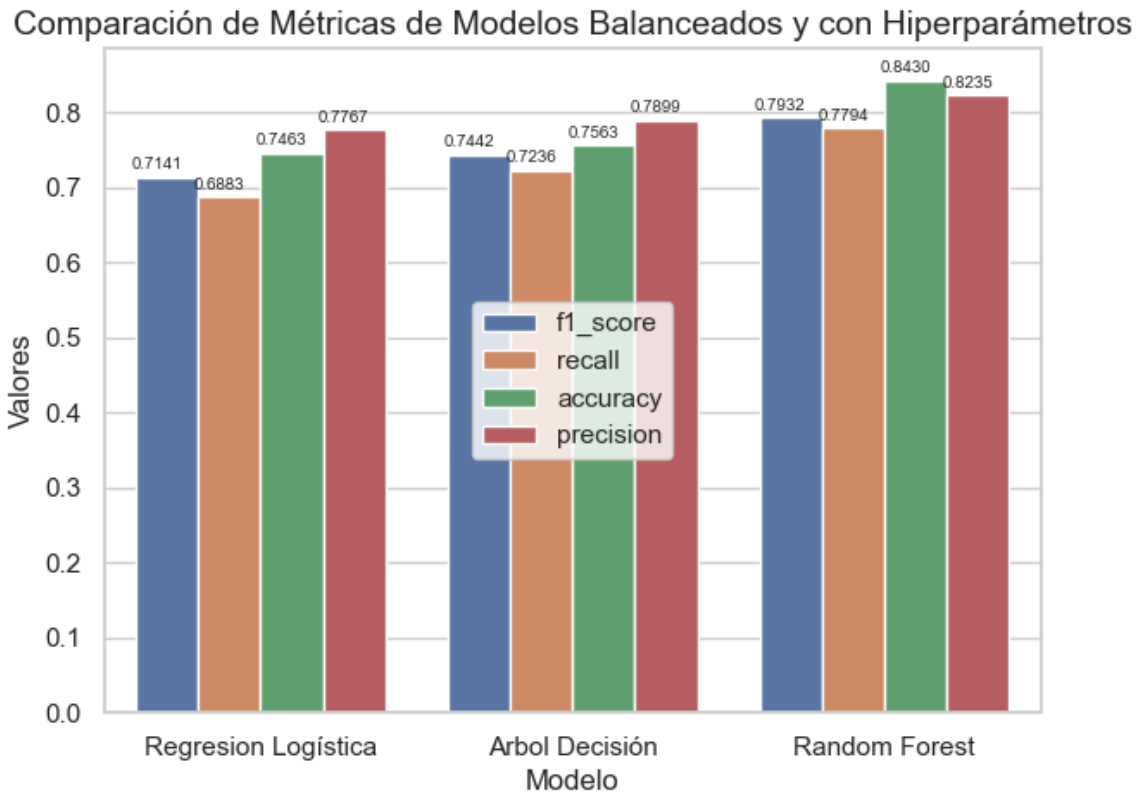


COMPARACION METRICAS DE LOS MODELOS BALANCEADOS Y CON HIPERPARAMETROS

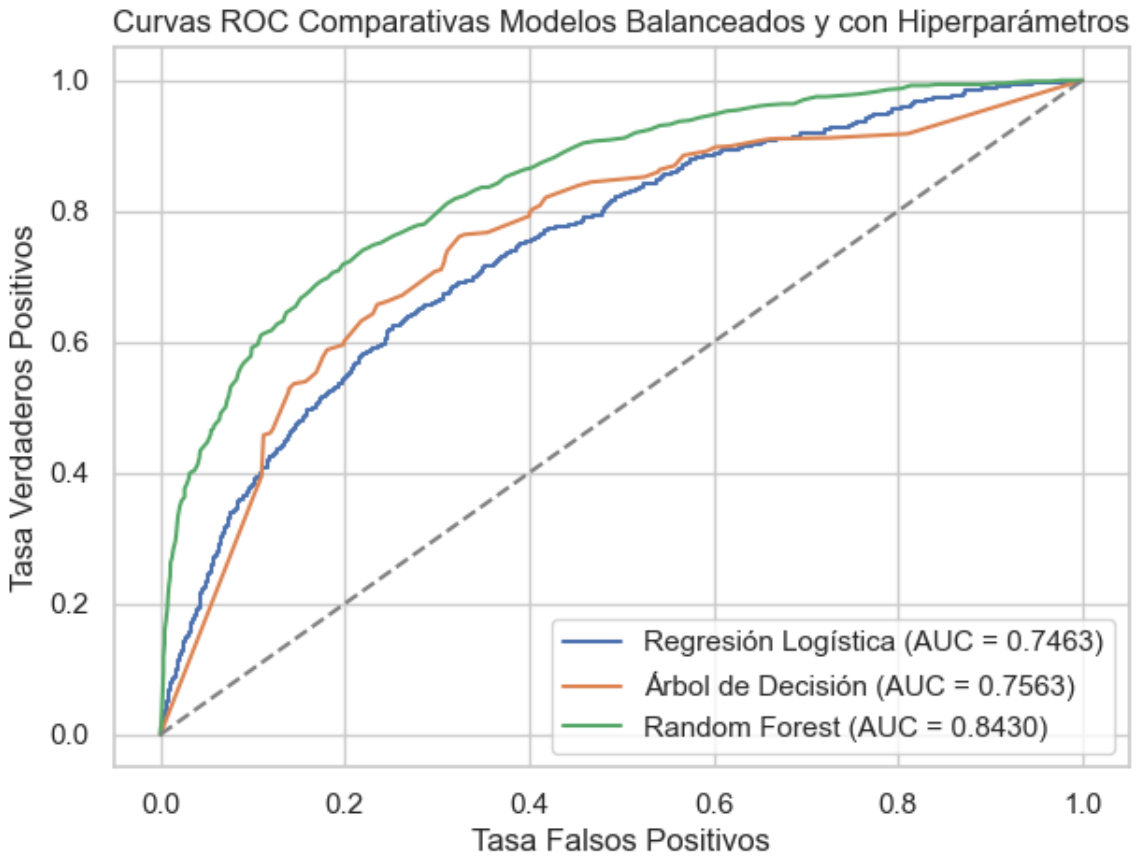
In [89]:

```
results = {}
results['Modelo'] = 'Regresion Logística'
results['f1_score'] = f1_score(y_test, y_predlog, average='weighted')
results['recall'] = recall_score(y_test, y_predlog, average='weighted')
results['accuracy'] = roc_auc_score(y_test, y_predlog_proba)
results['precision'] = precision_score(y_test, y_predlog, average='weighted')

# Calcular las métricas para el modelo de Árbol de Decisión
results_ad = {}
results_ad['Modelo'] = 'Arbol Decisión'
results_ad['f1_score'] = f1_score(y_test, y_predtree, average='weighted')
results_ad['recall'] = recall_score(y_test, y_predtree, average='weighted')
results_ad['accuracy'] = roc_auc_score(y_test, y_predtree_proba)
results_ad['precision'] = precision_score(y_test, y_predtree, average='weighted'

# Calcular las métricas para el modelo de Bosque Aleatorio
results_rf = {}
results_rf['Modelo'] = 'Random Forest'
results_rf['f1_score'] = f1_score(y_test, y_predrfo, average='weighted')
results_rf['recall'] = recall_score(y_test, y_predrfo, average='weighted')
results_rf['accuracy'] = roc_auc_score(y_test, y_predrfo_proba)
results_rf['precision'] = precision_score(y_test, y_predrfo, average='weighted')

# Crear un DataFrame con los resultados de los modelos
metrics = pd.DataFrame([results, results_ad, results_rf], index=[0, 1, 2])
metrics= metrics.round(4)

# Mostrar el DataFrame con las métricas
print(metrics)

# Transformar el DataFrame para que sea más fácil de graficar
dfm = metrics.melt('Modelo', var_name='Métrica', value_name='Valores')

# Configurar el estilo del gráfico
sns.set_theme(style='whitegrid', rc={'figure.figsize':(7, 5)})  # Ajusta el tama

# Crear el gráfico de barras
ax = sns.barplot(data=dfm, x="Modelo", y="Valores", hue='Métrica', palette='deep


# Anotar los valores en las barras
for p in ax.patches:
    ax.annotate(f'{p.get_height():.4f}',
                (p.get_x() + p.get_width() / 3., p.get_height()),
                ha='center',va='center',
                xytext=(0,5),
                textcoords='offset points',
                fontsize=7)
#Título
plt.title('Comparación de Métricas de Modelos Balanceados y con Hiperparámetros'

# Mostrar el gráfico
ax.legend(loc='center')
plt.show()
```

```
                Modelo  f1_score  recall  accuracy  precision
0  Regresion Logística    0.7141  0.6883    0.7463     0.7767
1       Arbol Decisión    0.7442  0.7236    0.7563     0.7899
2        Random Forest    0.7932  0.7794    0.8430     0.8235
```



In [90]:
```
# Graficar las curvas ROC de los tres modelos balancedos y con hiperparámetros
plt.figure(figsize=(7, 5))

# Curva ROC Regresión Logística
plt.plot(fpr_log, tpr_log, label=f'Regresión Logística (AUC = {auc_log:.4f})')

# Curva ROC Árbol de Decisión
plt.plot(fpr_tree, tpr_tree, label=f'Árbol de Decisión (AUC = {auc_tree:.4f})')

# Curva ROC Random Forest
plt.plot(fpr_rfo, tpr_rfo, label=f'Random Forest (AUC = {auc_rfo:.4f})')

# Línea diagonal de referencia (clasificador aleatorio)
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')

# Etiquetas y título
plt.xlabel('Tasa Falsos Positivos')
plt.ylabel('Tasa Verdaderos Positivos')
plt.title('Curvas ROC Comparativas Modelos Balanceados y con Hiperparámetros')

# Leyenda y mostrar gráfico
plt.legend()
plt.show()
```


Curvas ROC Comparativas Modelos Balanceados y con Hiperparámetros
Tasa Verdaderos Positivos
Tasa Falsos Positivos
0.0
0.2
0.4
0.6
0.8
1.0
Regresión Logística (AUC = 0.7463)
Árbol de Decisión (AUC = 0.7563)
Random Forest (AUC = 0.8430)

INTERPRETACION RESULTADOS

La capacidad del modelo Regresión Logística para identificar a todos los clientes en riesgo de desertar es limitada con un recall del 69% y una precisión del 68%. Lo que significa que es probable que algunos clientes que efectivamente abandonarán no sean detectados, lo que puede llevar a una pérdida de oportunidades para prevenir la deserción.

El Modelo de Árbol de Decisión detectó correctamente el 72%(recall) de los clientes que desertaron y de estos el 79% (precisión) efectivamente lo eran, es decir el modelo tiene un buen desempeño.

El Modelo Random Forest identifica de forma correcta el 78%(recall) de los clientes que desertaran con una precisión del 82%, un f1-score de 79%. El valor de las métricas permite al banco actuar de manera efectiva sobre una gran cantidad de clientes en riesgo sin generar una gran cantidad de falsos positivos.

El modelo que mejor desempeño tiene es el del Random Forest, de acuerdo a los valores de las métricas de f1-score y recall, pero además su AUC es un poco más alto que el del Árbol de decisión y Regresión Logística, con lo que el modelo podrá distinguir mejor entre un desertor y un no desertor.

Random Forest con relación a la clase Desertora identifica correctamente el 72% (recall) de todos los clientes que efectivamente desertaron, es decir el modelo tiene un rendimiento razonable para detectar desertores. Sin embargo de todas los clientes que el modelo predijo como desertores, el 49% (precisión) efectivamente lo fueron. Esto indica que la mitad de las predicciones son incorrectas.

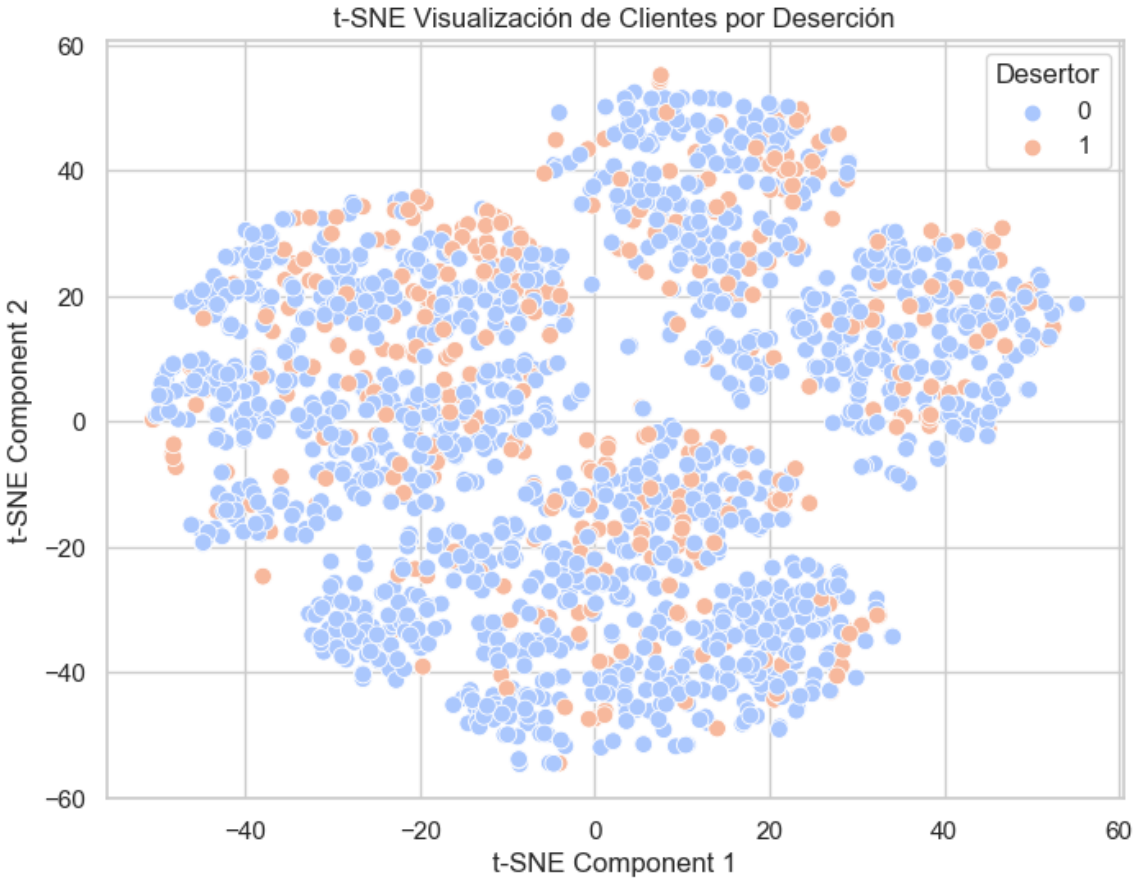
Por lo que se sugiere la inclusión de variables relevantes como: Número de reclamos, Número de transacciones en un período específico, Estado civil, Ocupación, Canal, Número de dependencias para mejor la precisión del modelo en la predicción de la clase de estudio

REDUCCION DE DIMENSIONES

Con información de TEST

In [91]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE

#df = df_balanceado_sm

#data_numeric = df.select_dtypes(include=['number'])

# Escalar los datos numéricos normalizados
scaler = StandardScaler()
X_data = scaler.fit_transform(X_test)

# Tomar una muestra de los datos (por ejemplo, 2000 filas) para mejorar la veloc
sampled_data = X_data[:2000]
sampled_labels = y_test[:2000]  # Etiquetas correspondientes a la muestra

# Aplicar t-SNE con 2 componentes
tsne = TSNE(n_components=2, perplexity=30, random_state=0)
tsne_results = tsne.fit_transform(sampled_data)

# Crear un DataFrame con los resultados de t-SNE
tsne_df = pd.DataFrame(tsne_results, columns=['C1', 'C2'])
tsne_df['Desertor'] = sampled_labels.values

# Visualización de los resultados de t-SNE
plt.figure(figsize=(8, 6))
sns.scatterplot(data=tsne_df, x='C1', y='C2', hue='Desertor', palette='coolwarm'
plt.title('t-SNE Visualización de Clientes por Deserción')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.show()
```



Al aplicar la técnica de reducción de dimensiones se puede visualizar datos de alta dimensionalidad en un espacio de menor dimensión. El t-SNE se especializa en preservar la estructura local de los datos, es decir, mantiene las relaciones de proximidad entre los puntos, lo que hace que sea especialmente útil para visualizar agrupamientos o patrones en los datos.

El gráfico muestra que no existe separabilidad entre las clases desertor y no desertor. Si bien el modelo en general tiene un rendimiento sobre 0.84, la visualización del t-SNE del modelo nos indica que no es eficiente, por lo que se sugiere como se indicaba anteriormente la inclusión de variables relevantes al modelo para alcanzar mayor precisión.

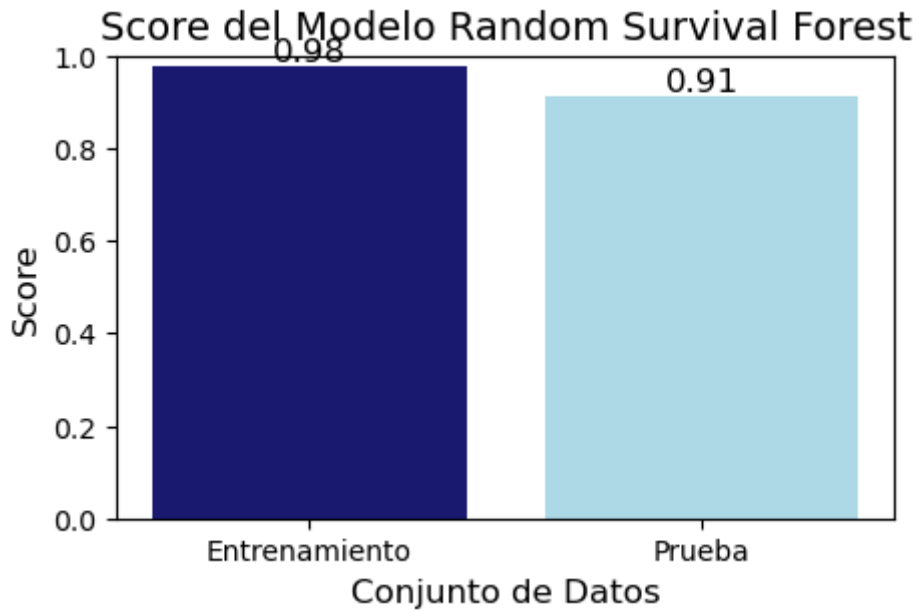
MODELO RANDOM SURVIVAL FOREST

Con el modelo de Random Surival Forest se pretende determinar la duración o permanencia de los clientes en la entidad financiera

In [64]:
```
#Survival
from sksurv.ensemble import RandomSurvivalForest
from sksurv.util import Surv

df=data_rsf
# Definir variables predictoras (X) y dependientes (y)


#X = df.drop(columns=[ 'Genero','Pais','SiTieneTarjeta','TipoTarjeta','Rangos Ed
X = df.drop(columns=[ 'Desertor','Antiguedad'])
y = Surv.from_arrays(event=df['Desertor'].astype(bool), time=df['Antiguedad'])

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_

# Crear y entrenar el modelo Random Survival Forest
rsf = RandomSurvivalForest(n_estimators=100, min_samples_split=10, min_samples_l
rsf.fit(X_train, y_train)

# Evaluar el modelo
train_score = rsf.score(X_train, y_train)
test_score = rsf.score(X_test, y_test)

print("SCORE DEL MODELO RANDOM SURVIVAL FOREST")
print("Training Score:", train_score)
print("Test Score:", test_score)
```

```
SCORE DEL MODELO RANDOM SURVIVAL FOREST
Training Score: 0.976582219980611
Test Score: 0.9128724648448671
```

In [65]:
```python
import matplotlib.pyplot as plt

# Asignar los valores de los scores de entrenamiento y prueba
scores = {'Entrenamiento': train_score, 'Prueba': test_score}

# Crear la gráfica de barras
plt.figure(figsize=(5,3))
plt.bar(scores.keys(), scores.values(), color=['#191970','#ADD8E6'])


# Añadir título y etiquetas
plt.title('Score del Modelo Random Survival Forest', fontsize=14)
plt.xlabel('Conjunto de Datos', fontsize=12)
plt.ylabel('Score', fontsize=12)

# Mostrar los valores de los scores encima de cada barra
for i, (key, value) in enumerate(scores.items()):
    plt.text(i, value + 0.01, f'{value:.2f}', ha='center', fontsize=12)

# Mostrar la gráfica
plt.ylim(0, 1)
plt.show()
```



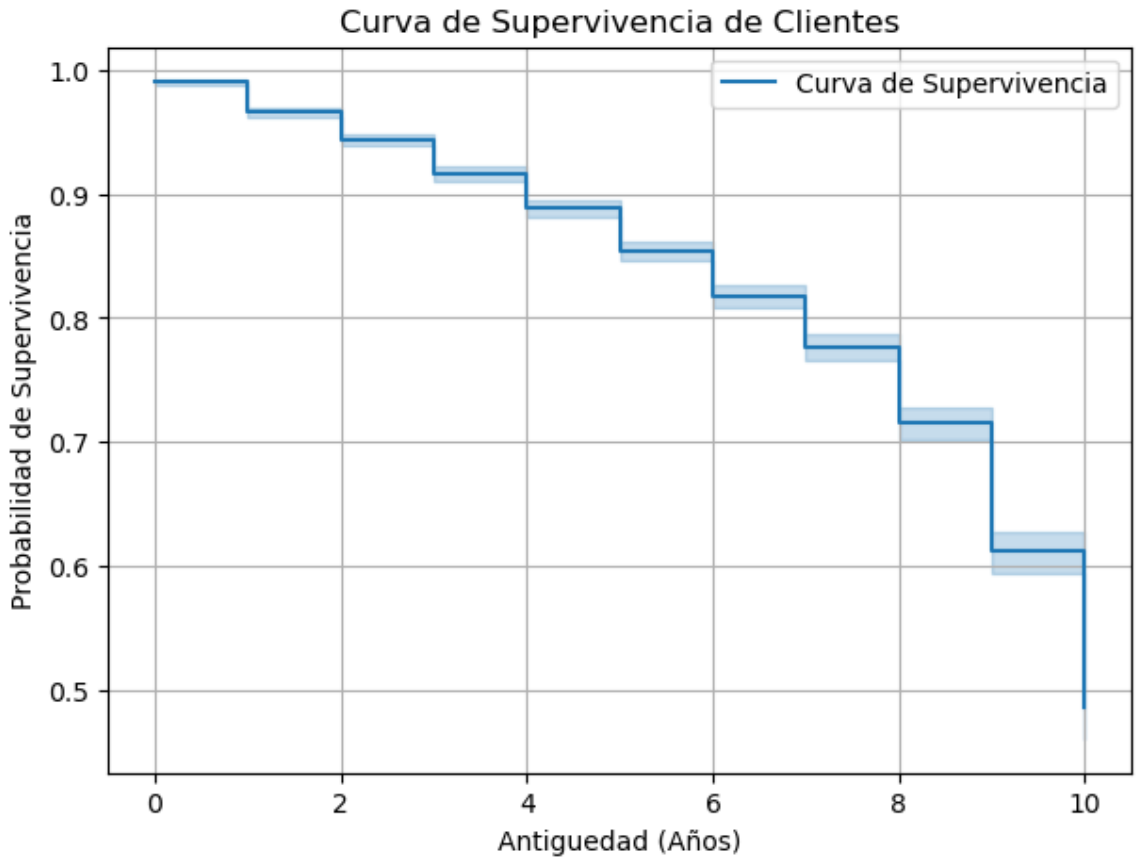
El Modelo Random Survival Forest tiene un buen desempeño, aprende de los patrones del conjunto de entrenamiento (0.92) y generaliza a datos nuevos (0.79). Aunque hay diferencia entre el training score y el test score, ambos valores son significativos, lo que indica que el modelo es aceptable.

ANALISIS DE SUPERVIVENCIA - PREDICCIONES CON RANDOM SURVIVAL FOREST

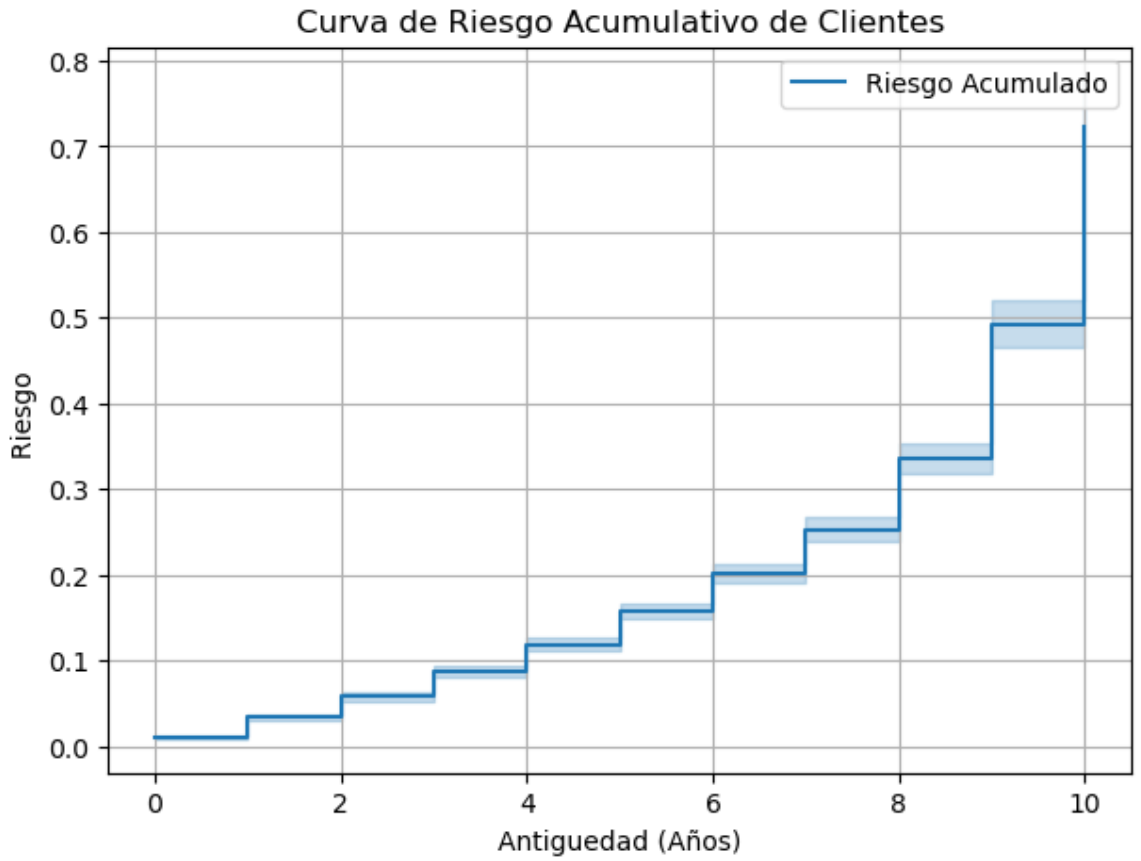
In [66]:
```
#Survival
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sksurv.ensemble import RandomSurvivalForest
from sksurv.datasets import load_whas500
from sksurv.util import Surv

data=pd.read_csv('data/Customer-Churn-Records.csv')

data.rename({'RowNumber':'NroRegistro', 'CustomerId': 'IdCliente', 'Surname':'No

#Elimino variables no relevantes
data = data.drop(columns=['NombreCliente','NroRegistro'])


# Definir variables predictoras (X) y dependientes (y)

X = data.drop(columns=['Desertor','Antiguedad'])
y = Surv.from_arrays(event=data['Desertor'].astype(bool), time=data['Antiguedad'
```

In [67]:
```
# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

In [68]:
```
# Seleccionar las columnas necesarias para el análisis de supervivencia

from lifelines import KaplanMeierFitter
#from lifelines.plotting import plot_lifetimes

duration = data['Antiguedad']
event_observed = data['Desertor']

# Inicializar el estimador Kaplan-Meier
kmf = KaplanMeierFitter()

# Ajustar el modelo
kmf.fit(duration, event_observed, label='Curva de Supervivencia')

# Graficar la función de supervivencia
plt.figure(figsize=(7, 5))
kmf.plot_survival_function()
plt.title('Curva de Supervivencia de Clientes')
plt.xlabel('Antiguedad (Años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.show()
```



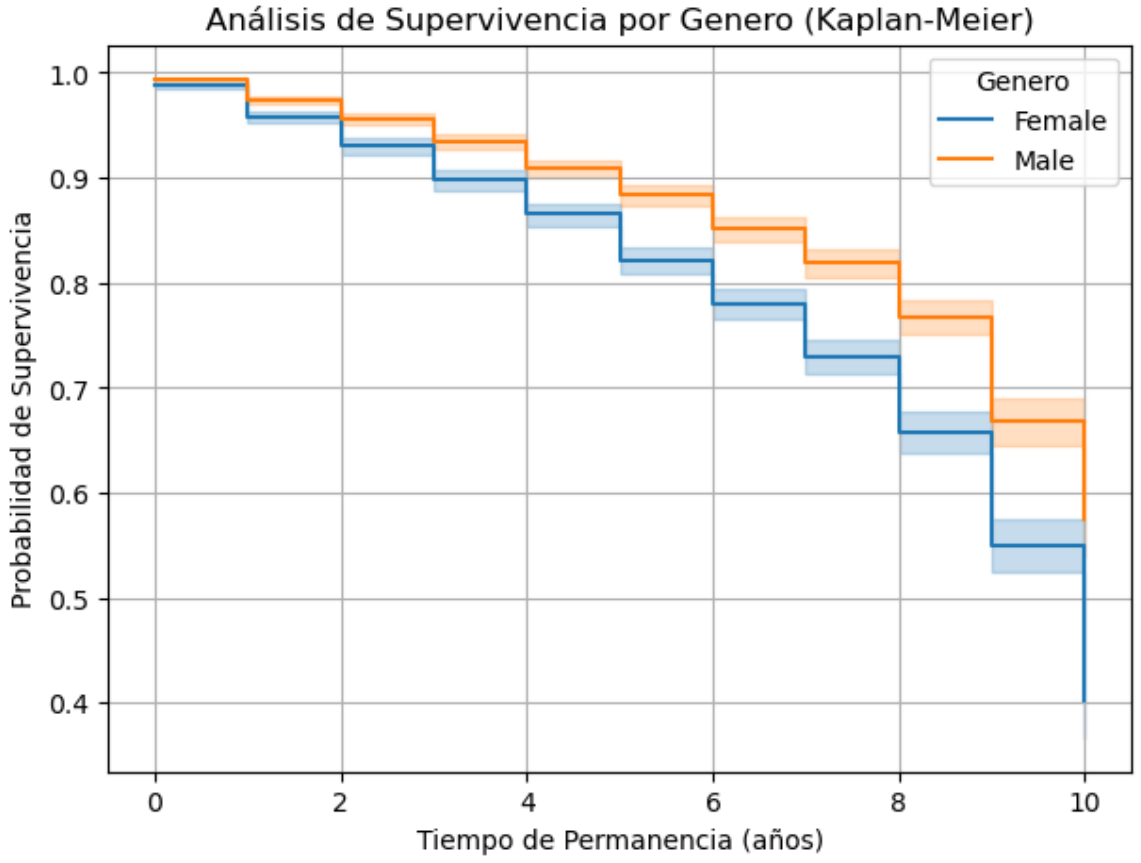
In [69]:
```python
from lifelines import NelsonAalenFitter
naf = NelsonAalenFitter()

# Ajustar el modelo
naf.fit(duration, event_observed, label='Riesgo Acumulado')

# Graficar la función de riesgo acumulativa
plt.figure(figsize=(7, 5))
naf.plot_cumulative_hazard()
plt.title('Curva de Riesgo Acumulativo de Clientes')
plt.xlabel('Antiguedad (Años)')
plt.ylabel('Riesgo ')
plt.grid(True)
plt.show()
```



ANALISIS DE SUPERVIVENCIA POR VARIABLE

In [70]:
```
# Crear un gráfico para el análisis por tipo de tarjeta
plt.figure(figsize=(7, 5))

T = df['Antiguedad']  # Tiempo de permanencia
E = df['Desertor']  # Evento de salida

# Obtener los tipos únicos de tarjetas
genero_types = data['Genero'].unique()

# Hacer el análisis por cada tipo de tarjeta
for genero_type in genero_types:
    mask_type = data['Genero'] == genero_type
    kmf.fit(T[mask_type], event_observed=E[mask_type], label=genero_type)
    kmf.plot_survival_function()

    # Imprimir tabla de supervivencia para cada tipo de tarjeta
    print(f"\nTabla de Supervivencia - {genero_type}")
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por Genero (Kaplan-Meier)')
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="Genero")
plt.show()
```

```
Tabla de Supervivencia - Female
            Female
timeline
0.0       0.988093
1.0       0.958441
2.0       0.930339
3.0       0.897802
4.0       0.865362
5.0       0.821849
6.0       0.779901
7.0       0.730003
8.0       0.658265
9.0       0.549706
10.0      0.401622

Tabla de Supervivencia - Male
              Male
timeline
0.0       0.993211
1.0       0.974082
2.0       0.955797
3.0       0.933990
4.0       0.908944
5.0       0.883471
6.0       0.851625
7.0       0.818619
8.0       0.767090
9.0       0.668110
10.0      0.575013
```



In [71]:

```python
# Crear un gráfico para el análisis por País
plt.figure(figsize=(7, 5))

# Obtener los tipos únicos de País
geo_types = data['Pais'].unique()

# Hacer el análisis
for geo_type in geo_types:
    mask_geo = data['Pais'] == geo_type
    kmf.fit(T[mask_geo], event_observed=E[mask_geo], label=geo_type)
    kmf.plot_survival_function()

    # Imprimir tabla de supervivencia para cada tipo de País
    print(f"\nTabla de Supervivencia - {geo_type}")
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por País (Kaplan-Meier)')
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="País")
plt.show()
```

Tabla de Supervivencia - France

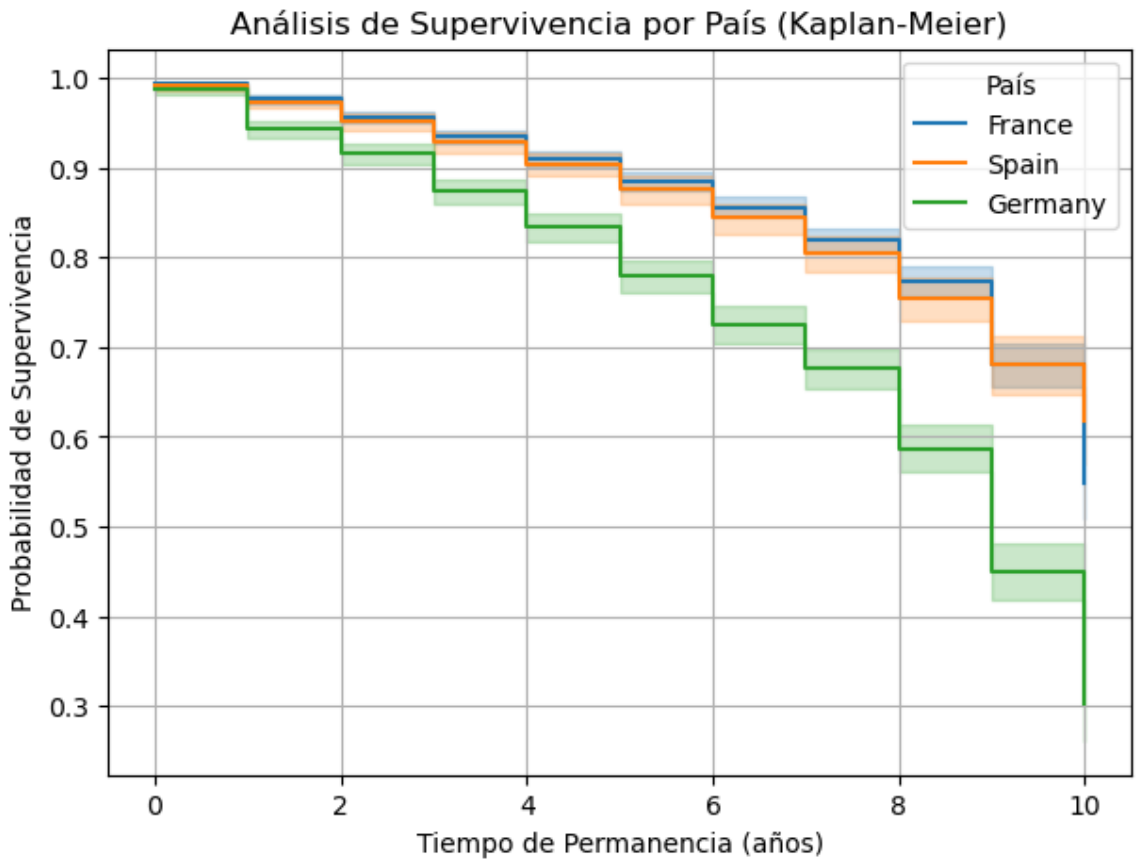
timeline	France
0.0	0.993211
1.0	0.976051
2.0	0.955502
3.0	0.934609
4.0	0.910240
5.0	0.885066
6.0	0.855733
7.0	0.819309
8.0	0.772627
9.0	0.680450
10.0	0.548296

Tabla de Supervivencia - Spain

timeline	Spain
0.0	0.990696
1.0	0.972721
2.0	0.951247
3.0	0.927984
4.0	0.903985
5.0	0.875797
6.0	0.843477
7.0	0.804962
8.0	0.753444
9.0	0.680383
10.0	0.617490

Tabla de Supervivencia - Germany

timeline	Germany
0.0	0.986427
1.0	0.943307
2.0	0.915069
3.0	0.873810
4.0	0.833447
5.0	0.778775
6.0	0.725597
7.0	0.675641
8.0	0.586714
9.0	0.449272
10.0	0.301855



In [72]:
```python
# Definir el tamaño de la figura
plt.figure(figsize=(7, 5))

# Obtener los valores únicos de NumOfProducts
product_groups = data['NroDeProductos'].unique()


# Realizar el análisis Kaplan-Meier para cada grupo de NumDeProductos
for product_group in product_groups:
    mask_product = data['NroDeProductos'] == product_group
    kmf.fit(T[mask_product], event_observed=E[mask_product], label=f'Productos:
    kmf.plot_survival_function()

    # Imprimir la tabla de supervivencia para cada grupo de productos
    print(f"\nTabla de Supervivencia - Número de Productos {product_group}")
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por Número de Productos (Kaplan-Meier)')
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="Número de Productos")
plt.show()
```

Tabla de Supervivencia - Número de Productos 1

timeline	Productos: 1
0.0	0.987384
1.0	0.953315
2.0	0.922081
3.0	0.889323
4.0	0.850184
5.0	0.805971
6.0	0.757996
7.0	0.707350
8.0	0.632980
9.0	0.499541
10.0	0.372574

Tabla de Supervivencia - Número de Productos 3

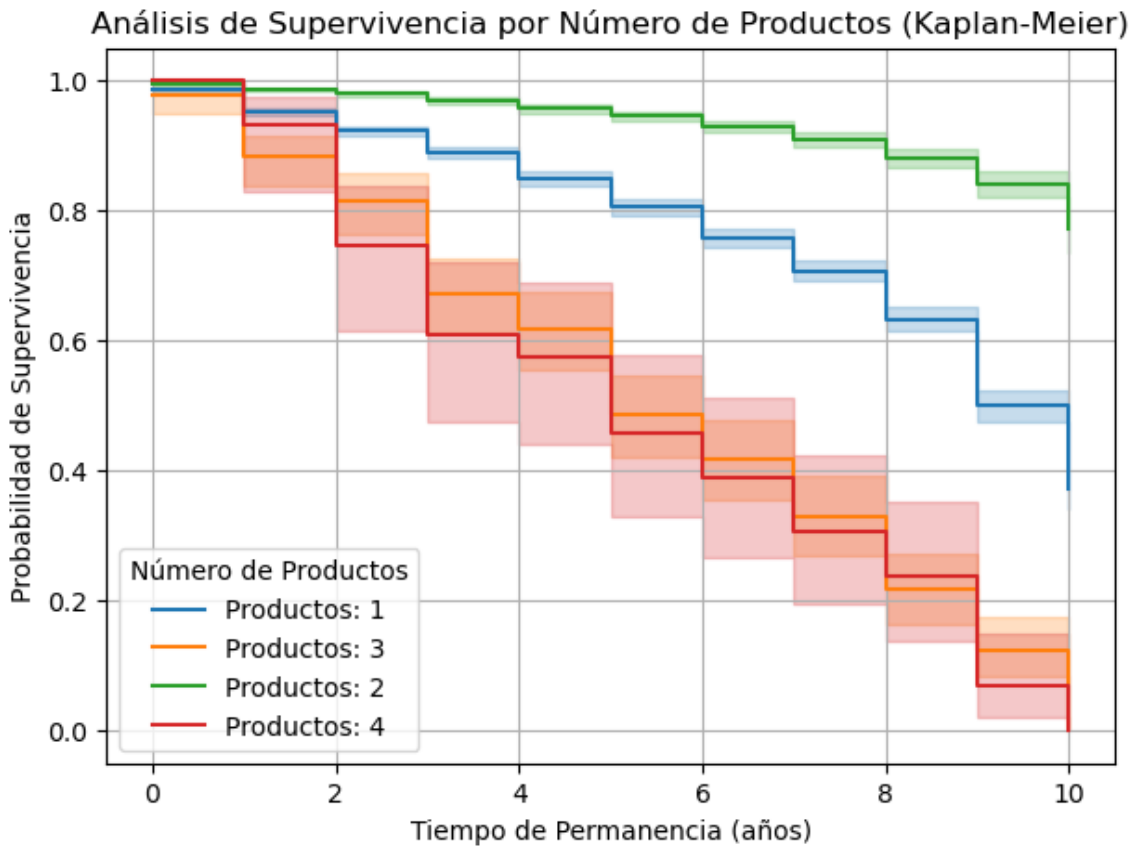
timeline	Productos: 3
0.0	0.977273
1.0	0.882576
2.0	0.816187
3.0	0.672857
4.0	0.619193
5.0	0.484963
6.0	0.416977
7.0	0.329916
8.0	0.216661
9.0	0.124580
10.0	0.007328

Tabla de Supervivencia - Número de Productos 2

timeline	Productos: 2
0.0	0.995424
1.0	0.987540
2.0	0.979368
3.0	0.969987
4.0	0.956474
5.0	0.946359
6.0	0.929111
7.0	0.910289
8.0	0.881719
9.0	0.841464
10.0	0.771342

Tabla de Supervivencia - Número de Productos 4

timeline	Productos: 4
0.0	1.000000
1.0	0.932203
2.0	0.745763
3.0	0.610169
4.0	0.576271
5.0	0.457627
6.0	0.389831
7.0	0.305085
8.0	0.237288
9.0	0.067797
10.0	0.000000



In [73]:
```
# Definir el tamaño de la figura
plt.figure(figsize=(7, 5))

# Obtener los valores únicos de CalificacionSatisfaccion
satisfaction_groups = data['CalificacionSatisfaccion'].unique()


# Realizar el análisis Kaplan-Meier para cada grupo de CalificacionSatisfaccion
for satisfaction_group in satisfaction_groups:
    mask_satisfaction = data['CalificacionSatisfaccion'] == satisfaction_group
    kmf.fit(T[mask_satisfaction], event_observed=E[mask_satisfaction], label=f'S
    kmf.plot_survival_function(ci_show=False)  # Desactivar las bandas de confia

    # Imprimir la tabla de supervivencia para cada grupo de satisfacción
    print(f"\nTabla de Supervivencia - Puntuación de Satisfacción {satisfaction_
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por Puntuación de Satisfacción (Kaplan-Meie
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="Puntuación de Satisfacción")
plt.show()
```

Tabla de Supervivencia - Puntuación de Satisfacción 2

timeline	Satisfacción: 2
0.0	0.991538
1.0	0.965282
2.0	0.942299
3.0	0.914347
4.0	0.886343
5.0	0.857440
6.0	0.824280
7.0	0.765567
8.0	0.685306
9.0	0.581020
10.0	0.431080

Tabla de Supervivencia - Puntuación de Satisfacción 3

timeline	Satisfacción: 3
0.0	0.990682
1.0	0.965965
2.0	0.945507
3.0	0.916575
4.0	0.884403
5.0	0.858802
6.0	0.819599
7.0	0.789492
8.0	0.744000
9.0	0.632623
10.0	0.542248

Tabla de Supervivencia - Puntuación de Satisfacción 5

timeline	Satisfacción: 5
0.0	0.991018
1.0	0.970307
2.0	0.944083
3.0	0.918789
4.0	0.885288
5.0	0.848537
6.0	0.820794
7.0	0.780870
8.0	0.720436
9.0	0.630970
10.0	0.494879

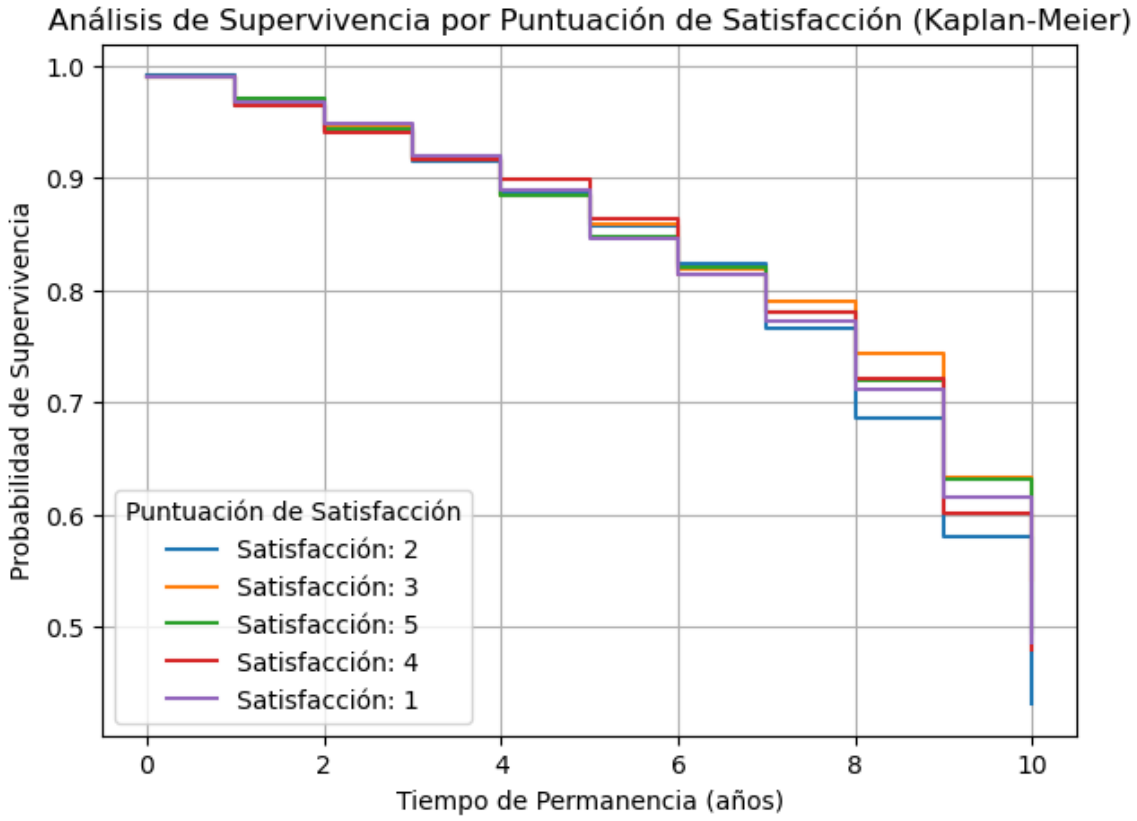
Tabla de Supervivencia - Puntuación de Satisfacción 4

timeline	Satisfacción: 4
0.0	0.990514
1.0	0.964840
2.0	0.940761
3.0	0.917365
4.0	0.899255
5.0	0.863789
6.0	0.814615
7.0	0.780049
8.0	0.720631
9.0	0.601790
10.0	0.480152

Tabla de Supervivencia - Puntuación de Satisfacción 1

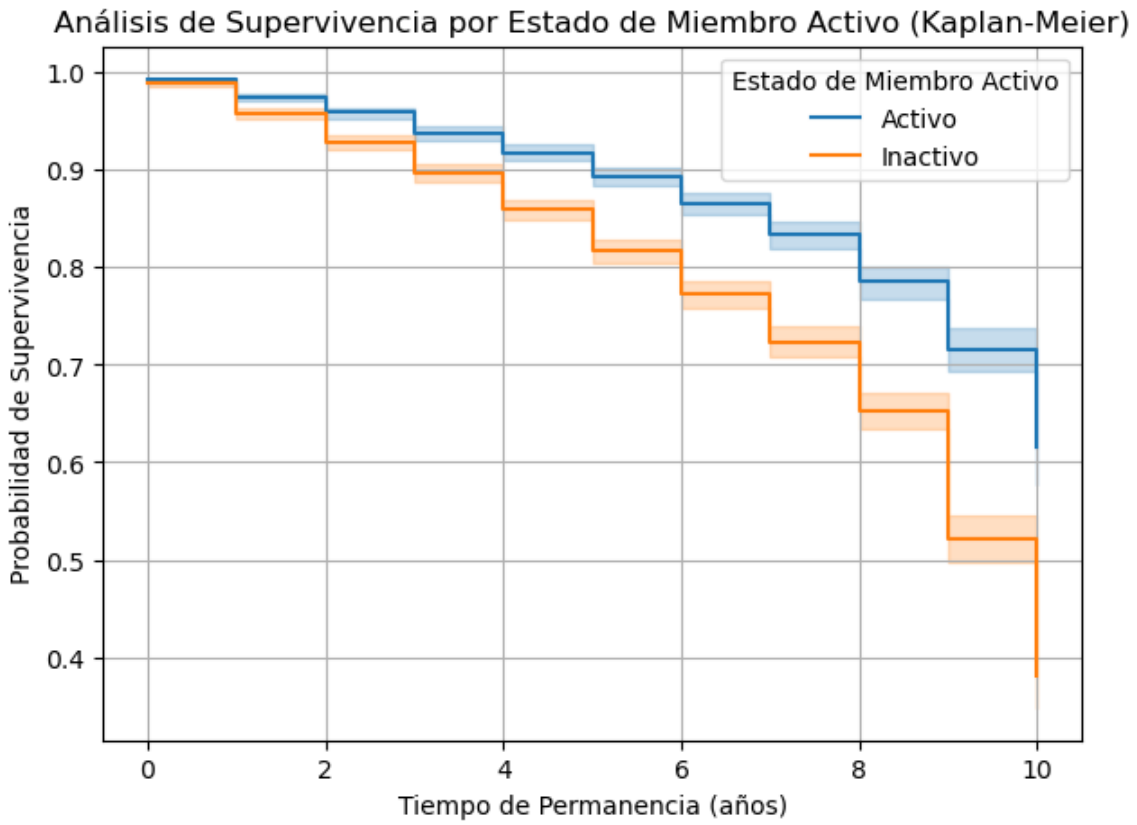
timeline	Satisfacción: 1

```
0.0             0.990674
1.0             0.968635
2.0             0.948651
3.0             0.920592
4.0             0.890030
5.0             0.846825
6.0             0.813858
7.0             0.772372
8.0             0.711926
9.0             0.615083
10.0            0.485592
```



In [74]:
```python
# Definir el tamaño de la figura
plt.figure(figsize=(7, 5))

# Obtener los valores únicos de EsMiembroActivo (0 o 1)
active_groups = data['EsMiembroActivo'].unique()


# Realizar el análisis Kaplan-Meier para cada grupo de EsMiembroActivo
for active_group in active_groups:
    mask_active = data['EsMiembroActivo'] == active_group
    label = 'Activo' if active_group == 1 else 'Inactivo'
    kmf.fit(T[mask_active], event_observed=E[mask_active], label=label)
    kmf.plot_survival_function()

    # Imprimir la tabla de supervivencia para cada grupo (activo/inactivo)
    print(f"\nTabla de Supervivencia - Miembro {label}")
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por Estado de Miembro Activo (Kaplan-Meier)')
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="Estado de Miembro Activo")
plt.show()
```

```
Tabla de Supervivencia - Miembro Activo
            Activo
timeline
0.0       0.992423
1.0       0.975111
2.0       0.959015
3.0       0.937827
4.0       0.918087
5.0       0.893631
6.0       0.865705
7.0       0.833479
8.0       0.785005
9.0       0.716222
10.0      0.615142

Tabla de Supervivencia - Miembro Inactivo
          Inactivo
timeline
0.0       0.989252
1.0       0.958378
2.0       0.928801
3.0       0.896600
4.0       0.859707
5.0       0.817054
6.0       0.772420
7.0       0.723896
8.0       0.652598
9.0       0.521906
10.0      0.381143
```



In [75]:
```python
# Crear un gráfico para el análisis por tipo de tarjeta
plt.figure(figsize=(7, 5))

# Obtener los tipos únicos de tarjetas
card_types = data['TipoTarjeta'].unique()

# Hacer el análisis por cada tipo de tarjeta
for card_type in card_types:
    mask_card = data['TipoTarjeta'] == card_type
    kmf.fit(T[mask_card], event_observed=E[mask_card], label=card_type)
    kmf.plot_survival_function(ci_show=False)  # Desactivar las bandas de confia

    # Imprimir tabla de supervivencia para cada tipo de tarjeta
    print(f"\nTabla de Supervivencia - {card_type}")
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por Tipo de Tarjeta (Kaplan-Meier)')
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="Tipo de Tarjeta")
plt.show()
```

Tabla de Supervivencia - DIAMOND

timeline	DIAMOND
0.0	0.992812
1.0	0.965634
2.0	0.938623
3.0	0.910515
4.0	0.877598
5.0	0.841244
6.0	0.798884
7.0	0.761014
8.0	0.697799
9.0	0.598392
10.0	0.471614

Tabla de Supervivencia - GOLD

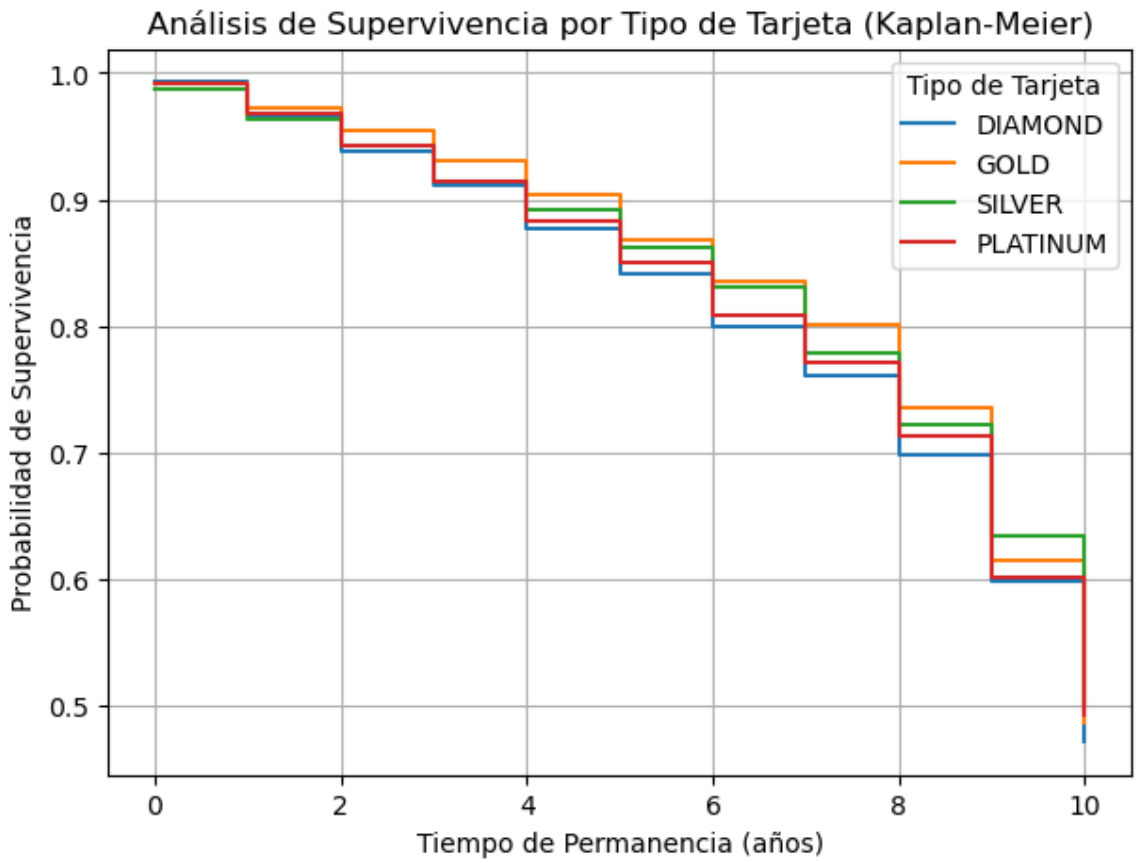
timeline	GOLD
0.0	0.992394
1.0	0.972116
2.0	0.953903
3.0	0.931167
4.0	0.904290
5.0	0.868170
6.0	0.835452
7.0	0.800411
8.0	0.734844
9.0	0.615309
10.0	0.486293

Tabla de Supervivencia - SILVER

timeline	SILVER
0.0	0.987154
1.0	0.962620
2.0	0.942302
3.0	0.914189
4.0	0.891533
5.0	0.861387
6.0	0.831216
7.0	0.778337
8.0	0.721445
9.0	0.634112
10.0	0.499603

Tabla de Supervivencia - PLATINUM

timeline	PLATINUM
0.0	0.991172
1.0	0.967563
2.0	0.942124
3.0	0.914251
4.0	0.882945
5.0	0.850440
6.0	0.809187
7.0	0.771438
8.0	0.713016
9.0	0.602102
10.0	0.492153


Análisis de Supervivencia por Tipo de Tarjeta (Kaplan-Meier)
Probabilidad de Supervivencia
Tiempo de Permanencia (años)
Tipo de Tarjeta
DIAMOND
GOLD
SILVER
PLATINUM
1.0
0.9
0.8
0.7
0.6
0.5
0
2
4
6
8
10

In [76]:

```
import matplotlib.pyplot as plt
from lifelines import KaplanMeierFitter

# Definir los rangos de edad
bins = [18, 30, 40, 50, 60, 100]
labels = ['18-30', '31-40', '41-50', '51-60', '61+']
data['AgeGroup'] = pd.cut(data['Edad'], bins=bins, labels=labels, right=False)


# Inicializar Kaplan-Meier Fitter
kmf = KaplanMeierFitter()

# Definir el tamaño de la figura
plt.figure(figsize=(7, 5))

# Obtener los grupos de edad únicos
age_groups = data['AgeGroup'].unique()

# Realizar el análisis Kaplan-Meier para cada grupo de edad
for age_group in age_groups:
    mask_age = data['AgeGroup'] == age_group
    kmf.fit(T[mask_age], event_observed=E[mask_age], label=age_group)
    kmf.plot_survival_function()

    # Imprimir la tabla de supervivencia para cada grupo de edad
    print(f"\nTabla de Supervivencia - Grupo de Edad {age_group}")
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por Grupos de Edad (Kaplan-Meier)')
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="Grupo de Edad")
plt.show()
```

Tabla de Supervivencia - Grupo de Edad 41-50

timeline	41-50
0.0	0.983544
1.0	0.949248
2.0	0.912657
3.0	0.876507
4.0	0.834547
5.0	0.788247
6.0	0.733003
7.0	0.680988
8.0	0.604635
9.0	0.477187
10.0	0.343183

Tabla de Supervivencia - Grupo de Edad 31-40

timeline	31-40
0.0	0.995625
1.0	0.983189
2.0	0.969419
3.0	0.956267
4.0	0.943985
5.0	0.926664
6.0	0.898798
7.0	0.874458
8.0	0.833786
9.0	0.770660
10.0	0.673354

Tabla de Supervivencia - Grupo de Edad 51-60

timeline	51-60
0.0	0.978035
1.0	0.902982
2.0	0.848630
3.0	0.771600
4.0	0.697063
5.0	0.625108
6.0	0.572328
7.0	0.492425
8.0	0.380000
9.0	0.228000
10.0	0.080470

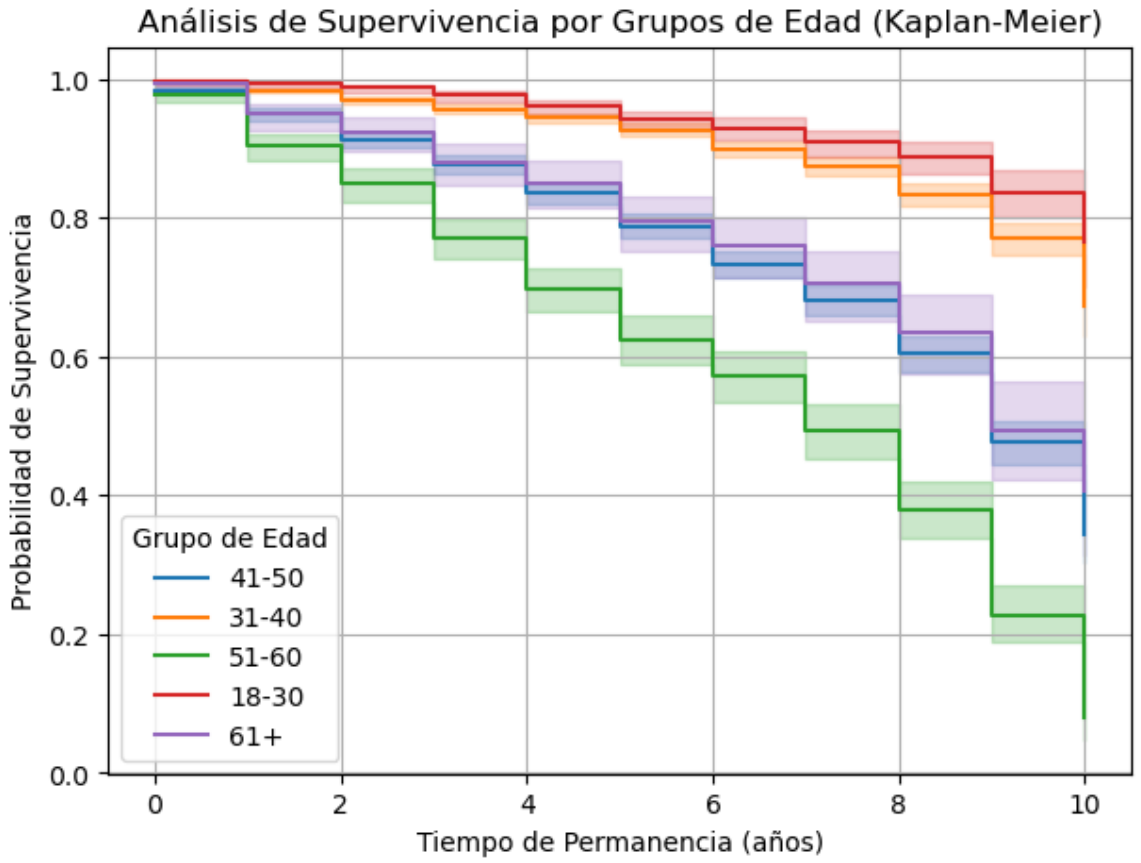
Tabla de Supervivencia - Grupo de Edad 18-30

timeline	18-30
0.0	0.996339
1.0	0.992560
2.0	0.987688
3.0	0.976800
4.0	0.959974
5.0	0.940902
6.0	0.929581
7.0	0.908105
8.0	0.888268
9.0	0.836946
10.0	0.766360

Tabla de Supervivencia - Grupo de Edad 61+

timeline	61+

```
0.0       0.992381
1.0       0.948890
2.0       0.923070
3.0       0.880137
4.0       0.850709
5.0       0.794586
6.0       0.759736
7.0       0.704866
8.0       0.634918
9.0       0.494645
10.0      0.406316
```



In [77]:

```
# Definir los rangos de ScoreCredito
bins = [ 0, 200, 400, 600, 800, 1000]
labels = ['0-200', '200-400', '400-600','600-800', '800+']
data['TotalScoreCredito'] = pd.cut(data['ScoreCredito'], bins=bins, labels=label

# Inicializar Kaplan-Meier Fitter
kmf = KaplanMeierFitter()

# Definir el tamaño de la figura
plt.figure(figsize=(7, 5))

# Obtener los grupos de Score
score_groups = data['TotalScoreCredito'].unique()


# Realizar el análisis Kaplan-Meier para cada grupo
for score_group in score_groups:
    mask_score = data['TotalScoreCredito'] == score_group
    kmf.fit(T[mask_score], event_observed=E[mask_score], label=f'Score: {score_g
    kmf.plot_survival_function()

    # Imprimir la tabla de supervivencia para cada grupo
    print(f"\nTabla de Supervivencia - Score Credito {score_group}")
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por Score Credito (Kaplan-Meier)')
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="Score Credito")
plt.show()
```

Tabla de Supervivencia - Score Credito 600-800

timeline	Score: 600-800
0.0	0.990968
1.0	0.969176
2.0	0.946420
3.0	0.919921
4.0	0.892890
5.0	0.858215
6.0	0.821868
7.0	0.778554
8.0	0.723713
9.0	0.625893
10.0	0.499914

Tabla de Supervivencia - Score Credito 400-600

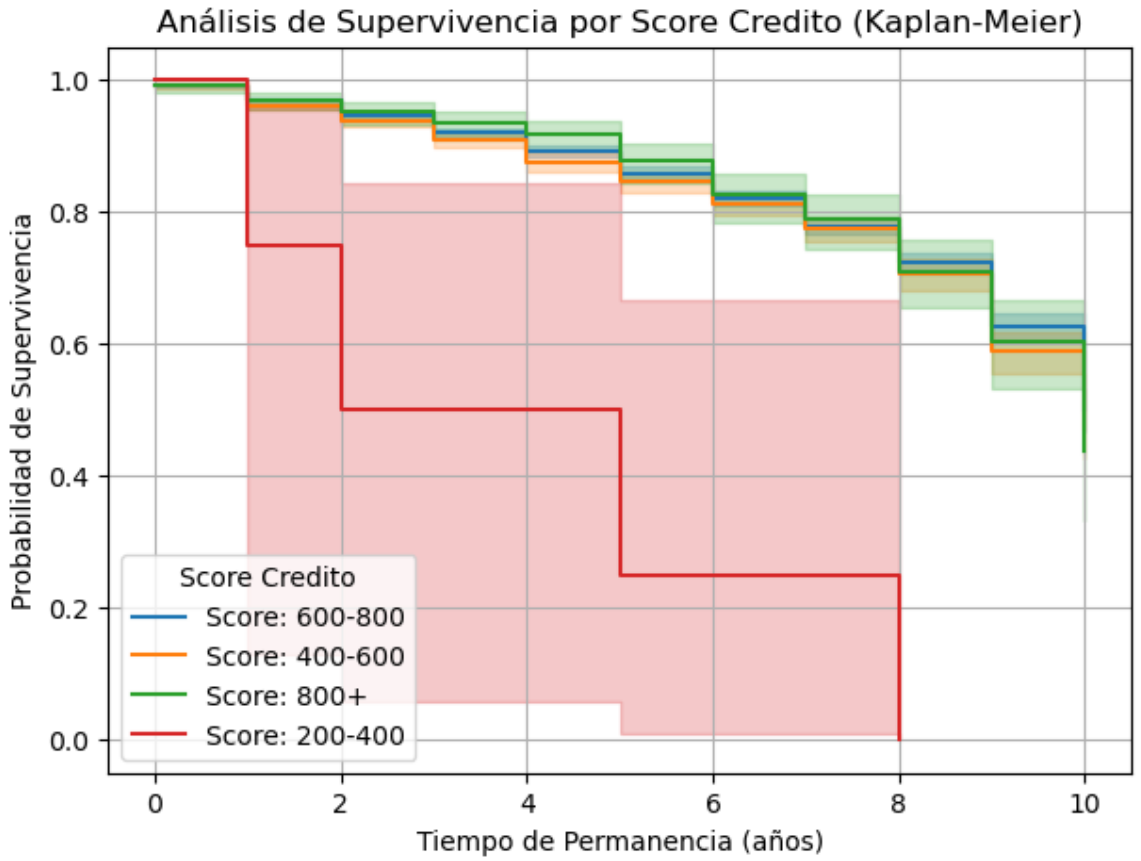
timeline	Score: 400-600
0.0	0.990713
1.0	0.961977
2.0	0.938395
3.0	0.909644
4.0	0.875520
5.0	0.845624
6.0	0.811547
7.0	0.774871
8.0	0.705091
9.0	0.587847
10.0	0.472744

Tabla de Supervivencia - Score Credito 800+

timeline	Score: 800+
0.0	0.990840
1.0	0.970329
2.0	0.953001
3.0	0.933552
4.0	0.917993
5.0	0.876619
6.0	0.825053
7.0	0.788384
8.0	0.710519
9.0	0.603559
10.0	0.438952

Tabla de Supervivencia - Score Credito 200-400

timeline	Score: 200-400
0.0	1.00
1.0	0.75
2.0	0.50
5.0	0.25
8.0	0.00



In [78]:
```
# Definir los rangos de Puntos Ganados
bins = [ 0, 200, 400, 600, 800, 1000]
labels = ['0-200', '200-400', '400-600', '600-800','800+']
data['TotalPuntosGanados'] = pd.cut(data['PuntosGanados'], bins=bins, labels=lab

# Inicializar Kaplan-Meier Fitter
kmf = KaplanMeierFitter()

# Definir el tamaño de la figura
plt.figure(figsize=(7, 5))

# Obtener los grupos de Score
puntos_groups = data['TotalScoreCredito'].unique()


# Realizar el análisis Kaplan-Meier para cada grupo
for puntos_group in puntos_groups:
    mask_puntos = data['TotalScoreCredito'] == puntos_group
    kmf.fit(T[mask_puntos], event_observed=E[mask_puntos], label=f'Puntos: {punt
    kmf.plot_survival_function()

    # Imprimir la tabla de supervivencia para cada grupo
    print(f"\nTabla de Supervivencia - Score Credito {puntos_group}")
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por Puntos Ganados (Kaplan-Meier)')
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="Puntos Ganados")
plt.show()
```

Tabla de Supervivencia - Score Credito 600-800

	Puntos: 600-800
timeline	
0.0	0.990968
1.0	0.969176
2.0	0.946420
3.0	0.919921
4.0	0.892890
5.0	0.858215
6.0	0.821868
7.0	0.778554
8.0	0.723713
9.0	0.625893
10.0	0.499914

Tabla de Supervivencia - Score Credito 400-600

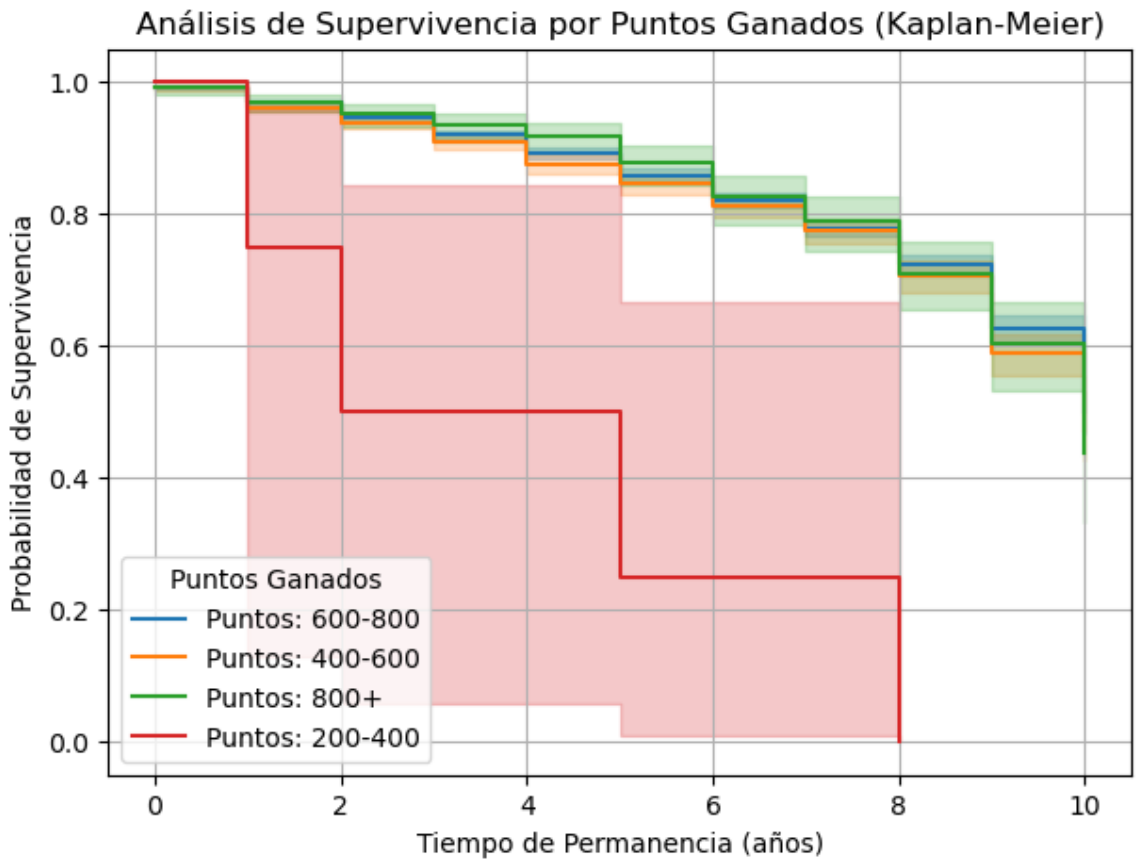
	Puntos: 400-600
timeline	
0.0	0.990713
1.0	0.961977
2.0	0.938395
3.0	0.909644
4.0	0.875520
5.0	0.845624
6.0	0.811547
7.0	0.774871
8.0	0.705091
9.0	0.587847
10.0	0.472744

Tabla de Supervivencia - Score Credito 800+

	Puntos: 800+
timeline	
0.0	0.990840
1.0	0.970329
2.0	0.953001
3.0	0.933552
4.0	0.917993
5.0	0.876619
6.0	0.825053
7.0	0.788384
8.0	0.710519
9.0	0.603559
10.0	0.438952

Tabla de Supervivencia - Score Credito 200-400

	Puntos: 200-400
timeline	
0.0	1.00
1.0	0.75
2.0	0.50
5.0	0.25
8.0	0.00



In [79]:
```
# Definir los rangos de Saldos
bins = [ 0, 50000, 100000, 150000, 200000, 300000]
labels = ['0-50K', '50K-100K', '100K-150K', '150K-200K','200K+']
data['TotalSaldo'] = pd.cut(data['Saldo'], bins=bins, labels=labels, right=False

# Inicializar Kaplan-Meier Fitter
kmf = KaplanMeierFitter()

# Definir el tamaño de la figura
plt.figure(figsize=(7, 5))

# Obtener los grupos de Saldos únicos
balance_groups = data['TotalSaldo'].unique()


# Realizar el análisis Kaplan-Meier para cada grupo de Saldos
for balance_group in balance_groups:
    mask_balance = data['TotalSaldo'] == balance_group
    kmf.fit(T[mask_balance], event_observed=E[mask_balance], label=f'Saldo: {bal
    kmf.plot_survival_function()

    # Imprimir la tabla de supervivencia para cada grupo de Saldos
    print(f"\nTabla de Supervivencia - Total Saldo {balance_group}")
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por Saldo (Kaplan-Meier)')
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="Total Saldo")
plt.show()
```

Tabla de Supervivencia - Total Saldo 0-50K

timeline	Saldo: 0-50K
0.0	0.994576
1.0	0.977201
2.0	0.956689
3.0	0.938922
4.0	0.917732
5.0	0.895055
6.0	0.866869
7.0	0.835106
8.0	0.801447
9.0	0.732458
10.0	0.615103

Tabla de Supervivencia - Total Saldo 50K-100K

timeline	Saldo: 50K-100K
0.0	0.992042
1.0	0.972064
2.0	0.954529
3.0	0.923246
4.0	0.888282
5.0	0.848409
6.0	0.807696
7.0	0.773499
8.0	0.694838
9.0	0.610921
10.0	0.514067

Tabla de Supervivencia - Total Saldo 150K-200K

timeline	Saldo: 150K-200K
0.0	0.988210
1.0	0.963810
2.0	0.936272
3.0	0.914593
4.0	0.893217
5.0	0.855858
6.0	0.816674
7.0	0.753268
8.0	0.677280
9.0	0.560699
10.0	0.445261

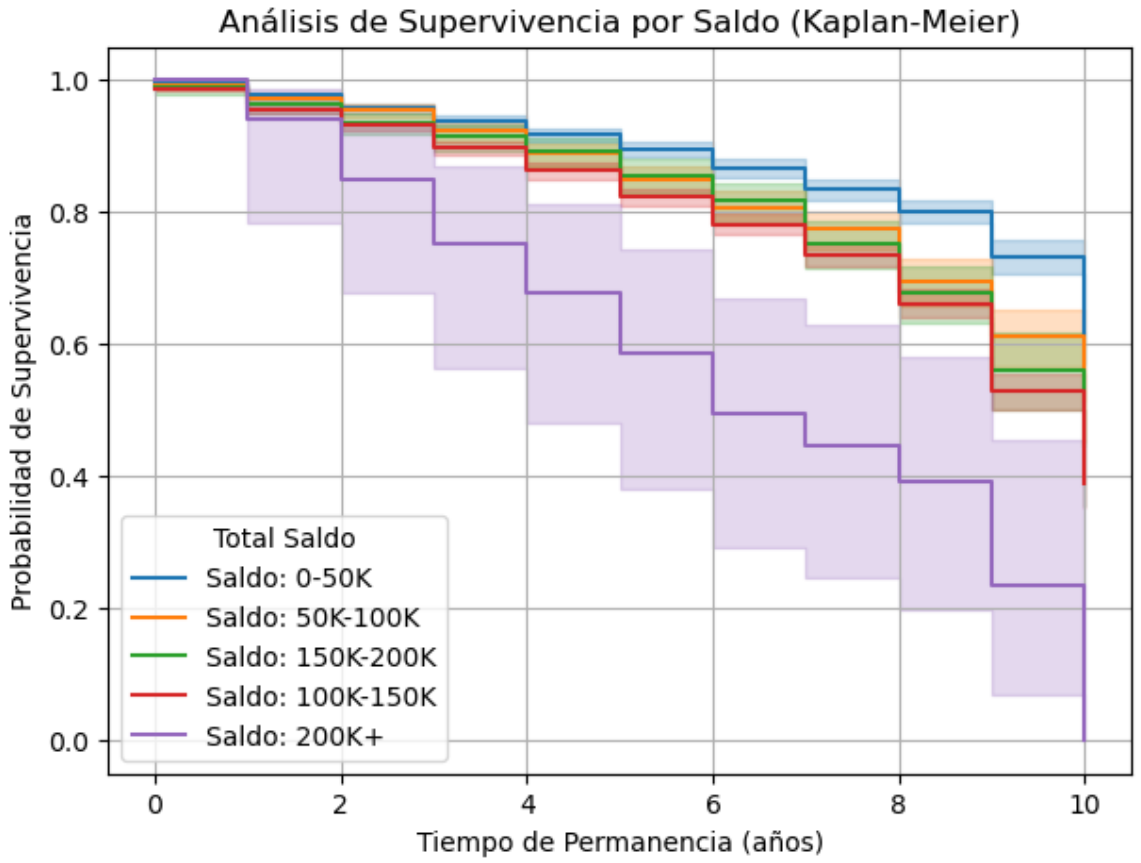
Tabla de Supervivencia - Total Saldo 100K-150K

timeline	Saldo: 100K-150K
0.0	0.987444
1.0	0.956165
2.0	0.931073
3.0	0.897073
4.0	0.863063
5.0	0.822632
6.0	0.781263
7.0	0.735605
8.0	0.661970
9.0	0.529129
10.0	0.390614

Tabla de Supervivencia - Total Saldo 200K+

timeline	Saldo: 200K+

```
0.0           1.000000
1.0           0.941176
2.0           0.850095
3.0           0.752007
4.0           0.676806
5.0           0.586565
6.0           0.496325
7.0           0.446692
8.0           0.390856
9.0           0.234513
10.0          0.000000
```



In [80]:

```
# Definir los rangos de Salario
bins = [ 0, 25000, 50000, 75000, 100000, 125000, 150000, 175000, 200000]
labels = ['0-25K', '25K-50K', '50K-75K', '75K-100K','100K-125K','125K-150K','150
data['SalarioEstimado'] = pd.to_numeric(data['SalarioEstimado'], errors='coerce'

data['TotalSalario'] = pd.cut(data['SalarioEstimado'], bins=bins, labels=labels,

# Inicializar Kaplan-Meier Fitter
kmf = KaplanMeierFitter()

# Definir el tamaño de la figura
plt.figure(figsize=(7, 5))

# Obtener los grupos de Salarios
salary_groups = data['TotalSalario'].unique()


# Realizar el análisis Kaplan-Meier para cada grupo
for salary_group in salary_groups:
    mask_salary = data['TotalSalario'] == salary_group
    kmf.fit(T[mask_salary], event_observed=E[mask_salary], label=f'Salario: {sal
    kmf.plot_survival_function(ci_show=False)  # Desactivar las bandas de confia

    # Imprimir la tabla de supervivencia para cada grupo
    print(f"\nTabla de Supervivencia - Total Salario Estimado {salary_group}")
    print(kmf.survival_function_)

# Personalizar el gráfico
plt.title('Análisis de Supervivencia por Salario Estimado(Kaplan-Meier)')
plt.xlabel('Tiempo de Permanencia (años)')
plt.ylabel('Probabilidad de Supervivencia')
plt.grid(True)
plt.legend(title="Total Salario Estimado")
plt.show()
```

Tabla de Supervivencia - Total Salario Estimado 100K-125K

timeline	Salario: 100K-125K
0.0	0.992138
1.0	0.957983
2.0	0.927627
3.0	0.908917
4.0	0.878255
5.0	0.851870
6.0	0.826055
7.0	0.783168
8.0	0.726677
9.0	0.615198
10.0	0.499848

Tabla de Supervivencia - Total Salario Estimado 75K-100K

timeline	Salario: 75K-100K
0.0	0.992891
1.0	0.973327
2.0	0.951915
3.0	0.921760
4.0	0.894582
5.0	0.866427
6.0	0.832388
7.0	0.796198
8.0	0.737939
9.0	0.631433
10.0	0.535761

Tabla de Supervivencia - Total Salario Estimado 125K-150K

timeline	Salario: 125K-150K
0.0	0.993730
1.0	0.972587
2.0	0.954070
3.0	0.924953
4.0	0.903317
5.0	0.871776
6.0	0.836011
7.0	0.793502
8.0	0.712968
9.0	0.612706
10.0	0.459530

Tabla de Supervivencia - Total Salario Estimado 0-25K

timeline	Salario: 0-25K
0.0	0.986853
1.0	0.964215
2.0	0.944764
3.0	0.915534
4.0	0.882710
5.0	0.846435
6.0	0.809160
7.0	0.775051
8.0	0.728856
9.0	0.649723
10.0	0.460712

Tabla de Supervivencia - Total Salario Estimado 50K-75K

timeline	Salario: 50K-75K

0.0	0.991332
1.0	0.965415
2.0	0.947798
3.0	0.916204
4.0	0.887118
5.0	0.856659
6.0	0.810394
7.0	0.756860
8.0	0.700157
9.0	0.605220
10.0	0.464002

Tabla de Supervivencia - Total Salario Estimado 25K-50K

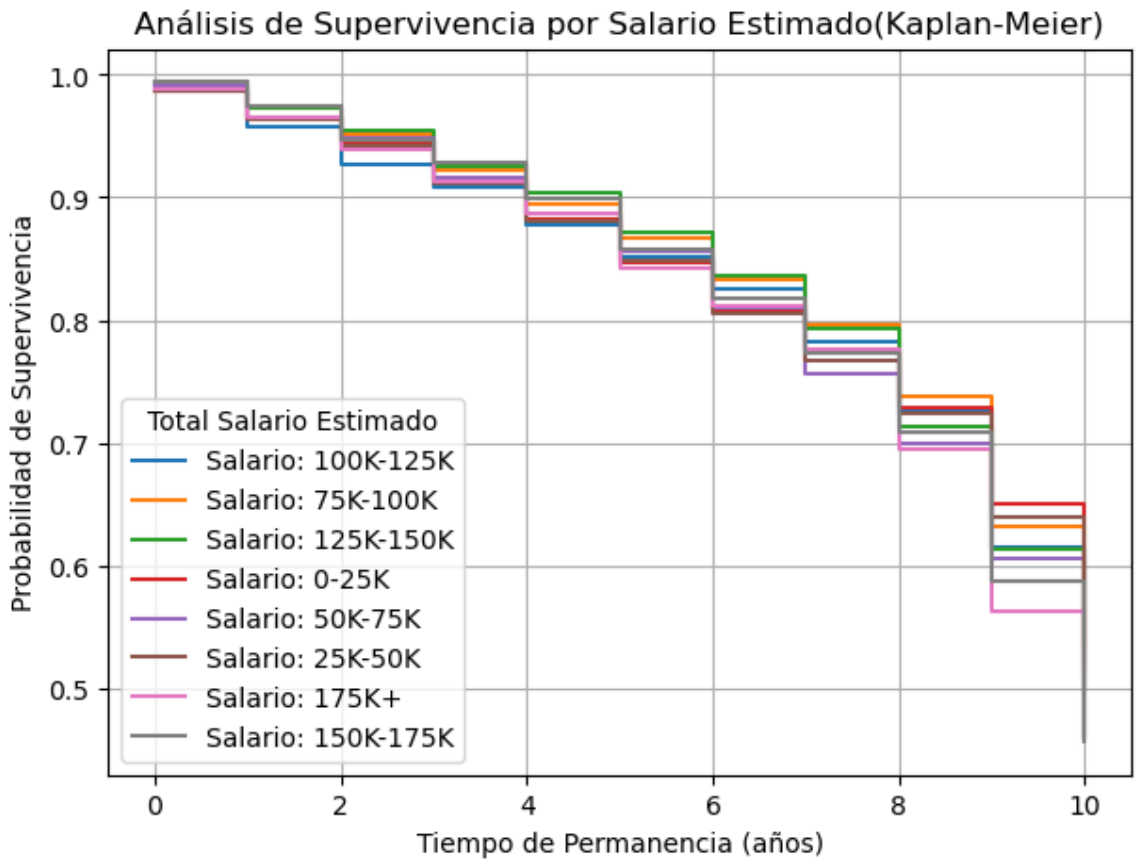
timeline	Salario: 25K-50K
0.0	0.987055
1.0	0.963514
2.0	0.941636
3.0	0.912210
4.0	0.881151
5.0	0.848125
6.0	0.804947
7.0	0.767421
8.0	0.724506
9.0	0.639723
10.0	0.516268

Tabla de Supervivencia - Total Salario Estimado 175K+

timeline	Salario: 175K+
0.0	0.988871
1.0	0.964993
2.0	0.939527
3.0	0.912309
4.0	0.886404
5.0	0.841701
6.0	0.811225
7.0	0.776183
8.0	0.695231
9.0	0.562264
10.0	0.506037

Tabla de Supervivencia - Total Salario Estimado 150K-175K

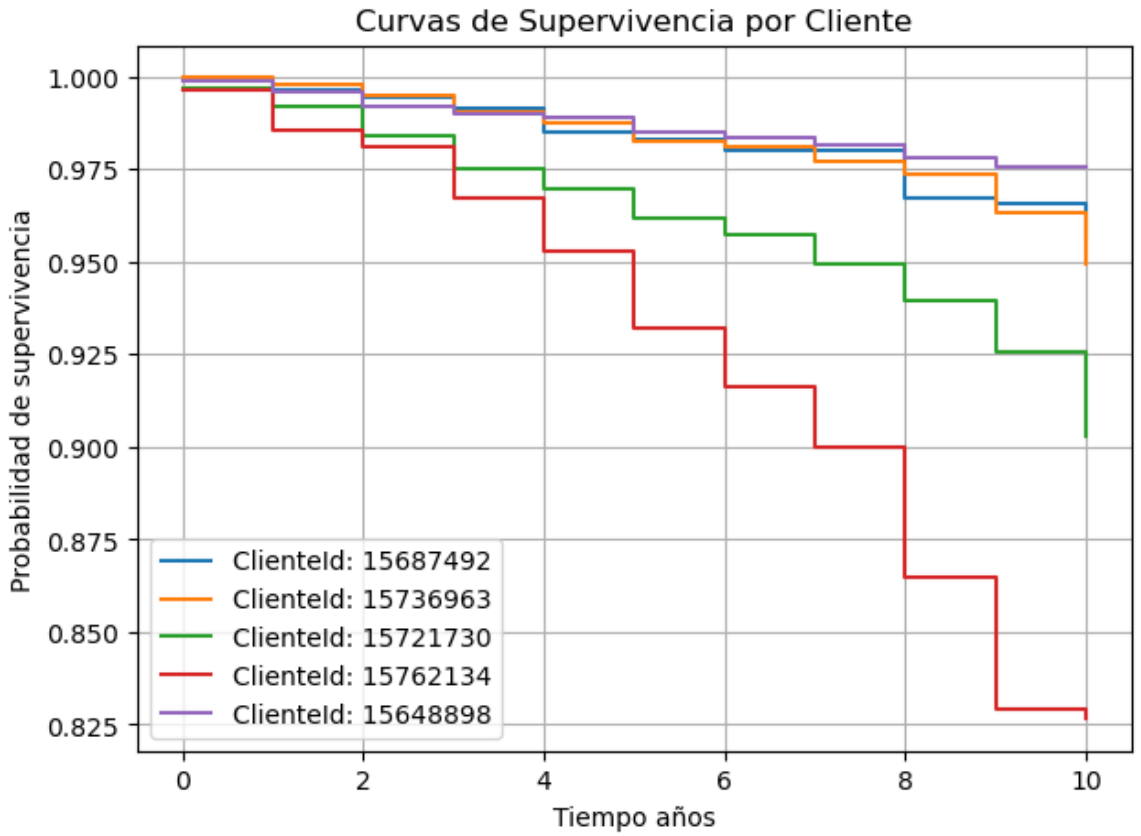
timeline	Salario: 150K-175K
0.0	0.994123
1.0	0.974066
2.0	0.946372
3.0	0.928436
4.0	0.898830
5.0	0.858464
6.0	0.818122
7.0	0.772671
8.0	0.708726
9.0	0.588092
10.0	0.456429



ANALISIS DE SUPERVIVENCIA POR CLIENTE

In [81]:

```
from sksurv.preprocessing import OneHotEncoder

#variables X_test e y_test
data_x=X_test
data_y=y_test


# Obtener los IDs de los clientes de X_test
id_cliente = X_test['IdCliente'].values  # Lista de CustomerId de los 3000 regis


# Convertir columnas categóricas a tipo 'category'
categorical_columns = data_x.select_dtypes(include=['object']).columns
data_x[categorical_columns] = data_x[categorical_columns].astype('category')

# Codificación one-hot para variables categóricas
encoder = OneHotEncoder()
data_x_encoded = encoder.fit_transform(data_x)

# Ajustar el modelo Random Survival Forest
rsf = RandomSurvivalForest(n_estimators=100, min_samples_split=10, min_samples_l
rsf.fit(data_x_encoded, data_y)

# Obtener las curvas de supervivencia para cada nodo hoja
survival_curves = rsf.predict_survival_function(data_x_encoded)

# Graficar las curvas de supervivencia para algunos de los nodos hoja
plt.figure(figsize=(7, 5))
v1=0
v2=5
# Iterar sobre algunas curvas y graficarlas
ids = [] # Lista de los Ids de clientes que quieres buscar

for i, surv_func in enumerate(survival_curves[v1:v2]):  # Grafica las primeras 5
    time_points = surv_func.x
    survival_prob = surv_func.y
    plt.step(time_points, survival_prob, where="post", label=f'ClienteId: {id_cl
    ids.append(id_cliente[v1+i])
plt.xlabel('Tiempo años')
plt.ylabel('Probabilidad de supervivencia')
plt.title('Curvas de Supervivencia por Cliente')
plt.legend()
plt.grid(True)
plt.show()

#clientes = data_x.query('IdCliente in @ids')
clientes = data.query('IdCliente in @ids')
clientes
```



Out[81]:

	IdCliente	ScoreCredito	Pais	Genero	Edad	Antiguedad	Saldo	NroDeProducto
1731	15721730	601	Spain	Female	44	4	0.00	
4521	15648898	560	Spain	Female	27	7	124995.98	
4684	15736963	623	France	Male	43	1	0.00	
4742	15762134	506	Germany	Male	59	8	119152.10	
6252	15687492	596	Germany	Male	32	3	96709.07	

5 rows × 21 columns

In [82]:
```
# Buscar dato específico
resultado = data[data['IdCliente'] == 15762134]
print(resultado)
```

```
      IdCliente  ScoreCredito     Pais Genero  Edad  Antiguedad     Saldo  \
4742   15762134           506  Germany   Male    59           8  119152.1

      NroDeProductos  SiTieneTarjeta  EsMiembroActivo  ...  Desertor  \
4742               2               1                1  ...         0

      Reclamos  CalificacionSatisfaccion  TipoTarjeta PuntosGanados  AgeGroup  \
4742         0                         2       SILVER           979     51-60

     TotalScoreCredito TotalPuntosGanados TotalSaldo TotalSalario
4742           400-600               800+  100K-150K    150K-175K

[1 rows x 21 columns]
```

ESTRATEGIAS

- Desarrollar productos y servicios personalizados de acuerdo a la edad.
- Crear o reforzar programas de retención en los clientes.
- Incentivar a los clientes a adquirir múltiples productos.
- Implementar o reforzar encuestas de satisfacción y recomendación.

INNOVACION Y COMPETITIVIDAD EMPRESARIAL

- Generar productos y servicios de alto valor a través de procesos más ágiles e innovación.
- Mejora continua en la experiencia digital del cliente.
- Excelencia operacional y digital.

CONCLUSIONES

- Random Forest es el mejor modelo de clasificación en la predicción de deserción de clientes
- La edad, el número de productos, el saldo, puntos ganados y el salario abarcan el 66% del comportamiento de deserción
- Edad es la variable que tiene mayor porcentaje de impacto al tener un 21.78%.
- Importancia de segmentar a los clientes por estas variables implementar estrategias personalizadas, lo que maximiza la efectividad de las acciones para disminuir el porcentaje de deserción. Optimizando los recursos destinados a la retención de clientes.
- Existió limitaciones debido al a falta de datos relevantes se recomienda:

  Inclusión de variables relevantes (nro transacciones, estado civil, nro de dependientes)

  Actualizar y perfeccionar regularmente el modelo

  Para mejorar continuamente la precisión y capacidad predictiva del modelo ya que el comportamiento del cliente también varía con el tiempo.

RECOMENDACIONES

- Mejorar la precisión del modelo mediante la inclusión de variables adicionales relevantes como: número de reclamos, número de transacciones, estado civil, ocupación, número de dependencias y canal de atención.
- Desarrollar productos y servicios financieros personalizados en base a las cinco variables de mayor impacto en la deserción.
- Fortalecer programas de retención a través de campañas de descuentos y promociones ofreciendo recompensas por el uso de productos financieros.
- Optimizar la experiencia digital,
- Actualizar y perfeccionar regularmente el modelo de análisis.