Introduccion

Realizaremos un análisis exploratorio de los datos de la compañía Interconnect. El propósito de la empresa es poder pronosticar la tasa de cancelación de clientes.

Plan

1. Preparación del Ambiente

- Cargar y verificar la integridad de los datos.
- Realizar las conversiones necesarias en los tipos de datos.
- Unir los conjuntos de datos en un único DataFrame.

1. Análisis Exploratorio de Datos (EDA)

- Describir las estadísticas generales de los datos.
- Visualizar distribuciones de variables importantes (e.g., MonthlyCharges, TotalCharges, SeniorCitizen).
- Explorar correlaciones entre variables mediante mapas de calor.
- Realizar análisis de contingencia para evaluar relaciones significativas (e.g., entre MonthlyCharges y StreamingTV).

1. Preparación de los Datos para el Modelado

- Crear nuevas características (e.g., ContractDuration, ActiveContract).
- Convertir variables categóricas a variables numéricas (codificación binaria).
- Manejar valores nulos (NaN), especialmente en TotalCharges y otras columnas relevantes.

1. Dividir los Datos en Conjuntos de Entrenamiento y Prueba

- Realizar la división de los datos en conjuntos de entrenamiento y prueba con una proporción adecuada (e.g., 80% entrenamiento, 20% prueba).

1. Desarrollo del Modelo

- Seleccionar y entrenar varios modelos de clasificación (e.g., regresión logística, árboles de decisión, Random Forest, XGBoost).
- Evaluar el desempeño de cada modelo utilizando la métrica AUC-ROC y exactitud.

1. Evaluación y Selección del Modelo

- Comparar los modelos basados en AUC-ROC y seleccionar el mejor modelo.
- Realizar validación cruzada para garantizar la robustez del modelo seleccionado.

1. Preparar el Informe

- Documentar todo el proceso, desde el EDA hasta la selección del modelo.
- Incluir visualizaciones y explicaciones claras de los resultados obtenidos.
- Proporcionar recomendaciones basadas en los hallazgos del análisis y modelado.

Primeramente, observamos la relación - entidad entre los conjuntos de datos: 'customerID' es el atributo que estos comparten, por lo que es lo que los relaciona de la siguiente manera:

1. Un cliente puede tener un contrato.
2. Un contrato está asociado a un cliente.
3. Un cliente puede tener varios servicios de internet.
4. Un cliente puede tener varios servicios telefónicos.

Esto se traduce a:

- El conjunto 'client' tiene una relación de uno a uno con el conjunto 'contract'.
- El conjunto 'client' tiene una relación de uno a muchos con el conjunto 'internet'.
- El conjunto 'client' tiene una relación de uno a muchos con el conjunto 'phone'.

Inicialización

Importamos las librerias necesarias para ralizar el trabajo.

In [ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings

from scipy.stats import chi2_contingency

from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from xgboost import XGBClassifier

from keras.models import Sequential
from keras.layers import Dense, Dropout, LSTM
from keras.callbacks import EarlyStopping
import tensorflow as tf

from imblearn.over_sampling import SMOTE
```

Inicialización de variables globales

In [ ]:
```
random_state = np.random.RandomState(417)
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
```

EDA

Comenzamos con el análisis exploratorio de los datos; primero cargamos los conjuntos y echamos un vistazo a estos.

In [ ]:
```
contract_df = pd.read_csv('final_provider\\final_provider\\contract.csv')
personal_df = pd.read_csv('final_provider\\final_provider\\personal.csv')
internet_df = pd.read_csv('final_provider\\final_provider\\internet.csv')
phone_df = pd.read_csv('final_provider\\final_provider\\phone.csv')
```

In [ ]:
```
contract_df.sample(n=5, random_state=117)
```

Out[ ]:

	customerID	BeginDate	EndDate	Type	PaperlessBilling	PaymentMethod	MonthlyCharges
6921	5243-SAOTC	2015-08-01	No	Month-to-month	Yes	Bank transfer (automatic)	79.85
6802	5626-MGTUK	2018-06-01	No	Month-to-month	Yes	Bank transfer (automatic)	89.10
1913	7188-CBBBA	2014-09-01	No	One year	Yes	Electronic check	95.50
4201	1166-PQLGG	2014-02-01	No	Two year	No	Bank transfer (automatic)	19.55
3857	3675-EQOZA	2019-09-01	No	Month-to-month	No	Bank transfer (automatic)	20.65

La información de los contratos con fechas anteriores al 1 de febrero de 2020 no es válida. Podemos ver aquellos contratos que ya están terminados (y su fecha de terminación) y los que aún no. Esto es importante para el tipo de predicción que la compañía desea hacer. También podemos observar los tipos de contratos (períodos), si reciben la factura por papel o no, el método de pago, los cargos mensuales y los cargos totales (por si el cliente tiene más de un contrato).

In [ ]:
```
personal_df.sample(n=5, random_state=117)
```

Out[ ]:

	customerID	gender	SeniorCitizen	Partner	Dependents
6921	5243-SAOTC	Male	0	No	No
6802	5626-MGTUK	Female	0	No	No
1913	7188-CBBBA	Female	0	No	No
4201	1166-PQLGG	Female	0	Yes	Yes
3857	3675-EQOZA	Male	0	No	No

Informacion general del cliente, como su genero, si es una persona de la tercera edad, si esta casado y si tiene dependientes.

In [ ]:
```
internet_df.sample(n=5, random_state=117)
```

Out[ ]:

	customerID	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	Stre
1809	4706-DGAHW	DSL	No	No	No	No	
4752	4024-CSNBY	Fiber optic	Yes	Yes	Yes	Yes	
3196	5606-AMZBO	Fiber optic	Yes	No	No	No	
5428	2314-TNDJQ	DSL	Yes	Yes	Yes	Yes	
770	5651-YLPRD	Fiber optic	No	Yes	No	No	

Si el cliente tiene mas de un servicio de internet lo podemos ver aqui, igual el tipo de servicio general.

In [ ]:
```
phone_df.sample(n=5, random_state=117)
```

Out[ ]:

	customerID	MultipleLines
5519	8775-LHDJH	Yes
2499	5074-FBGHB	Yes
1560	6303-KFWSL	Yes
5647	6394-HHHZM	No
5961	3118-UHVVQ	Yes

Aqui simplemente confirmamos si el cliente tiene tiene múltiples líneas telefónicas.

Pasamos a revisar la informacion de cada dataframe.

In [ ]:
```
contract_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   BeginDate         7043 non-null   object 
 2   EndDate           7043 non-null   object 
 3   Type              7043 non-null   object 
 4   PaperlessBilling  7043 non-null   object 
 5   PaymentMethod     7043 non-null   object 
 6   MonthlyCharges    7043 non-null   float64
 7   TotalCharges      7043 non-null   object 
dtypes: float64(1), object(7)
memory usage: 440.3+ KB
```

Las columnas 'BeginDate', 'EndDate' y 'TotalCharges' tienen el tipo de dato equivocado. Vamos a manejarlas:

In [ ]:
```
contract_df['BeginDate'] = pd.to_datetime(contract_df['BeginDate'])
contract_df['EndDate'] = contract_df['EndDate'].replace('No', pd.NaT)
contract_df['EndDate'] = pd.to_datetime(contract_df['EndDate'])
contract_df['TotalCharges'] = pd.to_numeric(contract_df['TotalCharges'], errors='coerc
valores_problematicos = contract_df[contract_df['TotalCharges'].isna()]
valores_problematicos.size
```

Out[ ]:
```
88
```

In [ ]:
```
reference_date = pd.to_datetime('2020-02-01')
m = contract_df[contract_df['BeginDate'] >= reference_date]
print(m.size)
m
```

```
88
```

Out[ ]:

	customerID	BeginDate	EndDate	Type	PaperlessBilling	PaymentMethod	MonthlyCharges	Tot
488	4472-LVYGI	2020-02-01	NaT	Two year	Yes	Bank transfer (automatic)	52.55	
753	3115-CZMZD	2020-02-01	NaT	Two year	No	Mailed check	20.25	
936	5709-LVOEQ	2020-02-01	NaT	Two year	No	Mailed check	80.85	
1082	4367-NUYAO	2020-02-01	NaT	Two year	No	Mailed check	25.75	
1340	1371-DWPAZ	2020-02-01	NaT	Two year	No	Credit card (automatic)	56.05	
3331	7644-OMVMY	2020-02-01	NaT	Two year	No	Mailed check	19.85	
3826	3213-VVOLG	2020-02-01	NaT	Two year	No	Mailed check	25.35	
4380	2520-SGTTA	2020-02-01	NaT	Two year	No	Mailed check	20.00	
5218	2923-ARZLG	2020-02-01	NaT	One year	Yes	Mailed check	19.70	
6670	4075-WKNIU	2020-02-01	NaT	Two year	No	Mailed check	73.35	
6754	2775-SEFEE	2020-02-01	NaT	Two year	Yes	Bank transfer (automatic)	61.90	

In [ ]:
```
valores_problematicos
```

Out[ ]:

	customerID	BeginDate	EndDate	Type	PaperlessBilling	PaymentMethod	MonthlyCharges	Tot
488	4472-LVYGI	2020-02-01	NaT	Two year	Yes	Bank transfer (automatic)	52.55	
753	3115-CZMZD	2020-02-01	NaT	Two year	No	Mailed check	20.25	
936	5709-LVOEQ	2020-02-01	NaT	Two year	No	Mailed check	80.85	
1082	4367-NUYAO	2020-02-01	NaT	Two year	No	Mailed check	25.75	
1340	1371-DWPAZ	2020-02-01	NaT	Two year	No	Credit card (automatic)	56.05	
3331	7644-OMVMY	2020-02-01	NaT	Two year	No	Mailed check	19.85	
3826	3213-VVOLG	2020-02-01	NaT	Two year	No	Mailed check	25.35	
4380	2520-SGTTA	2020-02-01	NaT	Two year	No	Mailed check	20.00	
5218	2923-ARZLG	2020-02-01	NaT	One year	Yes	Mailed check	19.70	
6670	4075-WKNIU	2020-02-01	NaT	Two year	No	Mailed check	73.35	
6754	2775-SEFEE	2020-02-01	NaT	Two year	Yes	Bank transfer (automatic)	61.90	

Los valores 'No' en 'EndDate' se sustituyeron por 'NaT' (not a time).

Nos encontramos con que muchos de los valores en 'TotalCharges' están vacíos (NaN); los cambiaremos a 0.

Los mismo datos que son a partir de 02/01/2020 son los que les falta los cargos totales.

In [ ]:
```
contract_df['TotalCharges'] = contract_df['TotalCharges'].fillna(0).astype(float)
```

In [ ]:
```
personal_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   customerID     7043 non-null   object
 1   gender         7043 non-null   object
 2   SeniorCitizen  7043 non-null   int64 
 3   Partner        7043 non-null   object
 4   Dependents     7043 non-null   object
dtypes: int64(1), object(4)
memory usage: 275.2+ KB
```

Los tipos de datos son correctos, no hay valores nulos.

In [ ]:
```
internet_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5517 entries, 0 to 5516
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype 
---  ------            --------------  ----- 
 0   customerID        5517 non-null   object
 1   InternetService   5517 non-null   object
 2   OnlineSecurity    5517 non-null   object
 3   OnlineBackup      5517 non-null   object
 4   DeviceProtection  5517 non-null   object
 5   TechSupport       5517 non-null   object
 6   StreamingTV       5517 non-null   object
 7   StreamingMovies   5517 non-null   object
dtypes: object(8)
memory usage: 344.9+ KB
```

Los tipos de datos son correctos, no hay valores nulos.

In [ ]:
```
phone_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6361 entries, 0 to 6360
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   customerID     6361 non-null   object
 1   MultipleLines  6361 non-null   object
dtypes: object(2)
memory usage: 99.5+ KB
```

Los tipos de datos son correctos, no hay valores nulos.

Después pasamos a tener una descripción estadística de algunos datos, como los gastos mensuales y totales por cliente, el servicio de internet más popular, si la mayoría o minoría de los clientes es mayor de edad o están casados, o si la mayoría de estos tiene múltiples líneas o no.

In [ ]:
```
print(contract_df[['MonthlyCharges','TotalCharges']].describe())
print()
print(personal_df.describe())
print()
print(internet_df['InternetService'].describe())
print()
print(phone_df['MultipleLines'].describe())
```

```
       MonthlyCharges  TotalCharges
count     7043.000000   7043.000000
mean        64.761692   2279.734304
std         30.090047   2266.794470
min         18.250000      0.000000
25%         35.500000    398.550000
50%         70.350000   1394.550000
75%         89.850000   3786.600000
max        118.750000   8684.800000

       SeniorCitizen
count    7043.000000
mean        0.162147
std         0.368612
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000

count            5517
unique              2
top       Fiber optic
freq             3096
Name: InternetService, dtype: object

count     6361
unique       2
top         No
freq      3390
Name: MultipleLines, dtype: object
```

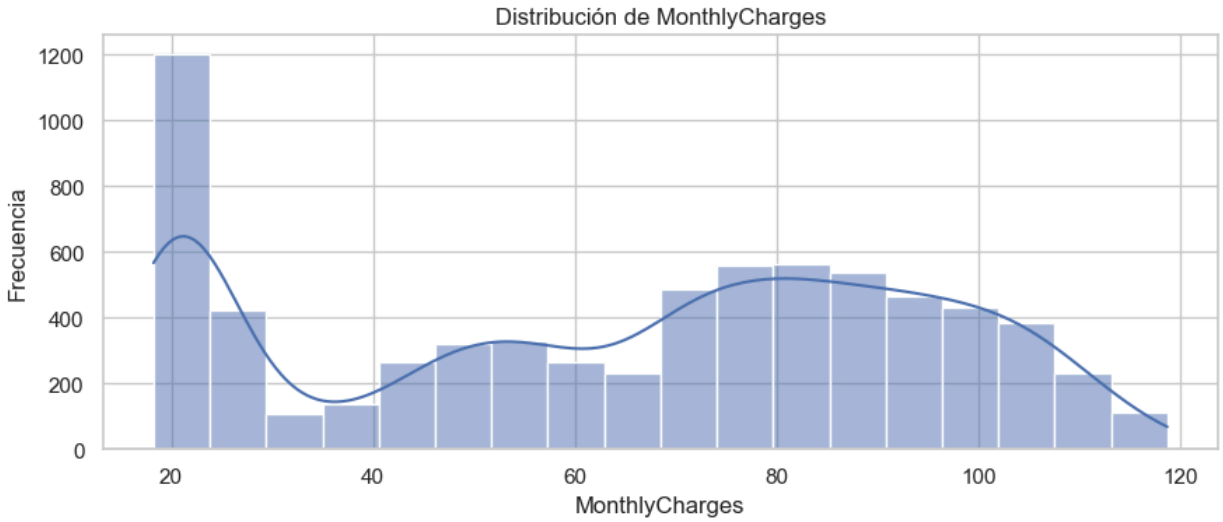
Nos damos una idea de la variabilidad en los datos, como que hubo clientes que nomás estuvieron un mes (cargo total igual al cargo mensual). Vemos que la mayoría de la gente paga entre 70 y 90 dólares al mes. La mayoría de los clientes no son de la tercera edad. La mayoría de los clientes prefiere fibra óptica o que también solo cuentan con una línea telefónica.

Distribuciones

Definimos una función que nos ayudará a graficar.

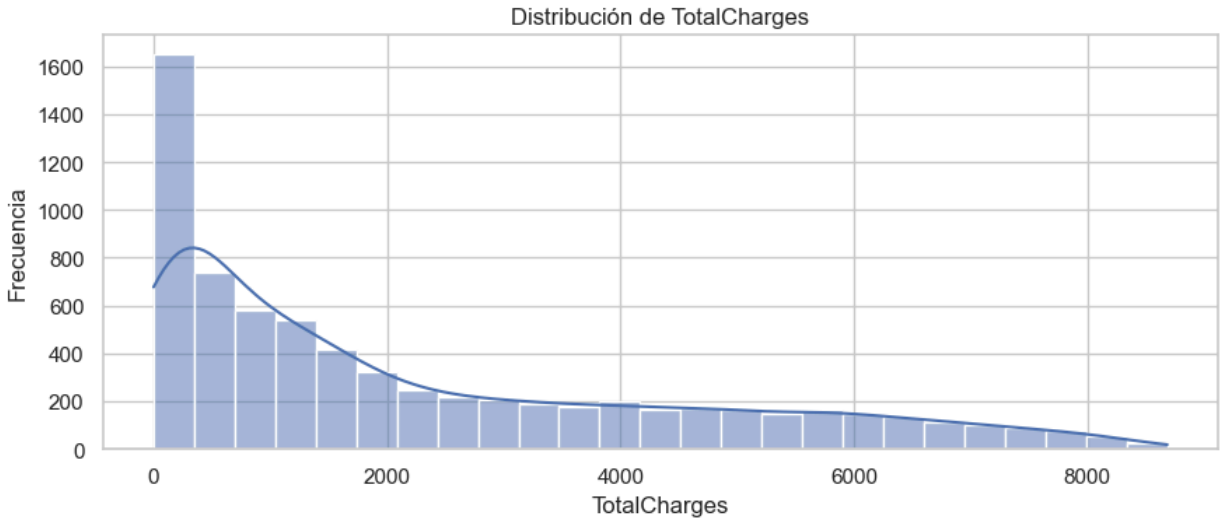
In [ ]:
```python
def distribution_plot(df, column):
    sns.set_theme(style="whitegrid")
    plt.figure(figsize=(9, 4))

    if column in ('InternetService', 'MultipleLines'):
        sns.countplot(x=column, data=df)
    else:
        sns.histplot(df[column], kde=True)

    plt.title(f'Distribución de {column}')
    plt.xlabel(column)
    plt.ylabel('Frecuencia')
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
distribution_plot(contract_df, 'MonthlyCharges')
```



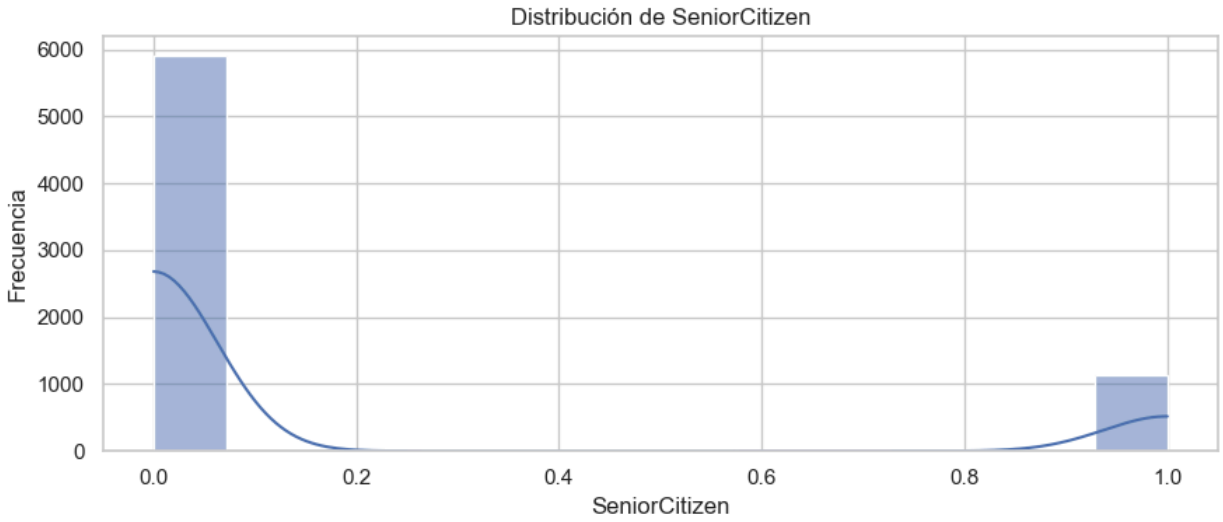
La distribución de gastos mensuales es uniforme y con un pico al inicio; la mayoría de los usuarios solo pagan el plan mínimo.

In [ ]:
```
distribution_plot(contract_df, 'TotalCharges')
```



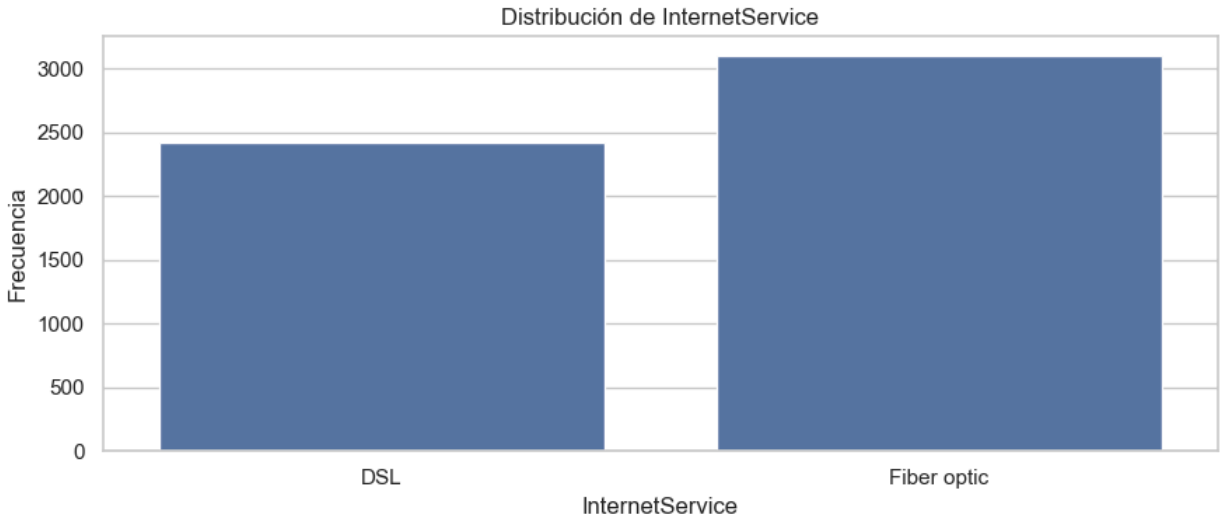
Un pico al inicio del gráfico en los gastos totales por cliente podría indicar que muchos de estos solo prueban el servicio por un par de meses antes de darse de baja. También puede indicar que hay muchos usuarios nuevos al momento de recolectar esta información.

In [ ]:
```
distribution_plot(personal_df, 'SeniorCitizen')
```



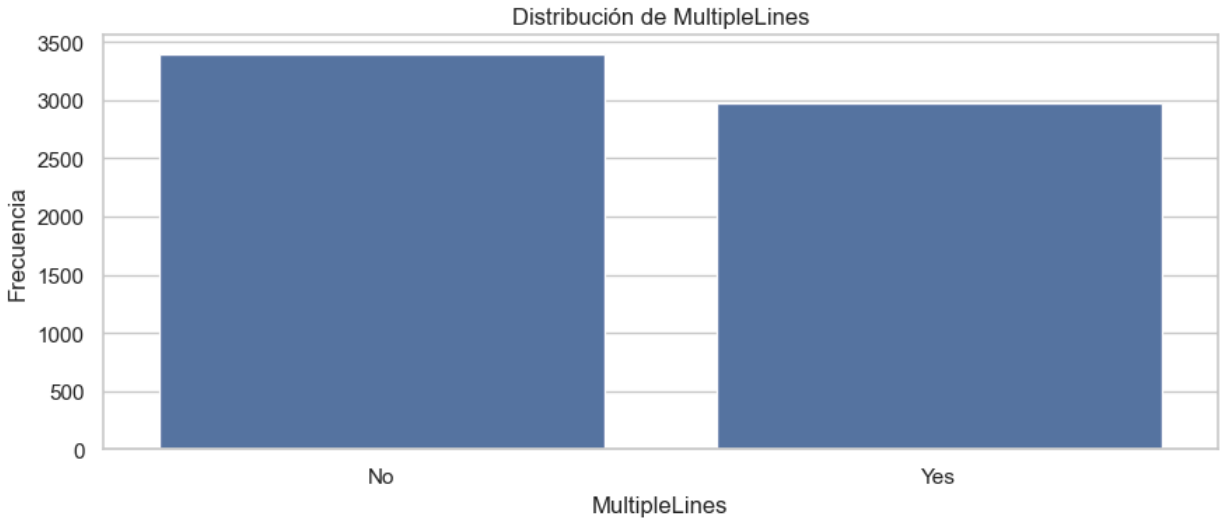
Una diferencia de casi x6 en las diferencias de edades, tal vez sea momento de hacer promociones entre personas de la tercera edad.

In [ ]:
```
distribution_plot(internet_df, 'InternetService')
```



La preferencia en tipos de servicio de internet es poca, siendo la fibra óptica la favorita.

In [ ]:
```
distribution_plot(phone_df, 'MultipleLines')
```



Aquí la diferencia es menor a la comparación anterior; el número de clientes con múltiples líneas telefónicas es casi igual al de los que no. ¿Tal vez podríamos motivar a los clientes que no tienen múltiples líneas a que contraten otra línea?

Relaciones entre tablas

Investiguemos las relaciones entre los datos de distintos conjuntos usando la clave 'customerID'. Primero, crearemos un único dataframe.

In [ ]:
```
merged_df = contract_df.merge(personal_df, on='customerID', how='left') \
                       .merge(internet_df, on='customerID', how='left') \
                       .merge(phone_df, on='customerID', how='left')

merged_df.sample(n=10, random_state=random_state)
```

Out[ ]:

	customerID	BeginDate	EndDate	Type	PaperlessBilling	PaymentMethod	MonthlyCharges
6139	3039-MJSLN	2019-11-01	NaT	Month-to-month	No	Bank transfer (automatic)	20.20
1208	5494-HECPR	2019-10-01	2019-11-01	Month-to-month	Yes	Electronic check	80.25
1606	6374-NTQLP	2014-02-01	NaT	Two year	Yes	Credit card (automatic)	104.10
2949	3489-VSFRD	2015-06-01	NaT	One year	Yes	Credit card (automatic)	60.25
2980	5598-IKHQQ	2014-02-01	NaT	Two year	No	Credit card (automatic)	25.45
3725	0968-GSIKN	2019-10-01	2019-11-01	Month-to-month	Yes	Mailed check	70.80
7024	7398-LXGYX	2016-06-01	NaT	Month-to-month	Yes	Credit card (automatic)	84.80
3367	7109-CQYUZ	2015-10-01	NaT	Two year	Yes	Mailed check	89.25
3674	7825-GKXMW	2019-12-01	2020-01-01	Month-to-month	No	Electronic check	45.80
574	6030-REHUX	2017-10-01	NaT	Month-to-month	Yes	Electronic check	110.85

Convertimos todos los valores NaN a la cadena 'No' en aquellas entradas donde el cliente no tiene múltiples líneas y no tienen servicio de internet.

In [ ]:
```
merged_df.fillna('No', inplace=True)

# Realizamos la conversion de nuevo ya que la operacion anterior convierte el tipo de
merged_df['EndDate'] = merged_df['EndDate'].replace('No', pd.NaT)
merged_df['EndDate'] = pd.to_datetime(merged_df['EndDate'])
```

Análisis de Correlación

Realicemos un análisis de correlación con el dataframe combinado.

Primero codificaremos las características categóricas para poder crear una matriz de correlación. Agregaremos una nueva columna que cuenta el número de días de duración del contrato del cliente, así podremos relacionar la duración del contrato con las demás variables. Después, realizaremos una codificación binaria a la columna 'EndDate', en donde 1 será que el contrato sigue activo y 0 si finalizó. Así podremos obtener más información.

In [ ]:
```
merged_df_encoded = merged_df.copy()
merged_df_encoded['ActiveContract'] = merged_df_encoded['EndDate'].isna().astype(int)
merged_df_encoded['ContractDuration'] = (merged_df_encoded['EndDate'] - merged_df_encc
merged_df_encoded['ContractDuration'].fillna(-1, inplace=True)  # -1 indica contrato a

merged_df_encoded = merged_df_encoded.drop(columns=['customerID', 'BeginDate', 'EndDat

categorical_columns = ['gender', 'Partner', 'Dependents', 'InternetService', 'OnlineSe
                       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV
                       'StreamingMovies', 'MultipleLines', 'PaperlessBilling', 'Paymer

for column in categorical_columns:
    merged_df_encoded[column] = merged_df_encoded[column].astype('category').cat.codes
```

In [ ]:
```
merged_df_encoded.sample(n=10, random_state=random_state)
```

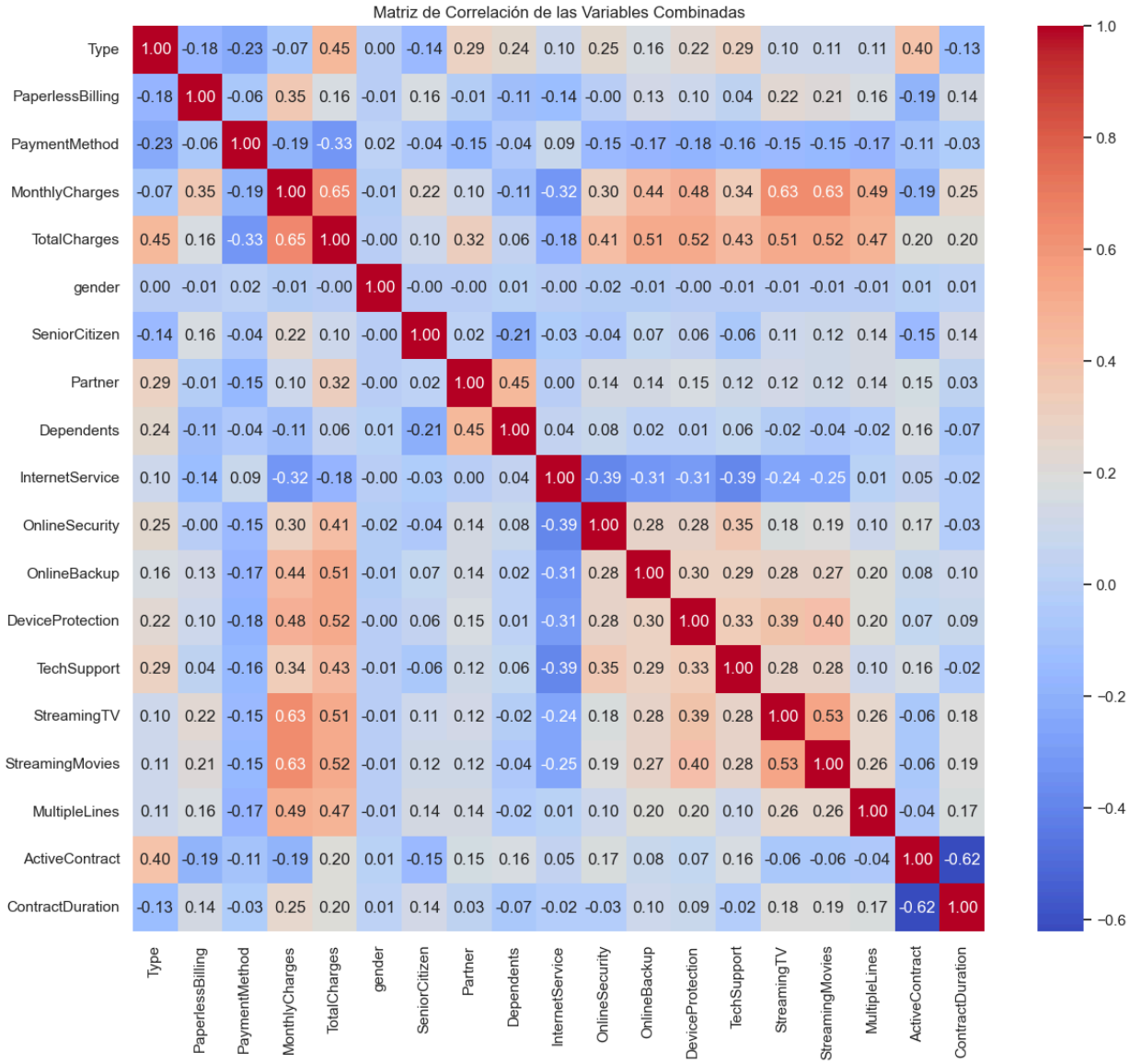
Out[ ]:

	Type	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges	gender	SeniorCitizen
4776	0	1	2	68.95	351.50	0	0
3310	0	0	2	78.45	78.45	0	1
5273	2	0	0	20.15	1337.50	0	0
549	0	0	3	19.75	311.60	1	0
5031	0	1	2	101.50	906.85	1	0
1901	1	0	2	20.15	260.70	0	0
1082	2	0	3	25.75	0.00	1	0
283	0	0	3	54.45	3687.75	0	0
3198	1	1	0	106.00	4178.65	1	0
6717	0	0	2	100.60	5746.15	1	1

In [ ]:
```
correlation_matrix = merged_df_encoded.corr()
```

Crearemos un mapa de calor para las correlaciones entre las características del dataframe combinado.

In [ ]:
```
plt.figure(figsize=(14, 12))
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Matriz de Correlación de las Variables Combinadas')
plt.show()
```



Observamos cómo se relacionan las variables; el área donde hay más calor es en las relacionadas con los servicios extras y los cargos mensuales y totales, lo que se traduce a más servicios, más pagas. También podemos ver una correlación importante entre los servicios de streaming de TV o películas, podemos entender que si el cliente contrata uno es probable que contrate el otro, así podemos introducir estrategias de marketing.

Podemos ver también la correlación negativa con mayor impacto, que es entre el servicio de internet y el soporte técnico. Esto podría indicar que clientes con ciertos tipos de servicios de internet tienden a no contratar el soporte técnico; puede ser que un servicio sea de mayor calidad que el otro (fibra óptica).

Hay caracteristicas con baja correlacion con el resto, otras que tienen una alta correlacion entre ellas (negativa o positiva). Estas podrian no ser importantes para entrenar el modelo. Estos resultados son importantes para el desarrollo posterior.

Análisis de Contingencia

Teniendo una mejor idea de las correlaciones entre las características, podemos realizar análisis de contingencia utilizando el estadístico chi cuadrado entre dos variables de interés. Lo haremos con los cargos mensuales y el servicio extra de TV.

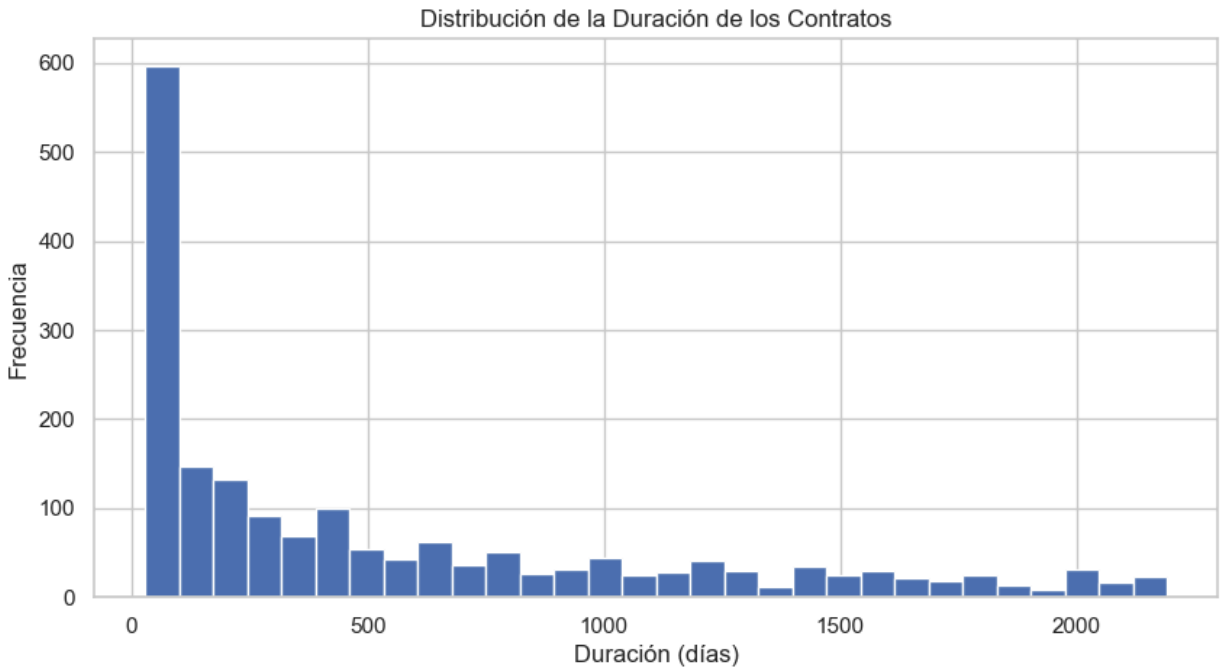
In [ ]:
```
# Función para calcular el estadístico Chi-cuadrado entre dos variables categóricas
def chi2_test_of_independence(df, col1, col2):
    contingency_table = pd.crosstab(df[col1], df[col2])
    chi2, p, dof, expected = chi2_contingency(contingency_table)
    return chi2, p

chi2, p = chi2_test_of_independence(merged_df, 'MonthlyCharges', 'StreamingTV')
print(f'Chi-cuadrado: {chi2}, p-valor: {p}')
```

```
Chi-cuadrado: 4139.134070744171, p-valor: 3.084600533235019e-227
```

Análisis de Tendencias Temporales

Realizemos una analisis temporar de los datos, especificamente en el dataframe. Comenzamos por visualizar la duracion de los contratos.

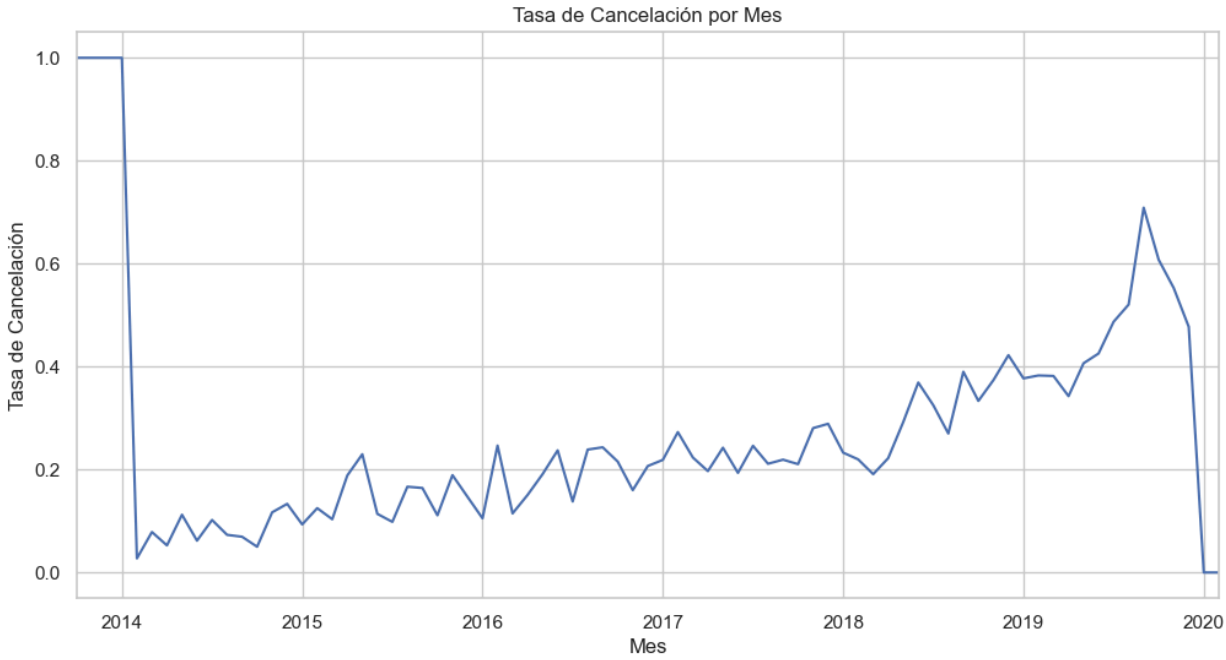
In [ ]:
```
contract_df['ContractDuration'] = contract_df['ContractDuration'] = (contract_df['EndD

plt.figure(figsize=(10, 5))
contract_df['ContractDuration'].hist(bins=30)
plt.title('Distribución de la Duración de los Contratos')
plt.xlabel('Duración (días)')
plt.ylabel('Frecuencia')
plt.show()
```



La duración de los contratos indica que aproximadamente el 75 % de los clientes nuevos no pasan los 100 días.

Calcularemos la tasa de cancelación por mes

In [ ]:
```
contract_df['Cancelled'] = contract_df['EndDate'].notna().astype(int)
contract_df['Month'] = contract_df['BeginDate'].dt.to_period('M')
cancellations_per_month = contract_df.groupby('Month')['Cancelled'].mean()

plt.figure(figsize=(12, 6))
cancellations_per_month.plot()
plt.title('Tasa de Cancelación por Mes')
plt.xlabel('Mes')
plt.ylabel('Tasa de Cancelación')
plt.show()
```



La tasa de cancelación por mes va en tendencia, aumentando cada año.

Crearemos una serie temporal del numero de contratos por mes.

In [ ]:
```
contracts_per_month = contract_df.groupby(contract_df['BeginDate'].dt.to_period('M')).

contracts_per_month.plot(figsize=(12, 6), title='Número de contratos por mes')
plt.xlabel('Fecha')
plt.ylabel('Número de contratos')
plt.show()
```

El número de contratos por mes se ha mantenido en el pasado, pero en los últimos años va en aumento.

Ingeniería de características

Trabajemos con las características y el objetivo para poder desarrollar un modelo de calidad. Haremos un escalado One Hot a las caracteristicas para que este sea uniforme entre las caracteristicas y funcione con las numericas.

In [ ]:
```
data = merged_df.copy()
data['ActiveContract'] = data['EndDate'].isna().astype(int)
```

Caracteristicas innecesarias

Nos desharemos de las columnas que no aportan información predictiva significativa, como 'gender', 'ContractDuration' y 'SeniorCitizen', que tienen una correlación irrelevante. Las columnas 'MonthlyCharges' y 'TotalCharges' tienen una alta correlación entre sí, por lo que podría haber información redundante. Sin embargo, estas son importantes para el análisis de ingresos, por lo que las mantendremos en las características.

In [ ]:
```
data.drop(columns=['gender', 'SeniorCitizen', 'customerID', 'EndDate'], inplace=True)
```

Escalado de caracteristicas

Utilizaremos un escalado estándar en las características 'MonthlyCharges' y 'TotalCharges'. La característica 'Type' quedará igual ya que la escala es prácticamente la misma; además, tiene un peso alto en la correlación con la característica objetivo. Para las caracteristicas categoricas utilizaremos One Hot.

In [ ]:
```
categorical_transformer = OneHotEncoder(drop='first')
scaler = StandardScaler()

categorical_columns = ['Partner', 'Dependents', 'InternetService', 'OnlineSecurity',
                       'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV
                       'StreamingMovies', 'MultipleLines', 'PaperlessBilling', 'Paymer
numerical_columns = ['MonthlyCharges', 'TotalCharges']

categorical_encoded = categorical_transformer.fit_transform(data[categorical_columns])
categorical_encoded_df = pd.DataFrame(categorical_encoded,
                                      columns=categorical_transformer.get_feature_name

categorical_encoded_df.index = data.index
data = pd.concat([data.drop(columns=categorical_columns), categorical_encoded_df], axi

data[numerical_columns] = scaler.fit_transform(data[numerical_columns])
```

Manejor de fechas

Convertiremos las columnas 'BeginDate' y 'EndDate' en varias caracteristicas; una para cada componente.

In [ ]:
```
data['BeginDate_year'] = data['BeginDate'].dt.year
data['BeginDate_month'] = data['BeginDate'].dt.month
data['BeginDate_day'] = data['BeginDate'].dt.day

data = data.drop(columns=['BeginDate'])
```

Analisis de clases

In [ ]:
```
data['ActiveContract'].value_counts()
```

Out[ ]:
```
1    5174
0    1869
Name: ActiveContract, dtype: int64
```

Hay una descompensación en las clases. Algunos modelos, como Random Forest o XGBClassifier, manejan correctamente este problema. En otros modelos, lo podemos abordar de distintas formas, como el sobremuestreo o la asignación de pesos a las clases, e incluso el submuestreo (aunque no es una buena idea debido a la cantidad de datos disponibles).
Primero, vamos a dividir el conjunto de datos para poder aplicar una técnica de sobremuestreo al conjunto de entrenamiento.

Haremos la división de los datos en una proporción 70/15/15 para entrenamiento, prueba y validación, respectivamente.

In [ ]:
```
X = data.drop(columns=['ActiveContract'])
y = data['ActiveContract']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.17
```

```
print("Tamaño de entrenamiento:", len(X_train))
print("Tamaño de validación:", len(X_valid))
print("Tamaño de prueba:", len(X_test))
```

```
Tamaño de entrenamiento: 4930
Tamaño de validación: 1056
Tamaño de prueba: 1057
```

Ahora realizamos una tecnica de sobremuestreo al conjunto de entrenamiento. Haremos una copia de este, asi podremos utilizar ambas tecnicas en los modelos y ver cual funciona mejor.

In [ ]:
```
smote = SMOTE(random_state=random_state, n_jobs=5, sampling_strategy='minority')
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
print("Tamaño original del conjunto de entrenamiento:", len(X_train))
print("Tamaño del conjunto de entrenamiento después de SMOTE:", len(X_train_resampled))
```

```
Tamaño original del conjunto de entrenamiento: 4930
Tamaño del conjunto de entrenamiento después de SMOTE: 7194
c:\Users\Martin\anaconda3\envs\ml\lib\site-packages\imblearn\over_sampling\_smote\base.py:370: FutureWarning: The parameter `n_jobs` has been deprecated in 0.10 and will be removed in 0.12. You can pass an nearest neighbors estimator where `n_jobs` is already set instead.
  warnings.warn(
```

In [ ]:
```
y_train_resampled.value_counts()
```

Out[ ]:
```
1    3597
0    3597
Name: ActiveContract, dtype: int64
```

In [ ]:
```
X_train_resampled
```

Out[ ]:

	MonthlyCharges	TotalCharges	Partner_Yes	Dependents_Yes	InternetService_Fiber optic	InternetServi
0	-1.489359	-0.957426	0.000000	0.0	0.0	
1	1.026602	0.652649	0.000000	0.0	1.0	
2	0.700890	-0.931970	0.000000	0.0	1.0	
3	-1.186912	-0.948603	1.000000	0.0	0.0	
4	-0.282894	-0.364190	1.000000	1.0	0.0	
...	...	...	...	...	...	
7189	1.266401	0.069288	0.538348	0.0	1.0	
7190	0.759505	-0.572449	1.000000	0.0	1.0	
7191	-0.634599	-0.971695	0.000000	0.0	0.0	
7192	-0.817587	-0.967426	0.306795	0.0	0.0	
7193	-0.533524	-0.911711	0.034002	1.0	0.0	

7194 rows × 22 columns

Ya tenemos un conjunto de entrenamiento en donde las clases estan equilibradas y otro donde no.

Evaluación

Definimos una función que nos facilitará la evaluación de los modelos.

In [ ]:
```
def evaluate_model(model, train_features, train_target, test_features, test_target,
                   cv_enabled=False, oversampling_enabled=True, cv_folds=5):
    """
    Evalúa un modelo de clasificación utilizando múltiples métricas de rendimiento, vi

    Parámetros:
    ----------
    model : object
        Modelo de clasificación entrenado que implementa los métodos predict y predict

    train_features : array-like or sparse matrix, shape (n_samples, n_features)
        Conjunto de características de entrenamiento.

    train_target : array-like, shape (n_samples,)
        Etiquetas verdaderas para el conjunto de entrenamiento.

    test_features : array-like or sparse matrix, shape (n_samples, n_features)
        Conjunto de características de prueba.

    test_target : array-like, shape (n_samples,)
        Etiquetas verdaderas para el conjunto de prueba.

    cv_enabled : bool, optional (default=False)
        Si se debe realizar validación cruzada.

    oversampling_enabled : bool, optional (default=True)
        Si se debe utilizar el conjunto con oversampling para el entrenamiento.

    cv_folds : int, optional (default=5)
        Número de pliegues (folds) a utilizar en la validación cruzada.

    Devuelve:
    --------
    None
        Imprime las métricas de evaluación y muestra gráficos de las curvas F1, ROC y

    Descripción:
    -----------
    Esta función evalúa un modelo de clasificación utilizando varias métricas de rendi
    Genera gráficos para visualizar el rendimiento del modelo en términos de:
    - Curvas F1 para diferentes umbrales de decisión
    - Curvas ROC (Receiver Operating Characteristic)
    - Curvas PRC (Precision-Recall)
    """
    eval_stats = {}

    # Definir los datos de entrenamiento según el parámetro oversampling_enabled
    if oversampling_enabled:
        train_features, train_target = X_train_resampled, y_train_resampled
```

```
    fig, axs = plt.subplots(1, 3, figsize=(20, 6))

    for type, features, target in (('train', train_features, train_target), ('test', t

        eval_stats[type] = {}

        pred_target = model.predict(features)
        pred_proba = model.predict_proba(features)[:, 1]

        # Calcular F1 para varios umbrales
        f1_thresholds = np.arange(0, 1.01, 0.05)
        f1_scores = [metrics.f1_score(target, pred_proba>=threshold) for threshold in

        # Calcular Accuracy para varios umbrales
        accuracy_scores = [metrics.accuracy_score(target, pred_proba>=threshold) for t

        # ROC y AUC
        fpr, tpr, roc_thresholds = metrics.roc_curve(target, pred_proba)
        roc_auc = metrics.roc_auc_score(target, pred_proba)
        eval_stats[type]['ROC AUC'] = roc_auc

        color = 'blue' if type == 'train' else 'green'

        # Gráfico F1
        ax = axs[0]
        max_f1_score_idx = np.argmax(f1_scores)
        ax.plot(f1_thresholds, f1_scores, color=color, label=f'{type}, max={f1_scores[
        for threshold in (0.2, 0.4, 0.5, 0.6, 0.8):
            closest_value_idx = np.argmin(np.abs(f1_thresholds-threshold))
            marker_color = 'orange' if threshold != 0.5 else 'red'
            ax.plot(f1_thresholds[closest_value_idx], f1_scores[closest_value_idx], cc
        ax.set_xlim([-0.02, 1.02])
        ax.set_ylim([-0.02, 1.02])
        ax.set_xlabel('threshold')
        ax.set_ylabel('F1')
        ax.legend(loc='lower center')
        ax.set_title(f'Valor F1')

        # Gráfico ROC
        ax = axs[1]
        ax.plot(fpr, tpr, color=color, label=f'{type}, ROC AUC={roc_auc:.2f}')
        for threshold in (0.2, 0.4, 0.5, 0.6, 0.8):
            closest_value_idx = np.argmin(np.abs(roc_thresholds-threshold))
            marker_color = 'orange' if threshold != 0.5 else 'red'
            ax.plot(fpr[closest_value_idx], tpr[closest_value_idx], color=marker_color
        ax.plot([0, 1], [0, 1], color='grey', linestyle='--')
        ax.set_xlim([-0.02, 1.02])
        ax.set_ylim([-0.02, 1.02])
        ax.set_xlabel('FPR')
        ax.set_ylabel('TPR')
        ax.legend(loc='lower center')
        ax.set_title(f'Curva ROC')

        # Gráfico exactitud
        ax = axs[2]
        max_accuracy_idx = np.argmax(accuracy_scores)
        ax.plot(f1_thresholds, accuracy_scores, color=color, label=f'{type}, max={accu
        for threshold in (0.2, 0.4, 0.5, 0.6, 0.8):
            closest_value_idx = np.argmin(np.abs(f1_thresholds-threshold))
            marker_color = 'orange' if threshold != 0.5 else 'red'
```

```
            ax.plot(f1_thresholds[closest_value_idx], accuracy_scores[closest_value_id
        ax.set_xlim([-0.02, 1.02])
        ax.set_ylim([-0.02, 1.02])
        ax.set_xlabel('threshold')
        ax.set_ylabel('Accuracy')
        ax.legend(loc='lower center')
        ax.set_title(f'Exactitud')

        eval_stats[type]['Accuracy'] = metrics.accuracy_score(target, pred_target)
        eval_stats[type]['F1'] = metrics.f1_score(target, pred_target)

    df_eval_stats = pd.DataFrame(eval_stats).round(2)

    # Si cv_enabled es True, realiza validación cruzada
    if cv_enabled and not isinstance(model, Sequential):
        skf = StratifiedKFold(n_splits=cv_folds, shuffle=True, random_state=random_sta
        cv_scores = cross_val_score(model, train_features, train_target, cv=skf, scori
        print(f"ROC AUC CV (mean): {cv_scores.mean():.2f}, (std): {cv_scores.std():.2f

    print(df_eval_stats)

    return
```

Definimos una funcion que nos ayudara con la busqueda de hiperparametros.

In [ ]:
```
def grid_search(model, param_grid, cv, x, y, scoring='roc_auc', refit=True, n_jobs=Nor
    search = GridSearchCV(estimator=model, param_grid=param_grid, cv=cv, scoring=scori
    search.fit(x, y)
    return search.best_estimator_
```

Entrenamiento y evaluacion

Modelo constante.

Comenzaremos con un modelo constante que nos servira de indicativo de rendimiento para los otros modelos. Utilizaremos la mediana

In [ ]:
```
median = y_train.median()

median_pred_df = pd.Series(np.full(y_test.size, median))
media_roc = metrics.roc_auc_score(y_test, median_pred_df)
print('ROC del modelo constante:', media_roc)
```

```
ROC del modelo constante: 0.5
```

Esto nos servira como referencia para una mejor evaluacion.

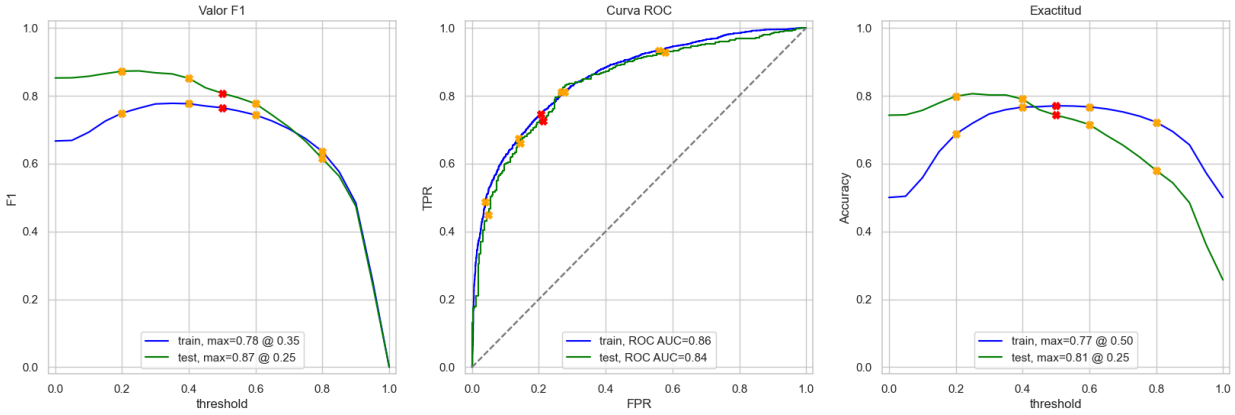
Modelo lineal

Buscamos el mejor modelo lineal.

In [ ]:
```
linear_model = LogisticRegression(max_iter=1000)
linear_model.fit(X_train_resampled, y_train_resampled);
```

In [ ]:
```
evaluate_model(linear_model, X_train_resampled, y_train_resampled, X_test, y_test, cv_
```

```
ROC AUC CV (mean): 0.85, (std): 0.01
          train  test
ROC AUC    0.86  0.84
Accuracy   0.77  0.74
F1         0.76  0.81
```

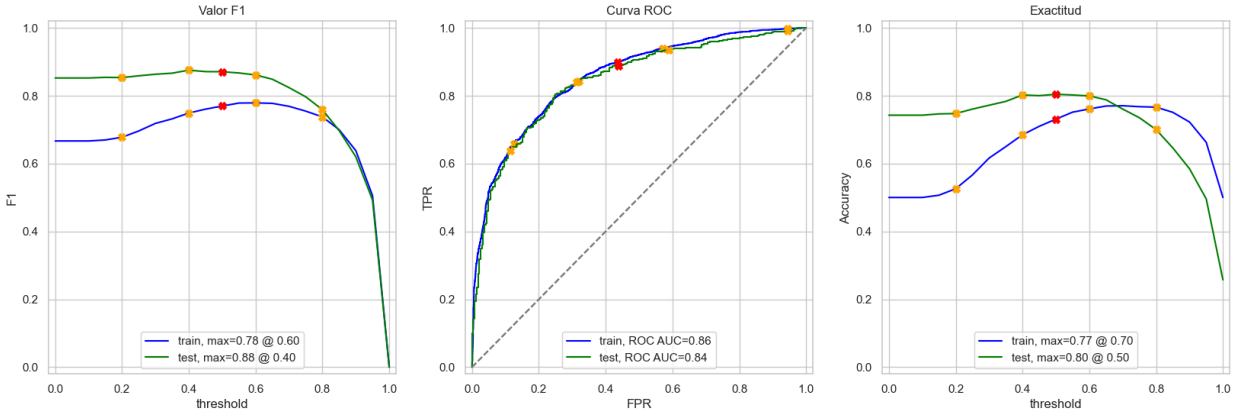


Observamos un buen desempeño del modelo lineal; la métrica ROC_AUC para el conjunto de entrenamiento y de prueba es muy cercana, lo que indica que el modelo es igualmente bueno identificando verdaderos positivos y negativos en ambos conjuntos. El umbral para la exactitud varía mucho entre los distintos grupos de datos, donde en las pruebas el umbral tuvo que reducirse para poder clasificar positivos.

Esto podría indicar sobreajuste. Para estar más seguros, corramos una prueba con los datos sin sobremuestreo utilizando el mismo modelo.

In [ ]:
```
linear_model.fit(X_train, y_train)
evaluate_model(linear_model, X_train, y_train, X_test, y_test, cv_enabled=True)
```

```
ROC AUC CV (mean): 0.85, (std): 0.02
          train  test
ROC AUC    0.86  0.84
Accuracy   0.73  0.80
F1         0.77  0.87
```



Como vemos, la exactitud mejoró para un umbral más alto (cercano a 0.5, que sería el caso ideal para la clasificación binaria), lo cual nos arroja mejores resultados para el conjunto de prueba. Es posible que el conjunto de sobremuestreo esté generando un sobreajuste; lo comprobaremos con los otros modelos.

La métrica ROC-AUC mide la tasa entre VP y FP para cada punto (área bajo la curva), por lo que puede verse menos afectada por la distribución de clases.

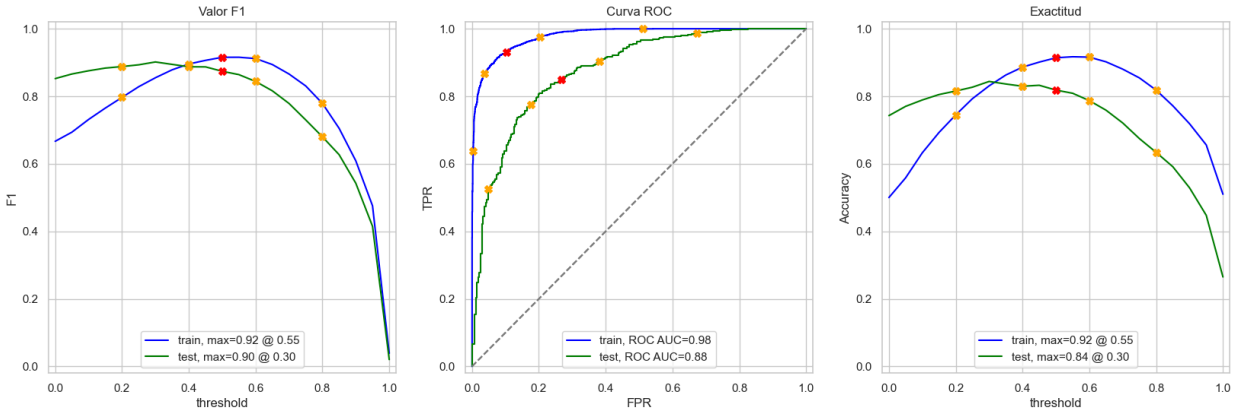
Modelo bosque aleatorio

Probaremos con ambos conjuntos de entrenamiento. Realizaremos una búsqueda de hiperparámetros para encontrar el mejor modelo.

In [ ]:
```
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}

forest_model_balanced = RandomForestClassifier(random_state=random_state, class_weight
forest_model_balanced = grid_search(forest_model_balanced, param_grid_rf, 5, X_train,
```

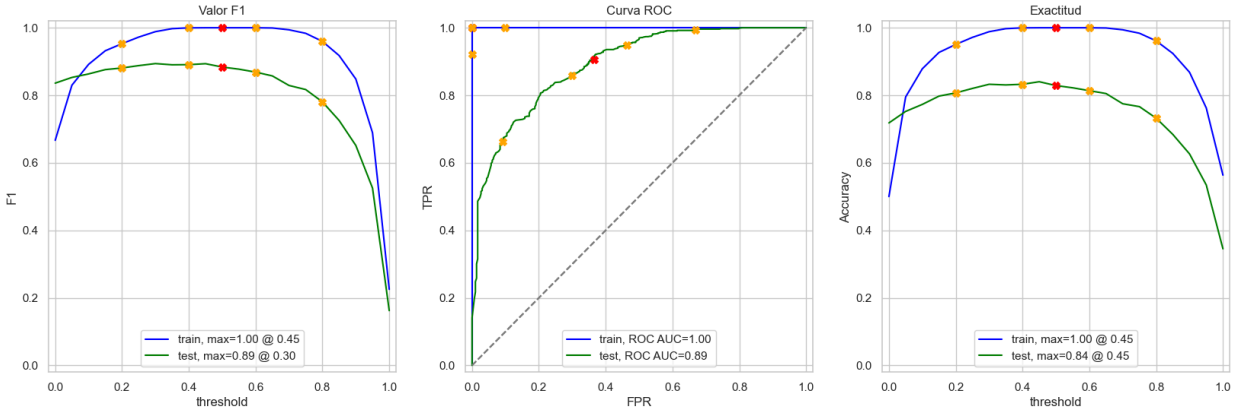
In [ ]:
```
evaluate_model(forest_model_balanced, X_train, y_train, X_test, y_test, cv_enabled=Tru
```

```
ROC AUC CV (mean): 0.95, (std): 0.00
          train  test
ROC AUC    0.98  0.88
Accuracy   0.91  0.82
F1         0.92  0.87
```



In [ ]:
```
fores_model = RandomForestClassifier(random_state=random_state)
fores_model = grid_search(fores_model, param_grid_rf, 5, X_train_resampled, y_train_re

evaluate_model(fores_model, X_train_resampled, y_train_resampled, X_test, y_test, cv_e
```

```
ROC AUC CV (mean): 0.96, (std): 0.00
          train  test
ROC AUC     1.0  0.89
Accuracy    1.0  0.83
F1          1.0  0.88
```



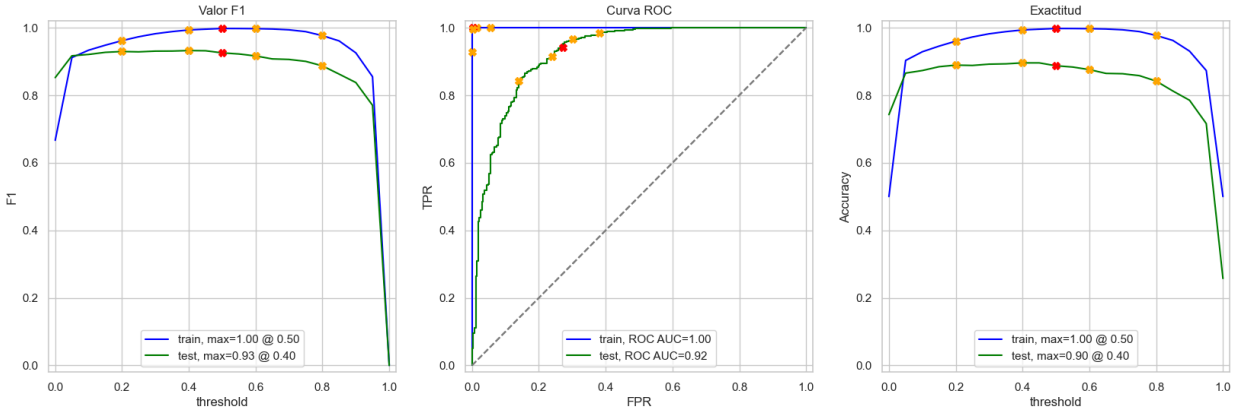
De nuevo, observamos una diferencia entre los resultados de los distintos conjuntos (con y sin sobremuestreo), principalmente en las métricas del conjunto de entrenamiento, donde todas son 1, lo que indica claramente un sobreajuste en los datos con sobremuestreo.

Claramente, el modelo tiene un mejor rendimiento con los datos originales y sin presentar un sobreajuste notable.

Modelo XGBoost

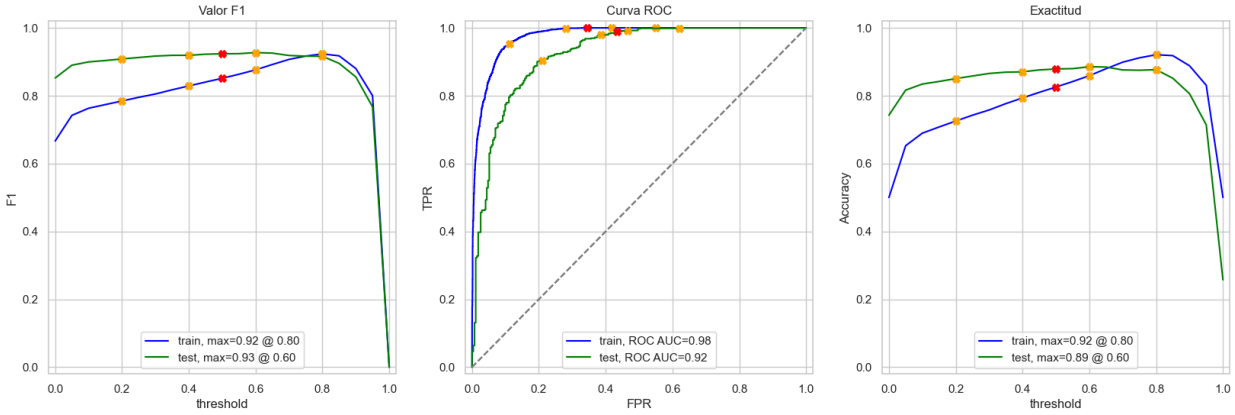
In [ ]:
```
param_grid_xgboost = {
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.3]
}

xbg_model = XGBClassifier(random_state=random_state)
xbg_model = grid_search(xbg_model, param_grid_xgboost, 5, X_train_resampled, y_train_r

evaluate_model(xbg_model, X_train_resampled, y_train_resampled, X_test, y_test, cv_ena
```

```
ROC AUC CV (mean): 0.97, (std): 0.01
           train  test
ROC AUC      1.0  0.92
Accuracy     1.0  0.89
F1           1.0  0.93
```



In [ ]:
```
xbg_model = XGBClassifier(random_state=random_state, scale_pos_weight=2.7)
xbg_model = grid_search(xbg_model, param_grid_xgboost, 5, X_train, y_train, n_jobs=2)
evaluate_model(xbg_model, X_train, y_train, X_test, y_test, cv_enabled=True)
```

```
ROC AUC CV (mean): 0.97, (std): 0.00
          train  test
ROC AUC    0.98  0.92
Accuracy   0.83  0.88
F1         0.85  0.92
```



Observamos los mismos resultados que en el modelo anterior, pero esta vez el umbral de exactitud en el conjunto de prueba es casi perfecto, lo que nos indica que el modelo logra identificar correctamente las clases, además de obtener una excelente puntuación en ROC-AUC. En general, este es el mejor rendimiento hasta el momento.

Modelo de red neuronal

Utilizaremos capas completamente conectadas en las que reduciremos logarítmicamente el número de unidades por capa. Realizaremos una búsqueda de hiperparámetros con la red y utilizaremos la precisión para medir el rendimiento del mejor modelo. Emplearemos el ROC_AUC para determinar el mejor candidato y también mostraremos la precisión.

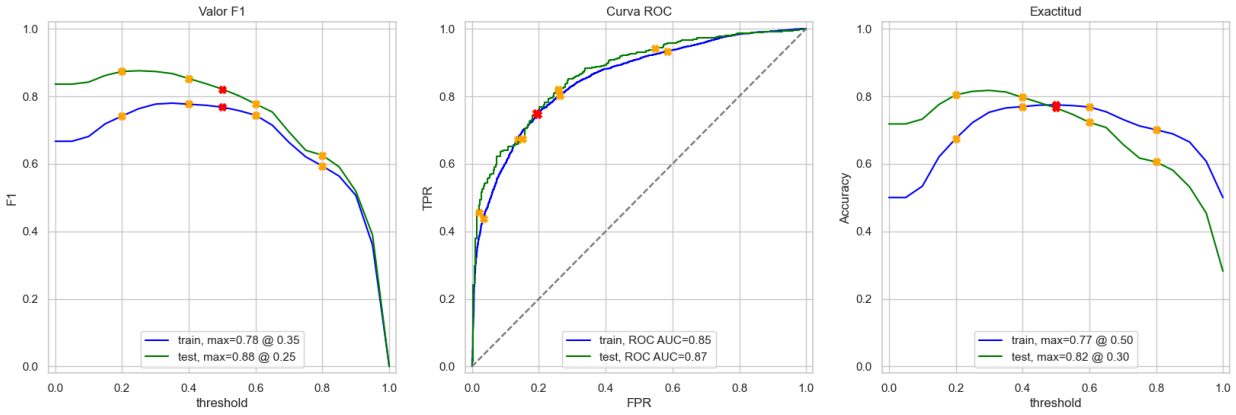
In [ ]:
```
param_grid_mlp = {
    'units': [32, 64],
    'epochs': [50, 75, 100],
    'batch_size': [32, 64]
}

scoring = {
    'roc_auc': 'roc_auc',
    'accuracy': 'accuracy'
}

# ignorar advertencias de deprecacion
warnings.filterwarnings('ignore', category=DeprecationWarning)

# convertimos los datos a arreglos de numpy para poder usarlos en keras
X_train_resampled_mlp = X_train_resampled.values
X_test_mlp = X_test.values

def create_mlp_model(units):
    model = Sequential()
    model.add(Dense(units, input_dim=X_train.shape[1], activation='relu'))
    model.add(Dense(units // 2, activation='relu'))
    model.add(Dense(units // 4, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))  # para clasificacion binaria
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'
```

```
    return model

# envoltorio para scikitlearn
wrapper = tf.keras.wrappers.scikit_learn.KerasClassifier(build_fn=create_mlp_model, ve
```

In [ ]:
```
grid_search_mlp = GridSearchCV(estimator=wrapper, param_grid=param_grid_mlp, cv=3, scc
grid_search_mlp_result = grid_search_mlp.fit(X_train_resampled_mlp, y_train_resampled)
```

In [ ]:
```
best_mlp_model = grid_search_mlp_result.best_estimator_
evaluate_model(best_mlp_model, X_train_resampled_mlp, y_train_resampled, X_test_mlp, y
```

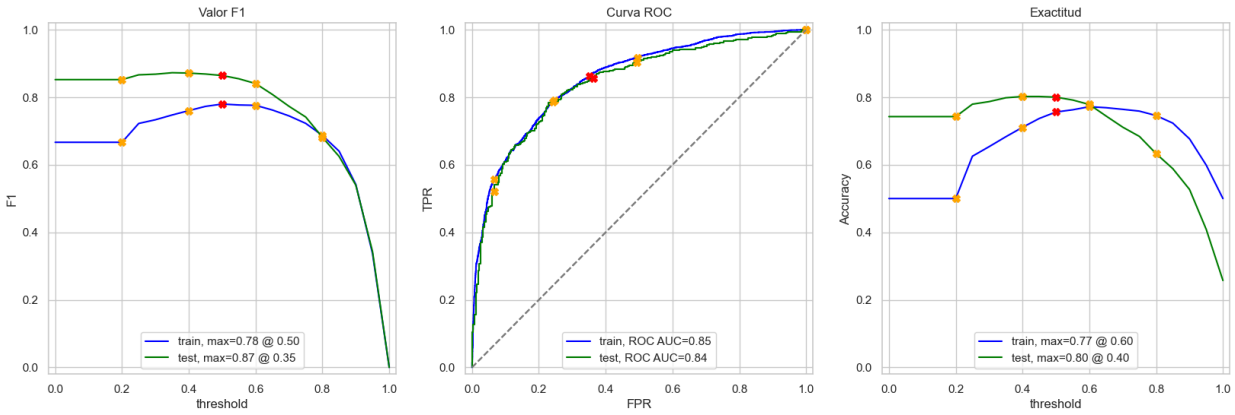
```
227/227 [==============================] - 0s 891us/step
227/227 [==============================] - 0s 697us/step
34/34 [==============================] - 0s 544us/step
34/34 [==============================] - 0s 483us/step
          train  test
ROC AUC    0.85  0.87
Accuracy   0.77  0.77
F1         0.77  0.82
```



In [ ]:
```
mlp = grid_search(wrapper, param_grid_mlp, 3, X_train, y_train, scoring=scoring, refit
evaluate_model(mlp, X_train, y_train, X_test, y_test, cv_enabled=True)
```

```
52/52 [==============================] - 0s 449us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 469us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 547us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 449us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 469us/step
52/52 [==============================] - 0s 449us/step
52/52 [==============================] - 0s 547us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 469us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 541us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 625us/step
52/52 [==============================] - 0s 449us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 586us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 547us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 430us/step
52/52 [==============================] - 0s 449us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 469us/step
52/52 [==============================] - 0s 449us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 547us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 469us/step
52/52 [==============================] - 0s 469us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 469us/step
52/52 [==============================] - 0s 449us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 567us/step
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 606us/step
52/52 [==============================] - 0s 430us/step
```

```
52/52 [==============================] - 0s 489us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 469us/step
52/52 [==============================] - 0s 508us/step
52/52 [==============================] - 0s 547us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 430us/step
52/52 [==============================] - 0s 528us/step
52/52 [==============================] - 0s 482us/step
52/52 [==============================] - 0s 469us/step
52/52 [==============================] - 0s 469us/step
52/52 [==============================] - 0s 567us/step
225/225 [==============================] - 0s 507us/step
225/225 [==============================] - 0s 512us/step
34/34 [==============================] - 0s 544us/step
34/34 [==============================] - 0s 544us/step
45/45 [==============================] - 0s 521us/step
45/45 [==============================] - 0s 476us/step
45/45 [==============================] - 0s 453us/step
45/45 [==============================] - 0s 453us/step
45/45 [==============================] - 0s 544us/step
ROC AUC CV (mean): 0.85, (std): 0.01
          train  test
ROC AUC    0.85  0.84
Accuracy   0.76  0.80
F1         0.78  0.86
```



Resultados

Modelo	Conjunto	ROC AUC	Accuracy	F1 Score
Regresión Logística	Entrenamiento	0.86	0.77	0.76
	Prueba	0.84	0.74	0.81
Regresión Logística	Entrenamiento	0.86	0.73	0.77
(sin sobremuestreo)	Prueba	0.84	0.80	0.87
Random Forest	Entrenamiento	0.98	0.91	0.92
	Prueba	0.88	0.82	0.87
Random Forest	Entrenamiento	1.00	1.00	1.00
(sin sobremuestreo)	Prueba	0.89	0.83	0.88

Modelo	Conjunto	ROC AUC	Accuracy	F1 Score
XGBoost	Entrenamiento	1.00	0.99	0.99
	Prueba	0.93	0.88	0.92
XGBoost	Entrenamiento	0.97	0.88	0.89
(sin sobremuestreo)	Prueba	0.92	0.88	0.92
MLP	Entrenamiento	0.85	0.77	0.77
	Prueba	0.87	0.77	0.82
MLP	Entrenamiento	0.50	0.50	0.67
(sin sobremuestreo)	Prueba	0.50	0.74	0.85