

✓ Telecom X – Parte 2: Predicción de Cancelación (Churn)

✓ 1. Introducción

✓ 📈 Historia del Desafío

¡Felicidades! 🎉 Has sido promovido después de tu excelente desempeño en el análisis exploratorio de la cancelación de clientes en Telecom X. Tu dedicación, claridad al comunicar los datos y visión estratégica marcaron la diferencia.

```
# Mi Challenge Telecom X - Parte 1  
# https://github.com/freedox-cts/challenge-telecom-x
```

Ahora, ¡has sido invitado oficialmente a formar parte del equipo de Machine Learning de la empresa!

✓ 🎯 Misión

Tu nueva misión es desarrollar modelos predictivos capaces de prever qué clientes tienen mayor probabilidad de cancelar sus servicios.

La empresa quiere anticiparse al problema de la cancelación, y te corresponde a ti construir un pipeline robusto para esta etapa inicial de modelado.

✓ 💬 Objetivos del Desafío

- 1 - Preparar los datos para el modelado (tratamiento, codificación, normalización).
- 2 - Realizar análisis de correlación y selección de variables.
- 3 - Entrenar dos o más modelos de clasificación.
- 4 - Evaluar el rendimiento de los modelos con métricas.
- 5 - Interpretar los resultados, incluyendo la importancia de las variables.
- 6 - Crear una conclusión estratégica señalando los principales factores que influyen en la cancelación.

✓ 💼 Lo que vas a practicar

- Preprocesamiento de datos para Machine Learning
- Construcción y evaluación de modelos predictivos
- Interpretación de resultados y entrega de insights
- Comunicación técnica con enfoque estratégico

La librería principal a usar en éste estudio será la de Scikit Learn:

https://scikit-learn.org/stable/user_guide.html

✓ 2. 🛠️ Preparación de los Datos. (EDA)

Análisis Exploratorio de Datos (EDA)

✓ Importar Librerías

```
import sys  
print(f'Versión de Python: {sys.version_info.major}.{sys.version_info.minor}.{sys.version_info.micro}')  
  
→ Versión de Python: 3.11.13  
  
import sys  
import pandas as pd # operaciones con DataFrames  
import numpy as np # operaciones matemáticas
```

```
import sklearn      # Machine Learning, Scikit-Learn
import yellowbrick # visualización de diagnóstico ML, extiende Scikit-Learn

import matplotlib           # crea gráficos
import matplotlib.pyplot as plt
import seaborn as sns       # crea gráficos
import plotly               # crea gráficos
import plotly.express as px
!pip install -U kaleido==0.2.1          # guarda gráficos

print('')
print(f'Versión de Python: {sys.version_info.major}.{sys.version_info.minor}.{sys.version_info.micro}')
print('')
print(f'Versión de Pandas: {pd.__version__}')
print(f'Versión de Numpy: {np.__version__}')
print('')
print(f'Versión de Scikit-learn (sklearn): {sklearn.__version__}')
print(f'Versión de Yellowbrick: {yellowbrick.__version__}')
print('')
print(f'Versión de Matplotlib: {matplotlib.__version__}')
print(f'Versión de seaborn: {sns.__version__}')
print(f'Versión de Plotly: {plotly.__version__}')
```

→ Requirement already satisfied: kaleido==0.2.1 in /usr/local/lib/python3.11/dist-packages (0.2.1)

Versión de Python: 3.11.13

Versión de Pandas: 2.2.2
Versión de Numpy: 2.0.2

Versión de Scikit-learn (sklearn): 1.6.1
Versión de Yellowbrick: 1.5

Versión de Matplotlib: 3.10.0
Versión de seaborn: 0.13.2
Versión de Plotly: 5.24.1

```
import requests # carga datos desde URL

import warnings
warnings.filterwarnings('ignore')
```

Guia del Usuario de Scikit Learn:

https://scikit-learn.org/stable/user_guide.html

▼ Extracción del Archivo Tratado

```
# dataset intervenido del Challenge anterior Telecom X Parte 1, formato CSV
url = 'https://github.com/freedox-cts/challenge-telecom-x/raw/refs/heads/main/TelecomX_Data_intervenido.csv'

try:
    # pedir respuesta de URL
    respuesta = requests.get(url)
    respuesta.raise_for_status() # error si la solicitud HTTP no fue exitosa

    # cargar datos CSV a DataFrame
    datos = pd.read_csv(url)

except requests.exceptions.RequestException as e:
    print('¡Error de conexión!')
    print(f'Verifica conexión o enlace: {e}')
except Exception as e:
    print('¡No se pudo cargar datos!')
    print(f'Error: {e}')

datos.shape
→ (7267, 22)

datos.head()
```

5 rows × 22 columns

	<code>id_cliente</code>	<code>cancelacion</code>	<code>genero</code>	<code>mayor_65</code>	<code>pareja</code>	<code>dependientes</code>	<code>antiguedad</code>	<code>serv_telefonico</code>	<code>multip_lineas</code>	<code>serv_internet</code>	...
0	0002-ORFBO	0	Femenino	0	1	1	9	1	0	DSL	...
1	0003-MKNFE	0	Masculino	0	0	0	9	1	1	DSL	...
2	0004-TLHLJ	1	Masculino	0	0	0	4	1	0	Fibra optica	...
3	0011-IGKFF	1	Masculino	1	1	0	13	1	0	Fibra optica	...
4	0013-EXCHZ	1	Femenino	1	1	0	3	1	0	Fibra optica	...

▼ Eliminación de Columnas Irrelevantes

Se consideran irrelevantes las columnas:

- 'id_cliente' por no influir en el estudio de ML y
- 'cuentas_diarias' por ser solo un cálculo de pago_mensual sobre una constante (/30 días).

```
datos = datos.drop(['id_cliente','cuentas_diarias'], axis=1)
```

Columnas 'id_clientes' y 'cuentas_diarias' eliminadas.

```
datos.info()
```

```
5 rows × 22 columns
RangeIndex: 7267 entries, 0 to 7266
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   cancelacion      7267 non-null   int64  
 1   genero           7267 non-null   object  
 2   mayor_65         7267 non-null   int64  
 3   pareja           7267 non-null   int64  
 4   dependientes     7267 non-null   int64  
 5   antiguedad       7267 non-null   int64  
 6   serv_telefonico  7267 non-null   int64  
 7   multip_lineas    7267 non-null   int64  
 8   serv_internet    7267 non-null   object  
 9   seguridad_online 7267 non-null   int64  
 10  respaldo_online  7267 non-null   int64  
 11  proteccion_dispositivo 7267 non-null   int64  
 12  serv_tecnico    7267 non-null   int64  
 13  tv_streaming    7267 non-null   int64  
 14  peliculas_streaming 7267 non-null   int64  
 15  contrato         7267 non-null   object  
 16  boleta_electronica 7267 non-null   int64  
 17  metodo_pago      7267 non-null   object  
 18  pago_mensual     7267 non-null   float64 
 19  pago_total        7267 non-null   float64 
dtypes: float64(2), int64(14), object(4)
memory usage: 1.1+ MB
```

▼ Variables

▼ Variables Numéricas o Cuantitativas

```
datos.describe()
```

	cancelacion	mayor_65	pareja	dependientes	antiguedad	serv_telefonico	multip_lineas	seguridad_online	respaldo_o
count	7267.000000	7267.000000	7267.000000	7267.000000	7267.000000	7267.000000	7267.000000	7267.000000	7267.000000
mean	0.257190	0.162653	0.484106	0.300124	32.346498	0.902711	0.421770	0.285950	0.34
std	0.437115	0.369074	0.499782	0.458343	24.571773	0.296371	0.493876	0.451897	0.41
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00
25%	0.000000	0.000000	0.000000	0.000000	9.000000	1.000000	0.000000	0.000000	0.00
50%	0.000000	0.000000	0.000000	0.000000	29.000000	1.000000	0.000000	0.000000	0.00
75%	1.000000	0.000000	1.000000	1.000000	55.000000	1.000000	1.000000	1.000000	1.00
max	1.000000	1.000000	1.000000	1.000000	72.000000	1.000000	1.000000	1.000000	1.00

```
# listado de variables numéricas
var_numericas = datos.select_dtypes(include=np.number).columns.tolist()
var_numericas

['cancelacion',
 'mayor_65',
 'pareja',
 'dependientes',
 'antiguedad',
 'serv_telefonico',
 'multip_lineas',
 'seguridad_online',
 'respaldo_online',
 'proteccion_dispositivo',
 'serv_tecnico',
 'tv_streaming',
 'peliculas_streaming',
 'boleta_electronica',
 'pago_mensual',
 'pago_total']

# cantidad de variables numéricas
num_vars = len(datos.select_dtypes(include=np.number).columns)

# remover la variable comparación
var_numericas.remove('cancelacion')

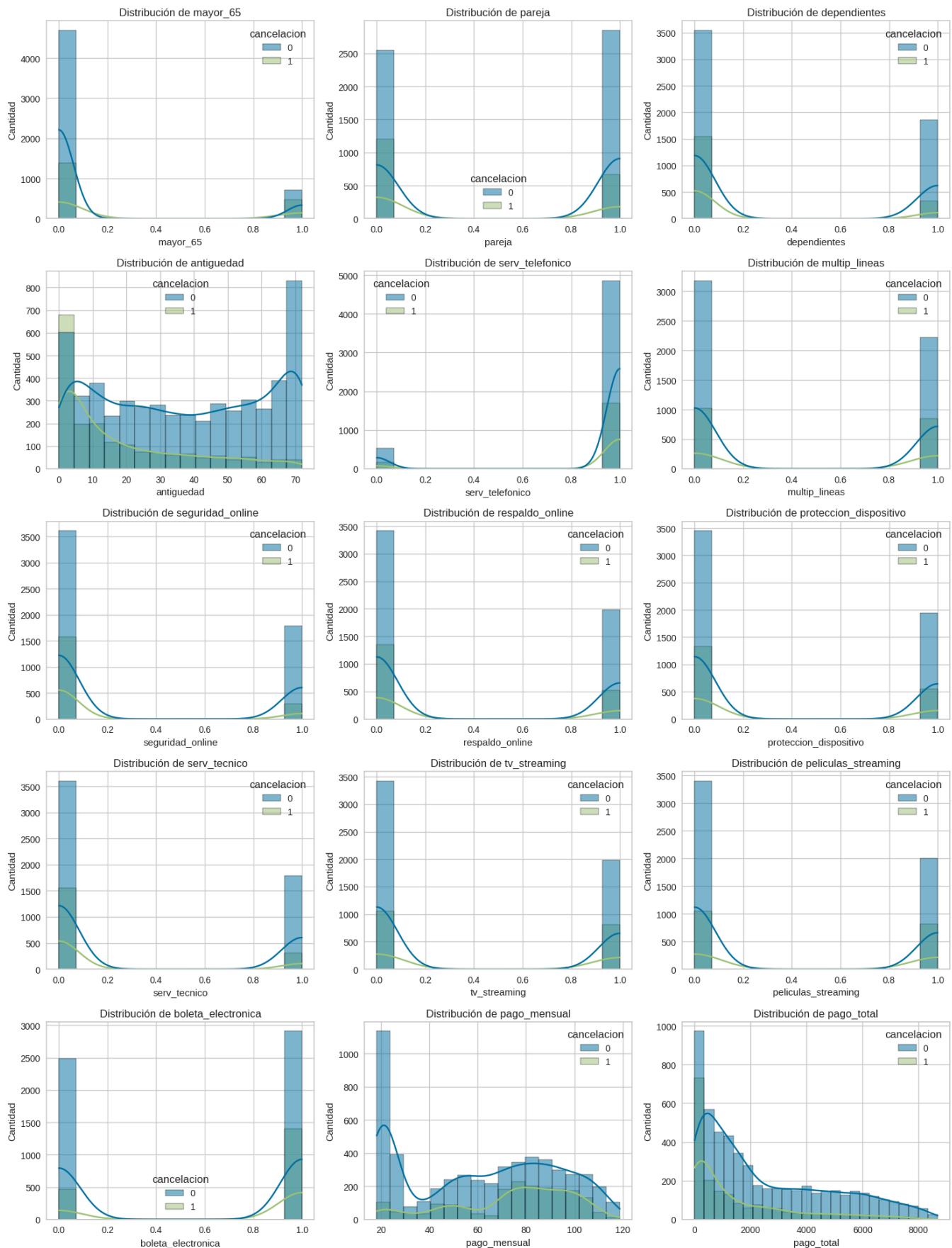
# filas y columnas para los subplots
n_cols = 3
n_rows = (num_vars-1 + n_cols - 2) // n_cols

plt.figure(figsize=(n_cols * 5, n_rows * 4))
plt.suptitle('Distribución de variables numéricas', y=1 ,fontsize=16, weight='bold')

for i, var in enumerate(var_numericas):
    plt.subplot(n_rows, n_cols, i + 1)
    sns.histplot(data=datos, x=var, hue='cancelacion', kde=True)
    plt.title(f'Distribución de {var}')
    plt.xlabel(var)
    plt.ylabel('Cantidad')

plt.tight_layout()
plt.savefig('1-cancelacionesVSnumericas.png', format='png', bbox_inches='tight')
plt.show()
```

Distribución de variables numéricas



➤ Variables Categóricas o Cualitativas

[] ↴ 4 celdas ocultas

▼ Variables explicativas y variable de respuesta

Para realizar la predicción de los valores con un modelo de aprendizaje automático, necesitamos separar la variable objetivo de las variables explicativas.

- variable **y** representa lo que queremos predecir,
- variable **X** incluye todas las variables que se utilizarán para explicar el comportamiento de **y**.

```
X_org = datos.drop(columns=['cancelacion'],axis=1)
y_org = datos['cancelacion']
```

```
X_org.sample(3,random_state=11)
```

	genero	mayor_65	pareja	dependientes	antiguedad	serv_telefonico	multip_lineas	serv_internet	seguridad_online	respald
3813	Femenino	0	1	0	33		1	1	Fibra optica	0
1687	Femenino	1	1	0	72		1	1	Fibra optica	1
4948	Masculino	0	1	0	61		1	1	DSL	0

```
y_org.sample(3,random_state=11)
```

	cancelacion
3813	1
1687	0
4948	0

dtype: int64

▼ Encoding: Codificación de variables categóricas

Transforma las variables categóricas a formato numérico para hacerlas compatibles con los algoritmos de machine learning. Utiliza un método de codificación adecuado, como one-hot encoding.

```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder, LabelEncoder

var_categoricas

['genero', 'serv_internet', 'contrato', 'metodo_pago']

one_hot = make_column_transformer((OneHotEncoder(drop='if_binary'),var_categoricas),
                                 remainder='passthrough',
                                 sparse_threshold=0,
                                 force_int_remainder_cols=False)

label_encoder = LabelEncoder()
```

```

datos_enc = one_hot.fit_transform(datos)
datos_enc

→ array([[ 0.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
       6.56000e+01, 5.93300e+02],
       [ 1.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
       5.99000e+01, 5.42400e+02],
       [ 1.00000e+00, 0.00000e+00, 1.00000e+00, ..., 1.00000e+00,
       7.39000e+01, 2.80850e+02],
       ...,
       [ 1.00000e+00, 1.00000e+00, 0.00000e+00, ..., 1.00000e+00,
       5.03000e+01, 9.27500e+01],
       [ 1.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
       6.78500e+01, 4.62765e+03],
       [ 1.00000e+00, 1.00000e+00, 0.00000e+00, ..., 0.00000e+00,
       5.90000e+01, 3.70760e+03]])

cols_datos_enc = one_hot.get_feature_names_out()
cols_datos_enc

→ array(['onehotencoder__genero_Masculino',
        'onehotencoder__serv_internet_DSL',
        'onehotencoder__serv_internet_Fibra optica',
        'onehotencoder__serv_internet_No', 'onehotencoder__contrato_Anual',
        'onehotencoder__contrato_Bi-anual',
        'onehotencoder__contrato_Mensual',
        'onehotencoder__metodo_pago_Cheque electrónico',
        'onehotencoder__metodo_pago_Cheque por correo',
        'onehotencoder__metodo_pago_Tarjeta de crédito (automático)',
        'onehotencoder__metodo_pago_Transferencia bancaria (automático)',
        'remainder_cancelacion', 'remainder_mayor_65',
        'remainder_pareja', 'remainder_dependientes',
        'remainder_antiguedad', 'remainder_serv_telefonico',
        'remainder_multip_lineas', 'remainder_seguridad_online',
        'remainder_respaldo_online', 'remainder_proteccion_dispositivo',
        'remainder_serv_tecnico', 'remainder_tv_streaming',
        'remainder_peliculas_streaming', 'remainder_boleta_electronica',
        'remainder_pago_mensual', 'remainder_pago_total'], dtype=object)

columnas_encoded = []
for columna in cols_datos_enc:
    columna = columna.split('__')[1]
    columnas_encoded.append(columna)

columnas_encoded

→ ['genero_Masculino',
    'serv_internet_DSL',
    'serv_internet_Fibra optica',
    'serv_internet_No',
    'contrato_Anual',
    'contrato_Bi-anual',
    'contrato_Mensual',
    'metodo_pago_Cheque electrónico',
    'metodo_pago_Cheque por correo',
    'metodo_pago_Tarjeta de crédito (automático)',
    'metodo_pago_Transferencia bancaria (automático)',
    'cancelacion',
    'mayor_65',
    'pareja',
    'dependientes',
    'antiguedad',
    'serv_telefonico',
    'multip_lineas',
    'seguridad_online',
    'respaldo_online',
    'proteccion_dispositivo',
    'serv_tecnico',
    'tv_streaming',
    'peliculas_streaming',
    'boleta_electronica',
    'pago_mensual',
    'pago_total']

datos_enc = pd.DataFrame(datos_enc, columns=columnas_encoded)
datos_enc

```



	genero_Masculino	serv_internet_DSL	serv_internet_Fibra optica	serv_internet_No	contrato_Annual	contrato_Bi-annual	contrato_Mensual	me
0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
1	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
2	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
3	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
4	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
...
7262	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
7263	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
7264	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
7265	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0
7266	1.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0

7267 rows × 27 columns

```
X_enc = datos_enc.drop(columns=['cancelacion'], axis=1)
y_enc = datos_enc['cancelacion']
```

```
import pickle
with open('one_hot_enc.pkl', 'wb') as file:
    pickle.dump(one_hot, file)
```

▼ Verificación de la Proporción de Cancelación (Churn)

- Calcula la proporción de clientes que cancelaron en relación con los que permanecieron activos.
- Evalúa si existe un desbalance entre las clases, ya que esto puede impactar en los modelos predictivos y en el análisis de los resultados.

```
# datos cancelación
datos_cancel = pd.DataFrame(datos['cancelacion'].map({0: 'No Canceló', 1: 'Canceló'})).rename(columns={'cancelacion': 'eleccion'})

from plotly.subplots import make_subplots

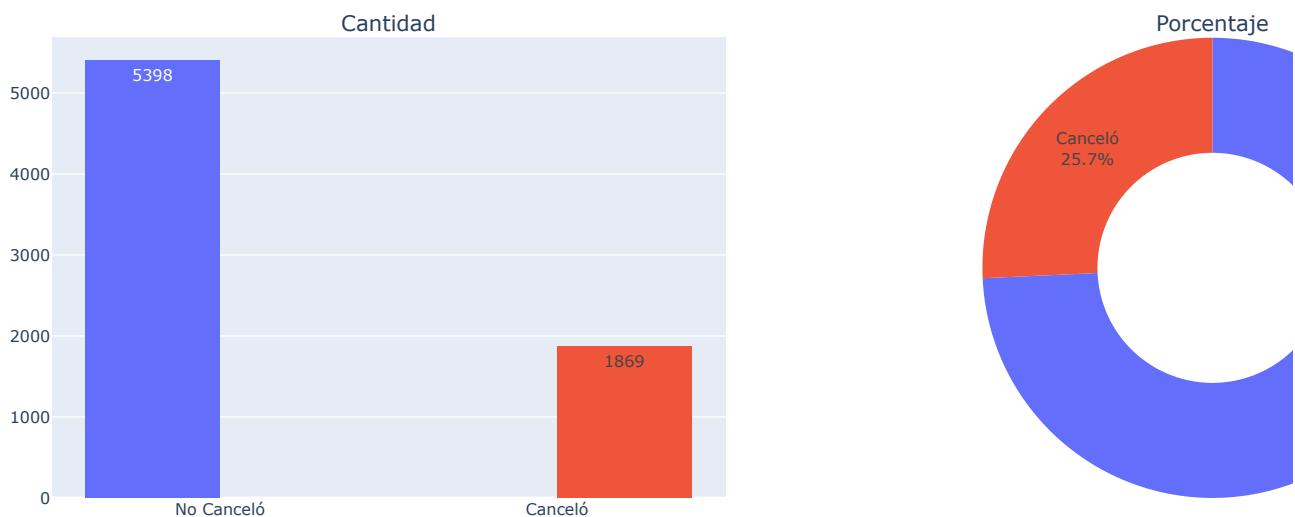
# gráficos individuales
fig_hist = px.histogram(datos_cancel, x='eleccion', color='eleccion', text_auto=True)
fig_pie = px.pie(datos_cancel, names='eleccion', hole=0.5)
# subplots
fig = make_subplots(rows=1, cols=2, subplot_titles=('Cantidad', 'Porcentaje'),
    specs=[[{'type': 'xy'}, {'type': 'domain'}]])
# agregar gráficos
fig.add_traces(fig_hist.data, rows=1, cols=1)
fig.add_traces(fig_pie.data, rows=1, cols=2)
# Ajustar diseño
fig.update_traces(textinfo='label+percent', selector={'type': 'pie'})
fig.update_layout(title="Proporción de Cancelación (Churn)", showlegend=False, autosize=True)

# guardar como imagen
fig.write_image('3-proporcion_cancelaciones.png', width=700, height=400)

fig.show()
```



Proporción de Cancelación (Churn)



- Se podría inferir que existe cierto desbalance entre clases.
- Se evaluará con resultados de modelos si es necesario balancear las clases.

✓ Balanceo de Clases (opcional)

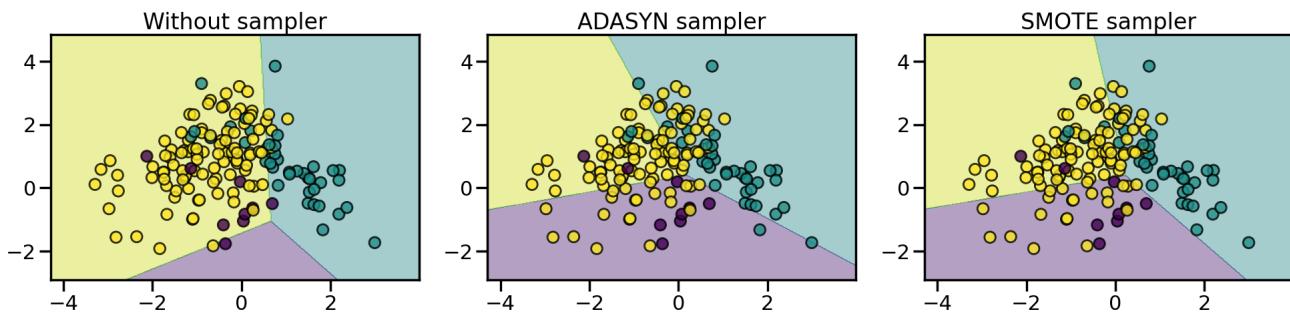
Si deseas profundizar en el análisis, aplica técnicas de balanceo como undersampling o oversampling. En situaciones de fuerte desbalanceo, herramientas como SMOTE pueden ser útiles para generar ejemplos sintéticos de la clase minoritaria.

✓ Oversampling

```
from imblearn.over_sampling import SMOTE, SMOTENC, ADASYN # Oversampling varios
```

- https://imbalanced-learn.org/stable/over_sampling.html
- https://imbalanced-learn.org/stable/over_sampling.html#smote-adasyn

Decision function using a LogisticRegression



```
var_categoricas
```

→ ['genero', 'serv_internet', 'contrato', 'metodo_pago']

SMOTENC

- variante de SMOTE
- puede tratar tipos de datos mixtos, como características continuas y categóricas.

```
oversampling_smotenc = SMOTENC(categorical_features=var_categoricas, random_state=0)
X_over_smotenc, y_over_smotenc = oversampling_smotenc.fit_resample(X_org, y_org)
```

```
y_over_smtc.value_counts() # generó muchos datos sintéticos
```

	count
cancelacion	
0	5398
1	5398

dtype: int64

```
y_over_smtc.value_counts(normalize=True) # proporción porcentual
```

	proportion
cancelacion	
0	0.5
1	0.5

dtype: float64

```
X_over_smtc.sample(3)
```

	genero	mayor_65	pareja	dependientes	antiguedad	serv_telefonico	multip_lineas	serv_internet	seguridad_online	respald
371	Masculino	1	1	0	55	1	1	Fibra optica	0	
254	Masculino	0	0	0	23	1	1	Fibra optica	0	
3909	Masculino	0	1	0	8	1	0	No	0	

▼ Undersampling

```
from imblearn.under_sampling import RandomUnderSampler, NearMiss, NeighbourhoodCleaningRule # Undersampling varios
```

https://imbalanced-learn.org/stable/under_sampling.html

neighbourhoodCleaningRule:

- es un submuestreo por limpieza de observaciones "ruidosas" u observaciones que son "demasiado fáciles de clasificar"

```
undersampling_neighbourhoodCleaningRule = NeighbourhoodCleaningRule(n_neighbors=11)
X_ncr, y_ncr = undersampling_neighbourhoodCleaningRule.fit_resample(X_enc, y_enc)
```

```
y_ncr.value_counts() # redujo por demás la clase minoritaria
```

	count
cancelacion	
1.0	1869
0.0	1578

dtype: int64

nearMiss:

- es un submuestreo controlado aleatorio que agrega algunas reglas heurísticas para seleccionar muestras
- puede manejar tipos de datos heterogéneos, es decir, numéricos, categórico, fechas, etc.
- implementa 3 tipos diferentes de heurística con el parámetro 'version'

```
undersampling_nearmiss = NearMiss(version=3)
```

```
X_under_nmss, y_under_nmss = undersampling_nearmiss.fit_resample(X_enc, y_enc)
```

```
y_under_nmss.value_counts() # redujo hasta la clase minoritaria
```

```

count
cancelacion
0.0    1869
1.0    1869
dtype: int64

```

Normalización o Estandarización

- Es fundamental distinguir entre normalización y escalado porque afectan diferentes aspectos del conjunto de datos.
- Mientras que la normalización se aplica a las muestras individuales, el escalado transforma las características.
- Esta diferencia es crucial al seleccionar técnicas de preprocesamiento para algoritmos que son sensibles a la magnitud de los datos.

Consideraciones sobre algoritmos:

- los que SI necesitan datos a la misma escala: KNN (K-Nearest Neighbours), Redes Neurales, Regresión Lineal, Regresión Logística y SVM.
- los que NO necesitan los datos en la misma escala: Árboles de Decisión, Bosque Aleatorio, AdaBoost, Naive Bayes, etc. (pero se recomienda realizar pruebas para normalización o estandarización).

Normalización:

- coloca los datos en el rango entre 0 y 1 o, -1 y 1 si hay valores negativos, sin distorsionar las diferencias en los rangos de valores. Es decir, no elimina los valores atípicos (valores extremos).

La siguiente fórmula matemática nos permite normalizar los datos:

$$X_{sc} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
from sklearn.preprocessing import Normalizer
```

- Es posible especificar diferentes tipos de normas mediante el parámetro norm.
- Por defecto, se utiliza la norma L2, pero se pueden utilizar otras, como la norma L1 o la norma máxima max.

```
normalizer = Normalizer(norm='l2')

X_normalizado = normalizer.fit_transform(X_enc)
X_normalizado = pd.DataFrame(X_normalizado, columns=X_enc.columns)
X_normalizado.head()
```

	genero_Masculino	serv_internet_DSL	serv_internet_Fibra_optica	serv_internet_No	contrato_Anual	contrato_Bi-anual	contrato_Mensual	metod
0	0.000000	0.001675	0.000000	0.0	0.001675	0.0	0.000000	
1	0.001832	0.001832	0.000000	0.0	0.000000	0.0	0.001832	
2	0.003443	0.000000	0.003443	0.0	0.000000	0.0	0.003443	
3	0.000805	0.000000	0.000805	0.0	0.000000	0.0	0.000805	
4	0.000000	0.000000	0.003568	0.0	0.000000	0.0	0.003568	

5 rows × 26 columns

Un ejemplo de cómo crear un pipeline con Normalizer:

```
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression

# Crear un pipeline que incluye el normalizador y un modelo
pipeline = make_pipeline(Normalizer(), LogisticRegression(), memory = None)

# Datos de ejemplo
X = np.array([[4, 1, 2, 2],
              [1, 3, 9, 3],
```

```

[5, 7, 5, 1])
y = np.array([0, 1, 0])

# Entrenar el modelo utilizando el pipeline
pipeline.fit(X, y)

# Realizar predicciones
predicciones = pipeline.predict(X)
print("\nPredicciones del modelo:")
print(predicciones)

```

Estandarización:

- tiene la misma idea que la normalización, es decir, poner los datos en la misma escala.
- Sin embargo, en la estandarización ponemos la media de los datos en 0 y la desviación estándar en 1.
- Este algoritmo se utiliza mejor cuando nuestra distribución es gaussiana.

La fórmula de puntuación z es una de las más comunes para la estandarización:

$$z = \frac{x - \mu}{\sigma}$$

```

from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler

scaler_standard = StandardScaler()

datos_stand_standardS = scaler_standard.fit_transform(X_enc)
datos_stand_standardS = pd.DataFrame(datos_stand_standardS, columns=X_enc.columns)
datos_stand_standardS.head()

# X_train_standard = scaler_standard.fit_transform(X_train)
# X_test_standard = scaler_standard.transform(X_test)

```

	genero_Masculino	serv_internet_DSL	serv_internet_Fibra optica	serv_internet_No	contrato_Anual	contrato_Bi-anual	contrato_Mensual	metod
0	-1.011487	1.385936	-0.886534	-0.527306	1.945268	-0.561723	-1.108050	
1	0.988643	1.385936	-0.886534	-0.527306	-0.514068	-0.561723	0.902487	
2	0.988643	-0.721534	1.127988	-0.527306	-0.514068	-0.561723	0.902487	
3	0.988643	-0.721534	1.127988	-0.527306	-0.514068	-0.561723	0.902487	
4	-1.011487	-0.721534	1.127988	-0.527306	-0.514068	-0.561723	0.902487	

5 rows × 26 columns

```

scaler_robust = RobustScaler()

datos_norm_robust = scaler_robust.fit_transform(datos_enc)
datos_norm_robust = pd.DataFrame(datos_norm_robust, columns=datos_enc.columns)
datos_norm_robust.head()

#X_train_normalizado = scaler_robust.fit_transform(X_train)
#X_test = scaler_robust.transform(X_test)

```

	genero_Masculino	serv_internet_DSL	serv_internet_Fibra optica	serv_internet_No	contrato_Anual	contrato_Bi-anual	contrato_Mensual	metod
0	-1.0	1.0	0.0	0.0	1.0	0.0	-1.0	
1	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
4	-1.0	0.0	1.0	0.0	0.0	0.0	0.0	

5 rows × 27 columns

▼ Pipeline y ColumnTransformer

```
from sklearn.pipeline import Pipeline
```

Empieza a programar o a crear código con IA.

```
from sklearn.compose import ColumnTransformer
from sklearn.compose import make_column_selector
```

Empieza a programar o a crear código con IA.

✓ 3. ⚡ Correlación y Selección de Variables

✓ Análisis de Correlación _ de variables numéricas

- Visualiza la matriz de correlación para identificar relaciones entre las variables numéricas.
- Presta especial atención a las variables que muestran una mayor correlación con la cancelación, ya que estas pueden ser fuertes candidatas para el modelo predictivo.

```
datos_enc.head(3)
```

→

	genero_Masculino	serv_internet_DSL	serv_internet_Fibra optica	serv_internet_No	contrato_Anual	contrato_Bi-anual	contrato_Mensual	metod
0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0
1	1.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0
2	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0

3 rows × 27 columns

```
# matrix de correlación
matriz_corr = datos_enc.corr(method='pearson')
matriz_corr.head(3)
```

→

	genero_Masculino	serv_internet_DSL	serv_internet_Fibra optica	serv_internet_No	contrato_Anual	contrato_Bi-anual	contrat
genero_Masculino	1.000000	0.004520	-0.009572	0.006318	0.007328	-0.005449	
serv_internet_DSL	0.004520	1.000000	-0.639664	-0.380469	0.047030	0.030054	
serv_internet_Fibra optica	-0.009572	-0.639664	1.000000	-0.467474	-0.073950	-0.212975	

3 rows × 27 columns

Mapa de Calor o Heatmap para visualizar Matriz de Correlación:

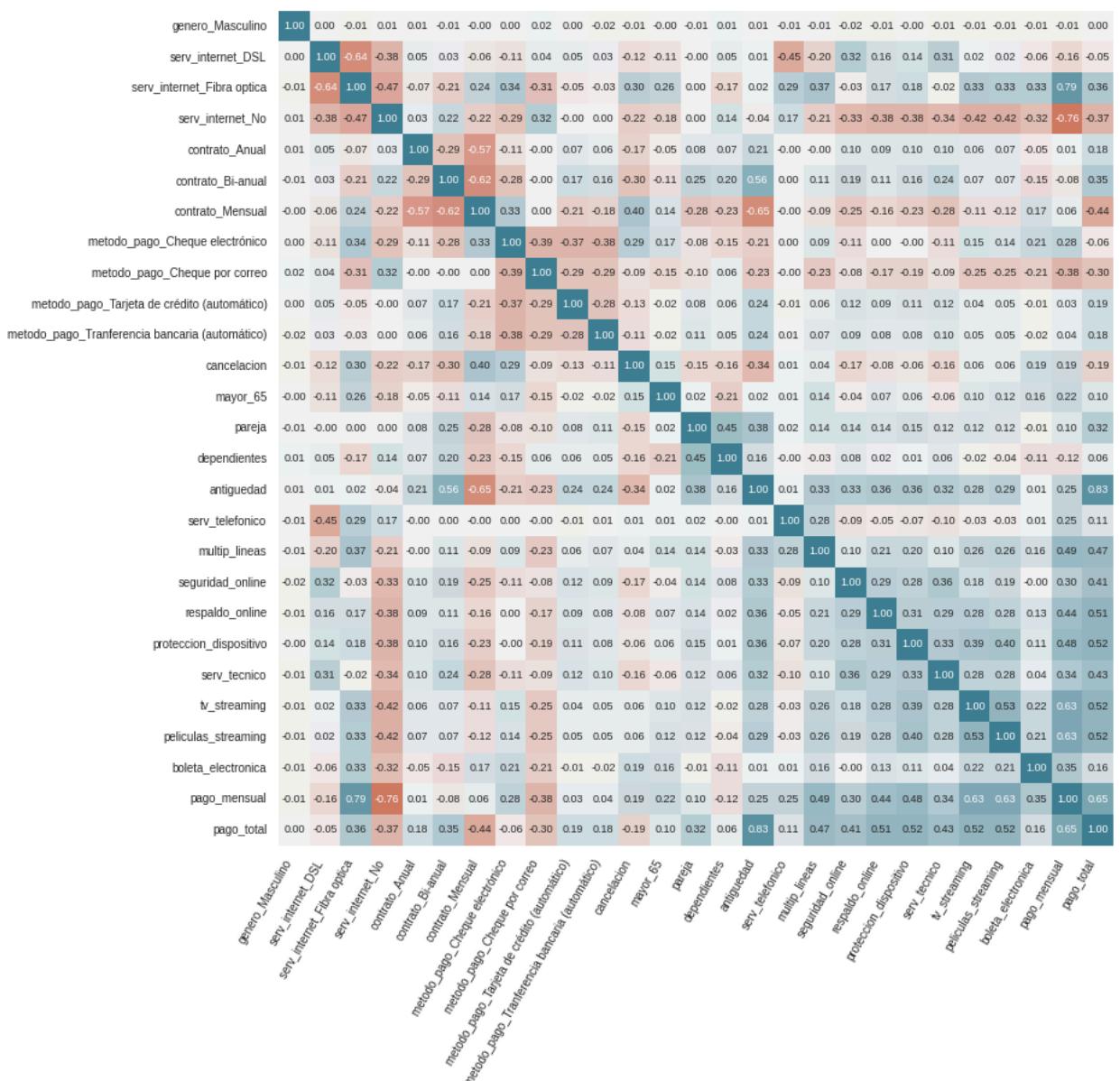
```
# mapa de calor
fig, ax = plt.subplots(figsize=(11, 9))

sns.heatmap(matriz_corr, annot=True, fmt=".2f", cbar=False, annot_kws={"size": 6.5},
             vmin=-1, vmax=1, center=0, cmap=sns.diverging_palette(20, 220, n=200),
             square=True, ax=ax)
ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation = 60,
    horizontalalignment = 'right',
)
ax.tick_params(labelsize = 7)
plt.title('Matriz de correlación', fontsize=16, pad=20)

# guardar el gráfico como imagen PNG
plt.savefig('4-MC_correlaciones.png', format='png', bbox_inches='tight')
```



Matriz de correlación



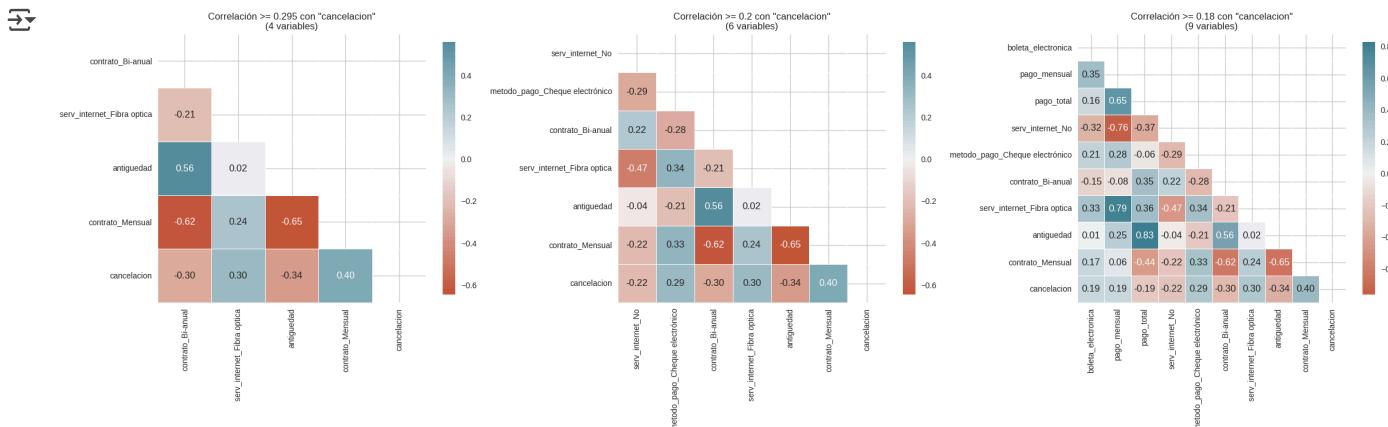
- muestra todas las correlaciones simultaneas de las 27 variables, $27 \times 27 = 729$ datos.
- no nos aporta relevancia al no tener un orden para clasificar variables mas relevantes

```
# variable objetivo
target_var = 'cancelacion'
# umbrales a graficar
umbrales = [0.295, 0.20, 0.18]
# figura con subplots
fig, axes = plt.subplots(1, 3, figsize=(24, 8))

# Iterar sobre los umbrales y crear un heatmap para cada uno
for i, limitar in enumerate(umbrales):
    # Filtrar variables relevantes y ordenar
    var_relevantes = matriz_corr.index[abs(matriz_corr[target_var]) >= limitar].tolist()
    var_relevantes_ord = matriz_corr.loc[var_relevantes, target_var].abs().sort_values(ascending=True).index.to_list()
        # Asegurar que la variable objetivo esté en la lista
    if target_var not in var_relevantes_ord:
        var_relevantes_ord.append(target_var)
    # Crear matriz de correlación filtrada
    corr_filtrada = matriz_corr.loc[var_relevantes_ord, var_relevantes_ord]
    # Generar máscara
    mascara = np.triu(np.ones_like(corr_filtrada, dtype=bool))
    # Crear heatmap en el subplot correspondiente
    sns.heatmap(corr_filtrada, annot=True, fmt=".2f", cmap=sns.diverging_palette(20, 220, n=200), center=0, square=True,
                linewidths=0.5, cbar_kws={"shrink": 0.7}, mask=mascara, ax=axes[i])
    axes[i].set_title(f'Correlación >= {limitar} con "{target_var}"\n{n(len(var_relevantes_ord)-1)} variables')

plt.tight_layout()
# Guardar y mostrar
```

```
plt.savefig('5-MC_correlaciones_varias.png', bbox_inches='tight')
plt.show()
```



Aquí se aprecian mucho mejor las variables más complicadas:

- reducimos variables aplicando umbrales de correlación
- ordenamos por valor absoluto para mejor comprensión

```
print('Variables mas relevantes ordenadas por valor absoluto de correlación con la variable "cancelación".')
print()
var_relevantes_ord = matriz_corr.loc[var_relevantes, target_var].abs().sort_values(ascending=False).index.to_list()
var_relevantes_ord
```

Variables mas relevantes ordenadas por valor absoluto de correlación con la variable "cancelación".

```
['cancelacion',
'contrato_Mensual',
'antiguedad',
'serv_internet_Fibra optica',
'contrato_Bi-anual',
'metodo_pago_Cheque electrónico',
'serv_internet_No',
'pago_total',
'pago_mensual',
'boleta_electronica']
```

Gráfico de Barras o Barplot con las Correlaciones Ordenadas

```
# DataFrame para el gráfico
df_graf = (pd.DataFrame({'Correlación': (matriz_corr.loc['cancelacion'].drop('cancelacion'))
                           .apply(lambda x: 'Directa' if x > 0 else 'Inversa'),
                           'Magnitud': np.round((matriz_corr.loc['cancelacion'].drop('cancelacion').abs()),4)})
            .reset_index()
            .rename(columns={'index': 'Columnas'})
            .sort_values(by='Magnitud', ascending=False))
df_graf.head(2)
```

	Columnas	Correlación	Magnitud
6	contrato_Mensual	Directa	0.3956
14	antiguedad	Inversa	0.3441

Pasos siguientes: [Generar código con df_graf](#) [Ver gráficos recomendados](#) [New interactive sheet](#)

```
# gráfico
plt.figure(figsize=(8, 7))
plot = sns.barplot(data=df_graf, x='Magnitud', y='Columnas', hue='Correlación',
                    palette={'Directa':'#338cff', 'Inversa':'#ffac33'}, dodge=False)

# añadir etiquetas
[plt.text(p.get_width()+0.005, p.get_y()+p.get_height()/2, f'{p.get_width():.3f}', ha='left', va='center', fontsize=9) for p in plot.patches]

# configuración del gráfico
plt.title('Correlación de cada variable con la cancelación', fontsize=15, pad=15)
plt.xlabel('Magnitud de correlación', labelpad=10)
```

```

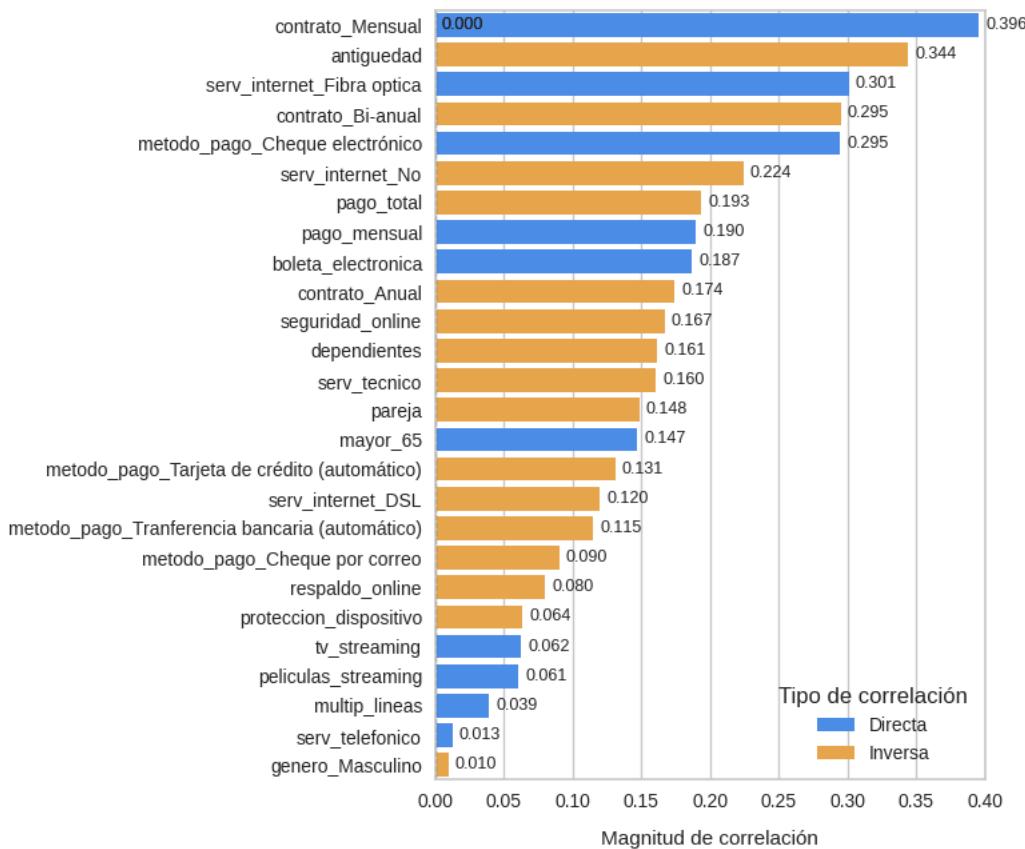
plt.ylabel('')
plt.xlim(0, 0.4)
plt.axvline(0, color='black', linestyle='--', linewidth=0.8)
plt.legend(title='Tipo de correlación')
plt.tight_layout()

# Guardar y mostrar
plt.savefig('6-GB_correlaciones_ordenadas.png', bbox_inches='tight')
plt.show()

```



Correlación de cada variable con la cancelación



Aquí se aprecian todas las variables:

- ordenadas por correlación
- de mayor a menor valor absoluto
- tipo de relación directa o inversa

▼ 4. 🔎 Análisis de Multicolinealidad con VIF

<https://www.datacamp.com/es/tutorial/variance-inflation-factor>

- La multicolinealidad hace que aumenten los errores estándar, lo que dificulta la evaluación de la importancia de los predictores individuales.
- El factor de inflación de la varianza (VIF) sirve como métrica diagnóstica precisa para identificar la multicolinealidad.

▼ Factor de Inflación de la Varianza (VIF):

- El VIF ayuda a detectar la presencia de multicolinealidad entre variables independientes.
- Generalmente, **VIF > 5** indica que la variable está colineal con otras y puede ser eliminada.
- **VIF > 10** multicolinealidad grave, eliminar el predictor o utilizar técnicas de regularización.

La fórmula del factor de inflación de la varianza El VIF de un predictor X se calcula como:

Fórmula del factor de inflación de la varianza (VIF)

$$VIF_{(X)} = 1 - R^2$$

Dónde:

R² es el coeficiente de determinación obtenido cuando X se hace una regresión sobre todos los demás predictores.

¿Cuándo es válido calcular el VIF?

Puedo (y debo) calcular el VIF si:

- **Voy a usar modelos lineales** (ej.: regresión logística, regresión lineal)
- **Quiero interpretar los coeficientes** con claridad (la multicolinealidad distorsiona signos y magnitudes)
- **Quieres garantizar estabilidad en el modelo**

¿Cuándo puedo descartar el VIF (o postergarlo)?

- Si vas a usar **modelos no lineales**, como:

- Árboles de decisión
- Random Forest
- XGBoost
- Redes neuronales

Estos modelos **no son sensibles a la multicolinealidad**.

Preguntas Clave:

Pregunta	Respuesta
¿El nuevo heatmap muestra multicolinealidad grave?	No
¿Necesito calcular el VIF obligatoriamente?	No, pero es recomendable si usas modelos lineales
¿Vale la pena como verificación extra?	Sí, especialmente si el modelo es interpretativo

- Como tengo programado utilizar regresión logística, **calcularé el VIF**.

▼ Calculo práctico de VIF

1. Importar las bibliotecas necesarias

```
from statsmodels.stats.outliers_influence import variance_inflation_factor  
from statsmodels.tools.tools import add_constant
```

2. Seleccionar las variables independientes

```
# quitamos la variable objetivo o dependiente  
X = datos_enc.drop(columns=['cancelacion'])
```

3. Añadir constante (intercepto)

```
X_const = add_constant(X)
```

4. Calcular el VIF

```
# convertir columnas booleanas a enteros (0 o 1)  
X_const = X_const.astype(float)  
# calcular el VIF  
vif_data = pd.DataFrame()  
vif_data["características"] = X_const.columns  
vif_data["VIF"] = [variance_inflation_factor(X_const.values, i) for i in range(X_const.shape[1])]
```

Función para calcular VIF

```
def calcular_vif(df,obj):  
    # importar bibliotecas  
    from statsmodels.stats.outliers_influence import variance_inflation_factor  
    from statsmodels.tools.tools import add_constant  
    # quitamos la variable objetivo o dependiente  
    X = df.drop(columns=obj)  
    # añadir constantes  
    X_const = add_constant(X)  
    # convertir columnas booleanas a enteros (0 o 1)  
    X_const = X_const.astype(float)  
    # calcular el VIF
```

```

vif_dt = pd.DataFrame()
vif_dt["características"] = X_const.columns
vif_dt["VIF"] = [variance_inflation_factor(X_const.values, i) for i in range(X_const.shape[1])]
vif_dt = vif_dt.sort_values(by='VIF', ascending=False)

return vif_dt

```

Función para Visualizar Resultados VIF

```

def mostrar_vif(vif_dat):
    # importar bibliotecas
    from math import inf
    from IPython.display import display, HTML

    bins = [0,5,10,inf]
    labels = ['VIF < 5', '5 ≤ VIF < 10', 'VIF ≥ 10']

    vif_dat['rango'] = pd.cut(vif_dat["VIF"], bins=bins, labels=labels, right=False)
    vif_dat = vif_dat.sort_values(by='VIF', ascending=False)
    #display(vif_dat.sort_values(by='VIF', ascending=False))

    vif_0_5 = vif_dat[vif_dat['rango'] == 'VIF < 5']
    vif_5_10 = vif_dat[vif_dat['rango'] == '5 ≤ VIF < 10']
    vif_10_x = vif_dat[vif_dat['rango'] == 'VIF ≥ 10']
    vif_inf = vif_dat[vif_dat['VIF'] == inf]

    # Crear HTML para el grid
    html = """
<h2>Niveles de Multicolinealidad</h2>
<div style="display: grid; grid-template-columns: 1fr 1fr; grid-gap: 20px;">
    <div style="grid-column: 1; grid-row: 1;"><h4>Tabla 1 - Perfecta (inaceptable) </h4>{}</div>
    <div style="grid-column: 2; grid-row: 1;"><h4>Tabla 2 - Alta (¡atención!)</h4>{}</div>
    <div style="grid-column: 1; grid-row: 2;"><h4>Tabla 3 - Moderada (vigilar)</h4>{}</div>
    <div style="grid-column: 2; grid-row: 2;"><h4>Tabla 4 - Baja (aceptable)</h4>{}</div>
</div>
    """.format(vif_inf.to_html(), vif_10_x.to_html(), vif_5_10.to_html(), vif_0_5.to_html())

    display(HTML(html))

df = datos_enc
col_obj = ['cancelacion']

mostrar_vif(calcular_vif(df,col_obj))

```



Niveles de Multicolinearidad

Tabla 1 - Perfecta (inaceptable)

		caracteristicas	VIF	rango
2		serv_internet_DSL	inf	NaN
4		serv_internet_No	inf	NaN
3		serv_internet_Fibra optica	inf	NaN
7		contrato_Mensual	inf	NaN
8		metodo_pago_Cheque electrónico	inf	NaN
5		contrato_Anual	inf	NaN
6		contrato_Bi-anual	inf	NaN
9		metodo_pago_Cheque por correo	inf	NaN
10		metodo_pago_Tarjeta de crédito (automático)	inf	NaN
11		metodo_pago_Transferencia bancaria (automático)	inf	NaN

Tabla 2 - Alta (¡atención!)

		caracteristicas	VIF	rango
25		pago_mensual	871.464258	VIF ≥ 10
16		serv_telefonico	35.174704	VIF ≥ 10
23		peliculas_streaming	24.203807	VIF ≥ 10
22		tv_streaming	24.123237	VIF ≥ 10
26		pago_total	10.815154	VIF ≥ 10

Tabla 3 - Moderada (vigilar)

		caracteristicas	VIF	rango
15		antiguedad	7.524785	$5 \leq VIF < 10$
17		multip_lineas	7.302265	$5 \leq VIF < 10$
20		proteccion_dispositivo	6.950955	$5 \leq VIF < 10$
19		respaldo_online	6.812465	$5 \leq VIF < 10$
21		serv_tecnico	6.470562	$5 \leq VIF < 10$
18		seguridad_online	6.347212	$5 \leq VIF < 10$

Tabla 4 - Baja (aceptable)

		caracteristicas	VIF	rango
13		pareja	1.456377	VIF < 5
14		dependientes	1.383017	VIF < 5
24		boleta_electronica	1.211179	VIF < 5
12		mayor_65	1.155178	VIF < 5
1		genero_Masculino	1.002453	VIF < 5
0		const	0.000000	VIF < 5

Principales alertas en los resultados

1. ! Tabla 1

VIF = inf = ∞ (infinito)

Estas variables **siguen siendo perfectamente colineales entre sí** o con otra variable.

🔍 Causa probable:

- Ambas provienen de la **misma variable categórica original** (PhoneService), y el encoding generó **redundancia**.
- Si el cliente **no tiene teléfono**, no puede tener múltiples líneas \Rightarrow valores 100% ligados.
- otros

⭐ Solución recomendada:

- **Elimina una de estas columnas.** Por ejemplo:

```
df.drop(columns=["phone.PhoneService_Yes"], inplace=True)
```

- O rehacer el encoding con `drop_first=True` para simplificar la estructura.

```
from math import inf
# columnas con VIF infinitos
cols_vif_inf = (calcular_vif(datos_enc,col_obj)[calcular_vif(datos_enc,col_obj)['VIF'] == inf])['caracteristicas'].tolist()

# variables rectificadas
var_rectif = datos_enc.columns
var_rectif = var_rectif.drop(cols_vif_inf)
var_rectif

→ Index(['genero_Masculino', 'cancelacion', 'mayor_65', 'pareja', 'dependientes',
       'antiguedad', 'serv_telefonico', 'multip_lineas', 'seguridad_online',
       'respaldo_online', 'proteccion_dispositivo', 'serv_tecnico',
       'tv_streaming', 'peliculas_streaming', 'boleta_electronica',
       'pago_mensual', 'pago_total'],
      dtype='object')
```

2. ! Tabla 2

pago_mensual → VIF = 813.86

- Esto es extremadamente alto.
- Esta variable está **altamente correlacionada con pago_total.

⭐ Solución:

- Verifico si pago_mensual y pago_total contiene **información repetida** (ej.: Total = pago_mensual * antiguedad).

```
# verifico
datos_a_verificar= pd.DataFrame()
datos_a_verificar['Pago Total'] = datos_enc['pago_total']
datos_a_verificar['PagoMensual X Antiguedad'] = pd.DataFrame(datos_enc['pago_mensual'] * datos_enc['antiguedad'])
datos_a_verificar['Diferencia'] = pd.DataFrame(datos_a_verificar['Pago Total'] - datos_a_verificar['PagoMensual X Antiguedad'])
datos_a_verificar['Porcentual'] = pd.DataFrame(((datos_a_verificar['Diferencia'] / datos_a_verificar['PagoMensual X Antiguedad'])*100).round(2))
datos_a_verificar.sample(5)
```

	Pago Total	PagoMensual X Antiguedad	Diferencia	Porcentual	
1012	19.00	19.00	0.00	0.00	Info
6158	1032.05	1014.30	17.75	1.75	Info
2823	1714.95	1635.20	79.75	4.88	Info
5602	5585.40	5687.10	-101.70	-1.79	Info
4507	426.35	393.75	32.60	8.28	Info

- Si es así, **elimino una o dos de estas columnas** para evitar redundancia.

```
# son casi iguales por eso producen alta colinealidad
# elimino variable 'pago_total'
var_rectif = var_rectif.drop('pago_total')
var_rectif

Index(['genero_Masculino', 'cancelacion', 'mayor_65', 'pareja', 'dependientes',
       'antiguedad', 'serv_telefonico', 'multip_lineas', 'seguridad_online',
       'respaldo_online', 'proteccion_dispositivo', 'serv_tecnico',
       'tv_streaming', 'peliculas_streaming', 'boleta_electronica',
       'pago_mensual'],
      dtype='object')
```

3. ⚠️ Otros VIFs altos (moderados a severos)

Variable	VIF	Posibles Soluciones
serv_telefonico	35.1	Dependencia con servicio DSL que ya fue eliminada
tv_streaming / peliculas_streaming	24.1-24.2	Dependen directamente de tener internet
pago_total	10.8	Ya eliminada, relacionado con pago_mensual y antiguedad

⭐ Soluciones combinadas:

- Mantener **solo una** de las variables entre serv_internet_Fibra_optica, serv_internet_No o usar drop_first=True.
- Evaluar si es necesario mantener **todas** las variables derivadas de internet (streaming, seguridad, etc.).

▼ VIF Rectificado

```
datos_enc_rectif= pd.DataFrame()
datos_enc_rectif = datos_enc[var_rectif]
#df
col_obj = ['cancelacion']

mostrar_vif(calcular_vif(datos_enc_rectif,col_obj))
```



Niveles de Multicolinearidad

Tabla 1 - Perfecta (inaceptable)

caracteristicas	VIF	rango

Tabla 2 - Alta (¡atención!)

caracteristicas	VIF	rango
0	const	16.862442 VIF ≥ 10

Tabla 3 - Moderada (vigilar)

caracteristicas	VIF	rango

Tabla 4 - Baja (aceptable)

caracteristicas	VIF	rango
15 pago_mensual	4.369251	VIF < 5
13 peliculas_streaming	1.900793	VIF < 5
12 tv_streaming	1.894994	VIF < 5
5 antiguedad	1.735491	VIF < 5
7 multip_lineas	1.533468	VIF < 5
10 proteccion_dispositivo	1.532360	VIF < 5
3 pareja	1.453415	VIF < 5
9 respaldo_online	1.444564	VIF < 5
6 serv_telefonico	1.395015	VIF < 5
4 dependientes	1.372693	VIF < 5
11 serv_tecnico	1.367058	VIF < 5
8 seguridad_online	1.343516	VIF < 5
14 boleta_electronica	1.194082	VIF < 5
2 mayor_65	1.145921	VIF < 5
1 genero_Masculino	1.001234	VIF < 5

Análisis del nuevo VIF

Rango de VIF	Interpretación
VIF ≈ 1	Sin multicolinealidad (óptimo)
1 < VIF ≤ 5	Baja (aceptable)
5 < VIF ≤ 10	Moderada (vigilar)
VIF > 10	Alta (atención o posible exclusión)

Conclusión práctica

- La multicolinealidad crítica fue resuelta (sin inf, sin redundancia perfecta)
- Solo pago_mensual tiene una multicolinealidad moderada que es aceptable

5. Modelos Predictivos

Importación de las bibliotecas

```
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score, cross_validate
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, roc_curve, auc, classification_report, accuracy_score, roc_auc_score, recall_score, make_
from sklearn.tree import DecisionTreeClassifier
```

Separar features y target - características y objetivo

```
x = datos_enc_rectif.drop(columns=['cancelacion'])
y = datos_enc_rectif['cancelacion']
```

Dividir entrenamiento, prueba y validación

```

df = cv(85%) + test(15%)
• cv = train + val
df = train(76.5%) + val(8.5%) + test(15%)

X_cv, X_test, y_cv, y_test = train_test_split(X, y, test_size=0.15, stratify=y, random_state=42)

X_train, X_val, y_train, y_val = train_test_split(X_cv, y_cv, test_size=0.10, stratify=y_cv, random_state=42)

```

▼ Normalizar entrenamiento

- Vamos a entrenar Regresión Logística, que se beneficia de la normalización. Por eso, normalizamos los datos para este modelo.
- En cambio, Random Forest no necesita normalización – pero como estamos usando los mismos datos para ambos modelos, los normalizamos para mantener consistencia y simplicidad.

```

# con StandardScaler
scaler_standard = StandardScaler()
X_train_scaled_std = scaler_standard.fit_transform(X_train)
X_test_scaled_std = scaler_standard.transform(X_test)

# con RobustScaler
scaler_robust = RobustScaler()
X_train_scaled_robust = scaler_robust.fit_transform(X_train)
X_test_scaled_robust = scaler_robust.transform(X_test)

```

▼ Balancear entrenamiento

Porque ya verificamos anteriormente que la proporción de churn estaba desbalanceada.

```

#con SMOTE (oversampling)
bal_smote = SMOTE(random_state=42)
X_train_bal_smt, y_train_bal_smt = bal_smote.fit_resample(X_train, y_train)

#con SMOTENC (oversampling)
bal_smotenc = SMOTENC(categorical_features=var_categoricas, random_state=42)
X_train_bal_smotenc, y_train_bal_smotenc = bal_smotenc.fit_resample(X_org, y_org) # no funciona sin categóricas

#con NearMiss (undersampling)
bal_nearmiss = NearMiss(version=3)
X_train_bal_nrmss, y_train_bal_nrmss = bal_nearmiss.fit_resample(X_train, y_train)

```

▼ Ajustes varios:

```

X_cv.shape
→ (6176, 15)

resultados_cv = pd.DataFrame()

# Reporte Visual
def reporte_gral(model_name, y_test, y_pred, y_prob, nombre_img=None, dpi=100):
    plt.figure(figsize=(12, 10))
    plt.suptitle(f"Evaluación del Modelo {model_name}", fontsize=16, y=1.02)
    # 1. Reporte de Clasificación
    plt.subplot(2, 2, 1)
    report = classification_report(y_test, y_pred, output_dict=True)
    sns.heatmap(pd.DataFrame(report).iloc[:-1, :].T, annot=True, cmap='YlOrRd', cbar=False)
    plt.title('Informe de clasificación')
    # 2. Métricas Principales
    plt.subplot(2, 2, 2)
    metrics = {'Accuracy': accuracy_score(y_test, y_pred), 'ROC AUC': roc_auc_score(y_test, y_prob)}
    plt.bar(metrics.keys(), metrics.values(), color=['skyblue', 'lightgreen'])
    plt.ylim(0, 1)
    for i, v in enumerate(metrics.values()):
        plt.text(i, v + 0.02, f"{v:.2f}", ha='center')
    plt.title('Métricas Clave')
    # 3. Curva ROC

```

```

plt.subplot(2, 2, 3)
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, color='darkorange', label=f'AUC = {roc_auc:.2f}')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0]); plt.ylim([0.0, 1.05])
plt.xlabel('Tasa falsos positivos'); plt.ylabel('Tasa verdaderos positivos')
plt.title('Curva ROC'); plt.legend(loc="lower right")
# 4. Matriz de Confusión
plt.subplot(2, 2, 4)
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Matriz de Confusión')
plt.xlabel('Predicción'); plt.ylabel('Actual')

plt.tight_layout()
# Guardar imagen si se especificó save_path
if nombre_img:
    plt.savefig(nombre_img, dpi=dpi, bbox_inches='tight')
plt.show()

# para ver las métricas posibles de cross_validate
# sklearn.metrics.get_scorer_names()

```

▼ Modelo Decision Tree Classifier

```

# Instanciar
dtc = DecisionTreeClassifier(max_depth=10, random_state=42)

# Entrenar
dtc.fit(X_cv, y_cv)

# Predicciones
y_pred_dtc = dtc.predict(X_test)
y_prob_dtc = dtc.predict_proba(X_test)[:, 1] # Probabilidades de la clase positiva (Churn=1)

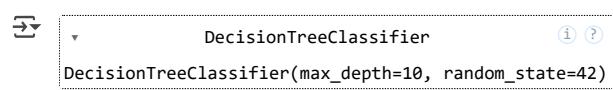
# Validar
cv_skf = StratifiedKFold(n_splits=6, shuffle=True, random_state=42)

resultados_cv_dtc = cross_validate(dtc, X_cv, y_cv, cv=cv_skf,
                                    scoring=['accuracy', 'precision', 'recall', 'f1', 'roc_auc'])

resultados_cv = pd.concat([resultados_cv, (pd.DataFrame([{'Modelo': 'Decision Tree Classifier',
                                                          'Precisión_media': resultados_cv_dtc['test_accuracy'].mean(),
                                                          'Precisión_std': resultados_cv_dtc['test_accuracy'].std(),
                                                          'ROC_AUC_media': resultados_cv_dtc['test_roc_auc'].mean(),
                                                          'ROC_AUC_std': resultados_cv_dtc['test_roc_auc'].std(),
                                                          'Tiempo_ajuste': resultados_cv_dtc['fit_time'].mean()}])), ignore_index=True])
resultados_cv

# Evaluación
reporte_gral('Decision Tree Classifier', y_test, y_pred_dtc, y_prob_dtc, '7-evaluacion_DTC')

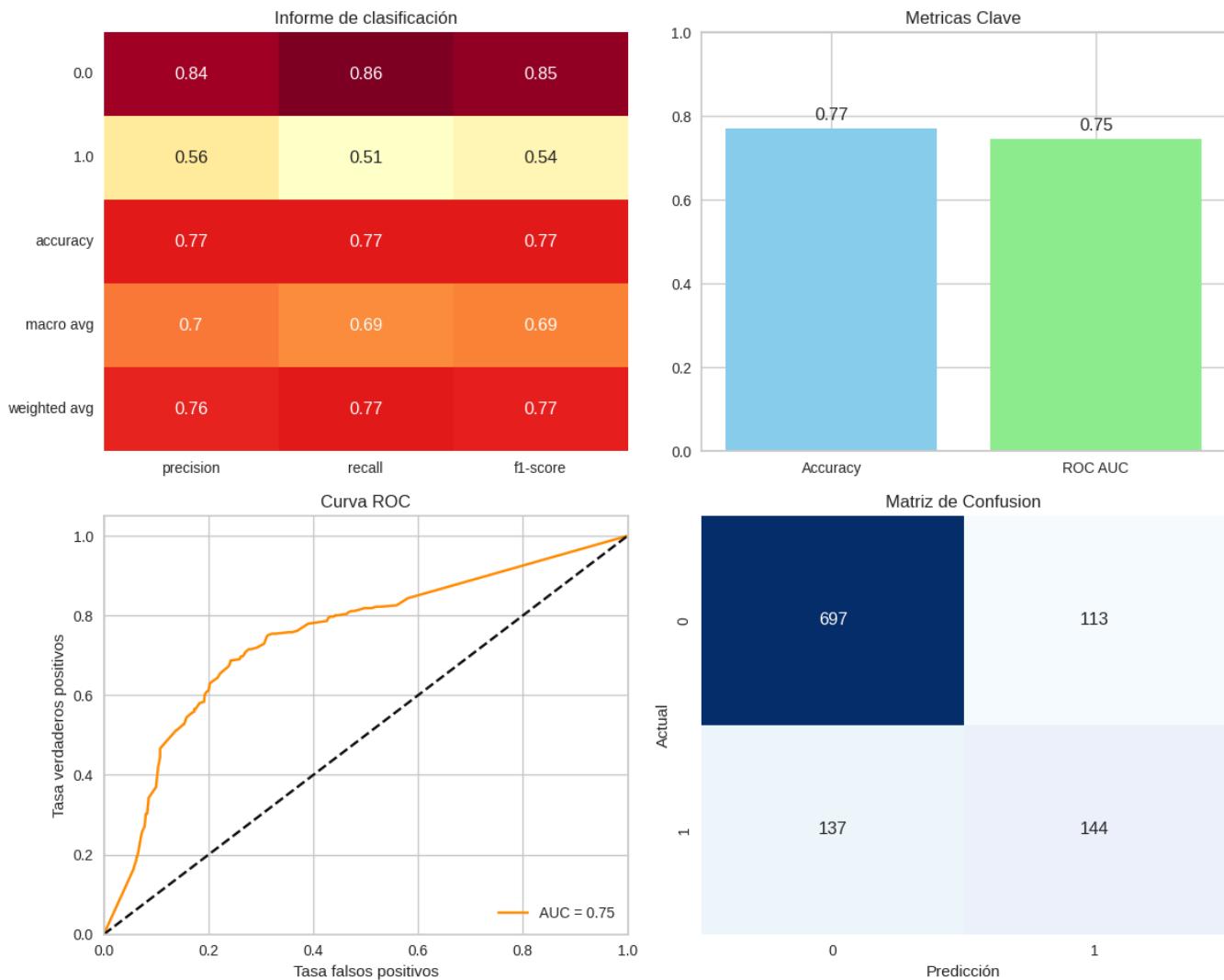
```



	Modelo	Precisión_media	Precisión_std	ROC_AUC_media	ROC_AUC_std	Tiempo_ajuste
0	Decision Tree Classifier	0.756315	0.003001	0.733766	0.009919	0.017224



Evaluación del Modelo Decision Tree Classifier



Modelo Regresión Logística

```
# Instanciar
#     'liblinear' es un buen solver para datasets pequeños y medianos, y soporta penalización L1/L2
logistic_regression = LogisticRegression(random_state=42, solver='liblinear', max_iter=1000)

# Validar
resultados_cv_lr = cross_validate(logistic_regression, X_cv, y_cv, cv=cv_skf, scoring=['accuracy', 'roc_auc'])

resultados_cv = pd.concat([resultados_cv, pd.DataFrame({'Modelo': 'Regresión Logística',
                                                       'Precisión_media': resultados_cv_lr['test_accuracy'].mean(),
                                                       'Precisión_std': resultados_cv_lr['test_accuracy'].std(),
                                                       'ROC_AUC_media': resultados_cv_lr['test_roc_auc'].mean(),
                                                       'ROC_AUC_std': resultados_cv_lr['test_roc_auc'].std(),
                                                       'Tiempo_ajuste': resultados_cv_lr['fit_time'].mean()}))], ignore_index=True)
resultados_cv
```

	Modelo	Precisión_media	Precisión_std	ROC_AUC_media	ROC_AUC_std	Tiempo_ajuste	Actions
0	Decision Tree Classifier	0.756315	0.003001	0.733766	0.009919	0.017224	
1	Regresión Logística	0.800677	0.008679	0.829352	0.008810	0.016890	

Pasos siguientes: [Generar código con resultados_cv](#) [Ver gráficos recomendados](#) [New interactive sheet](#)

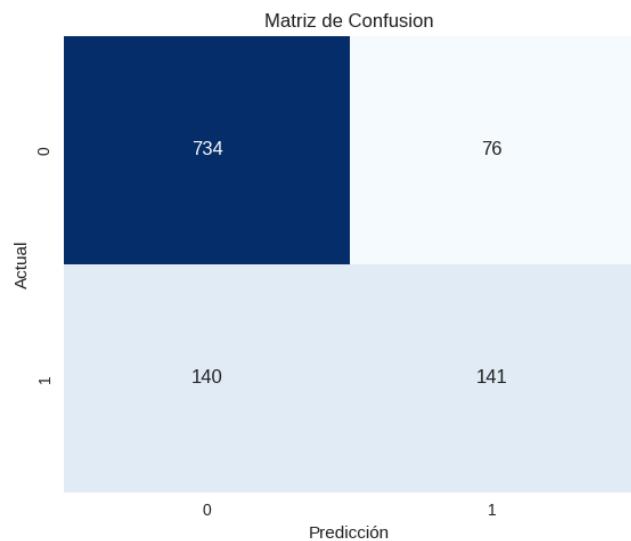
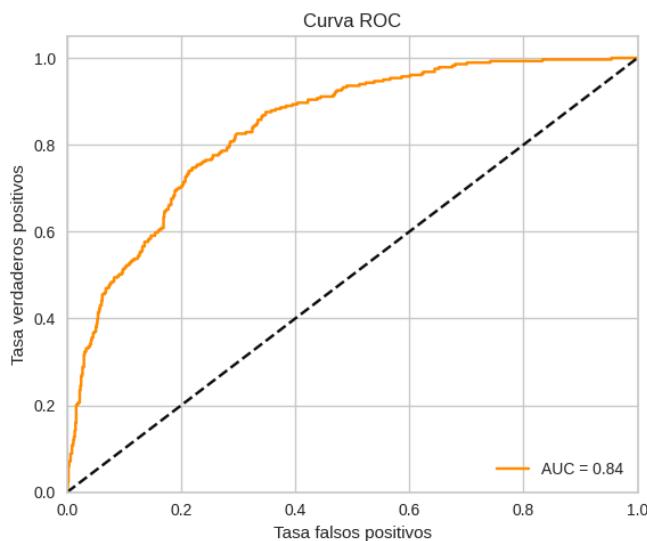
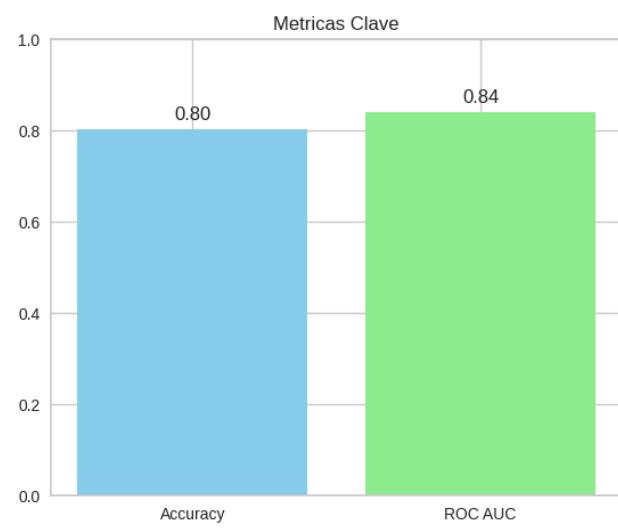
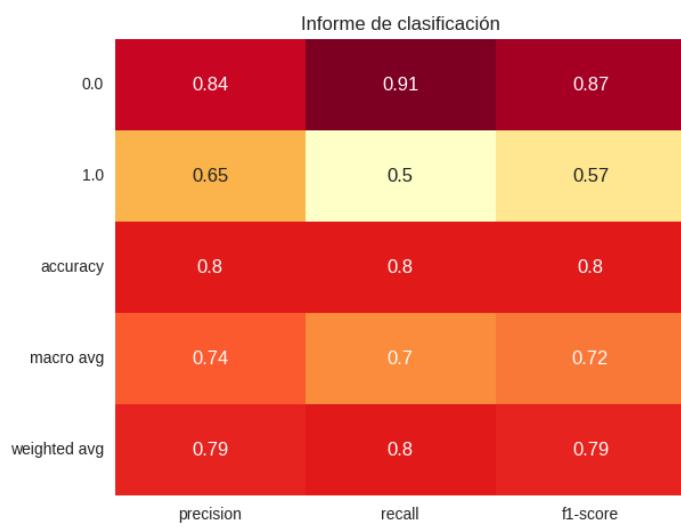
```
# Entrenar
logistic_regression.fit(X_train, y_train)
```

LogisticRegression
LogisticRegression(max_iter=1000, random_state=42, solver='liblinear')

```
# Predicciones
y_pred_lr = logistic_regression.predict(X_test)
y_prob_lr = logistic_regression.predict_proba(X_test)[:, 1]
```

```
# Evaluación
reporte_gral('Regresión Logística', y_test, y_pred_lr, y_prob_lr, '8-evaluacion_LR')
```

Evaluación del Modelo Regresión Logística



Modelo Random Forest

```
# Instanciar
random_forest = RandomForestClassifier(random_state=42, n_estimators=100)
```

```
# Entrenar
random_forest.fit(X_train, y_train)

RandomForestClassifier(random_state=42)

# Predicciones
y_pred_rf = random_forest.predict(X_test)
y_prob_rf = random_forest.predict_proba(X_test)[:, 1]

# Validar
resultados_cv_rf = cross_validate(random_forest, X_cv, y_cv, cv=cv_skf, scoring=['accuracy', 'roc_auc'])

resultados_cv = pd.concat([resultados_cv,(pd.DataFrame([{'Modelo': 'Random Forest',
    'Precisión_media': resultados_cv_rf['test_accuracy'].mean(),
    'Precisión_std': resultados_cv_rf['test_accuracy'].std(),
    'ROC_AUC_media': resultados_cv_rf['test_roc_auc'].mean(),
    'ROC_AUC_std': resultados_cv_rf['test_roc_auc'].std(),
    'Tiempo_ajuste': resultados_cv_rf['fit_time'].mean()}])), ignore_index=True)
resultados_cv
```

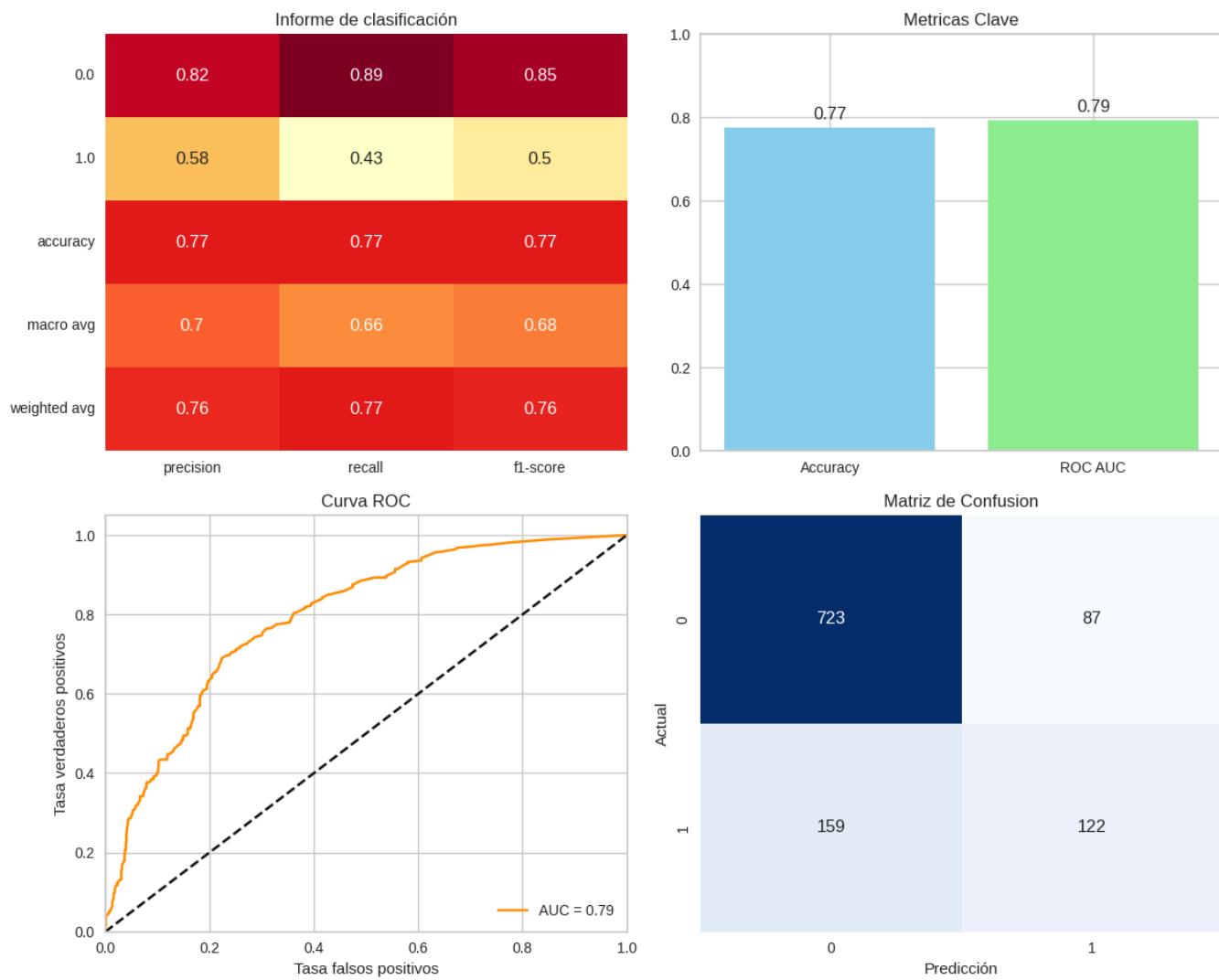
	Modelo	Precisión_media	Precisión_std	ROC_AUC_media	ROC_AUC_std	Tiempo_ajuste
0	Decision Tree Classifier	0.756315	0.003001	0.733766	0.009919	0.017224
1	Regresión Logística	0.800677	0.008679	0.829352	0.008810	0.016890
2	Random Forest	0.776878	0.006574	0.792296	0.015991	0.561018

Pasos siguientes: [Generar código con resultados_cv](#) [Ver gráficos recomendados](#) [New interactive sheet](#)

```
# Evaluación
reporte_gral('Random Forest',y_test,y_pred_rf,y_prob_rf,'9-evaluacion_RF')
```



Evaluación del Modelo Random Forest



Descripción rápida de Recall, Precisión y F1

- **Recall (Sensibilidad o TPR - True Positive Rate):**
 - Mide la capacidad del modelo para detectar **todos los casos positivos relevantes**.
 - Fórmula: $Recall = \frac{TP}{TP+FN}$
 - **Alto recall** = Menos falsos negativos (FN).
- **Precisión:**
 - Mide la **calidad de las predicciones positivas** del modelo (cuán seguros estamos cuando el modelo dice "positivo").
 - Fórmula: $Precision = \frac{TP}{TP+FP}$
 - **Alta precisión** = Menos falsos positivos (FP).
- **F1-Score:**
 - Es la **media armónica** entre precisión y recall, útil cuando hay desbalance de clases.
 - Fórmula: $F1 = 1 * \frac{Precision*Recall}{Precision+Recall}$
 - **Equilibra precisión y recall**, penalizando modelos con uno muy bajo.

Interrelaciones y Trade-offs

- ♦ **Recall vs. Precisión:** Generalmente hay un **trade-off**:
 - Aumentar el recall (capturar más positivos) suele aumentar los FP (baja precisión).
 - Aumentar la precisión (evitar FP) puede llevar a perder casos positivos (bajo recall).
- ♦ **F1-Score** ayuda a evaluar el equilibrio, pero depende del problema:
 - Si **FN son costosos** (ej.: diagnóstico médico), priorizar **recall**.
 - Si **FP son costosos** (ej.: spam en correos), priorizar **precisión**.

¿Cuándo priorizar Recall sobre Precisión?

Recall vs. Precisión

Métrica	Foco	Fórmula	Importancia
Recall	Minimizar falsos negativos (FN)	$\frac{TP}{TP+FN}$	Criticó cuando omitir un positivo es costoso (ej. salud, seguridad).
Precisión	Minimizar falsos positivos (FP)	$\frac{TP}{TP+FP}$	Importante cuando los falsos positivos son problemáticos (ej. spam en correos).

Ejemplo:

- **Modelo A:** Recall = 90%, Precisión = 50% → Detecta casi todos los positivos, pero muchos falsos positivos.
- **Modelo B:** Recall = 50%, Precisión = 90% → Detecta solo la mitad de los positivos, pero casi no tiene falsos positivos.
- ♦ **¿Cuál elegir?** Depende del problema:
 - Si es mejor **no dejar escapar ningún caso positivo** (ej. cáncer), prioriza **Recall**.
 - Si es mejor **evitar falsas alarmas** (ej. spam), prioriza **Precisión**.

Cómo mejorar el Recall (a costa de precisión)

1. **Ajustar el umbral de clasificación:**
 - Reducir el threshold de probabilidad para predecir la clase positiva (ej.: de 0.5 a 0.3).
2. **Usar modelos sensibles a FN:**
 - SVM con kernel adecuado, Random Forest con class_weight ajustado.
3. **Métricas de evaluación alternativas:**
 - Usar **ROC-AUC** (si el trade-off TPR vs. FPR es importante) o **Precisión-Recall Curve** (para clases desbalanceadas).
4. **Balancear datos:**
 - Oversampling de la clase minoritaria (SMOTE) o pérdidas ponderadas (ej.: class_weight='balanced' en sklearn).

Resumen

- **Recall alto** = Minimizar FN (importante en problemas de salud/seguridad).
- **Precisión alta** = Minimizar FP (importante en spam, recomendaciones).
- **F1** = Balance entre ambos (ideal cuando no hay prioridad clara).

Si tu problema requiere **no dejar escapar casos positivos**, optimiza para recall, incluso si eso genera más falsas alarmas.

✗ Informe Final Telecom X - Parte 2

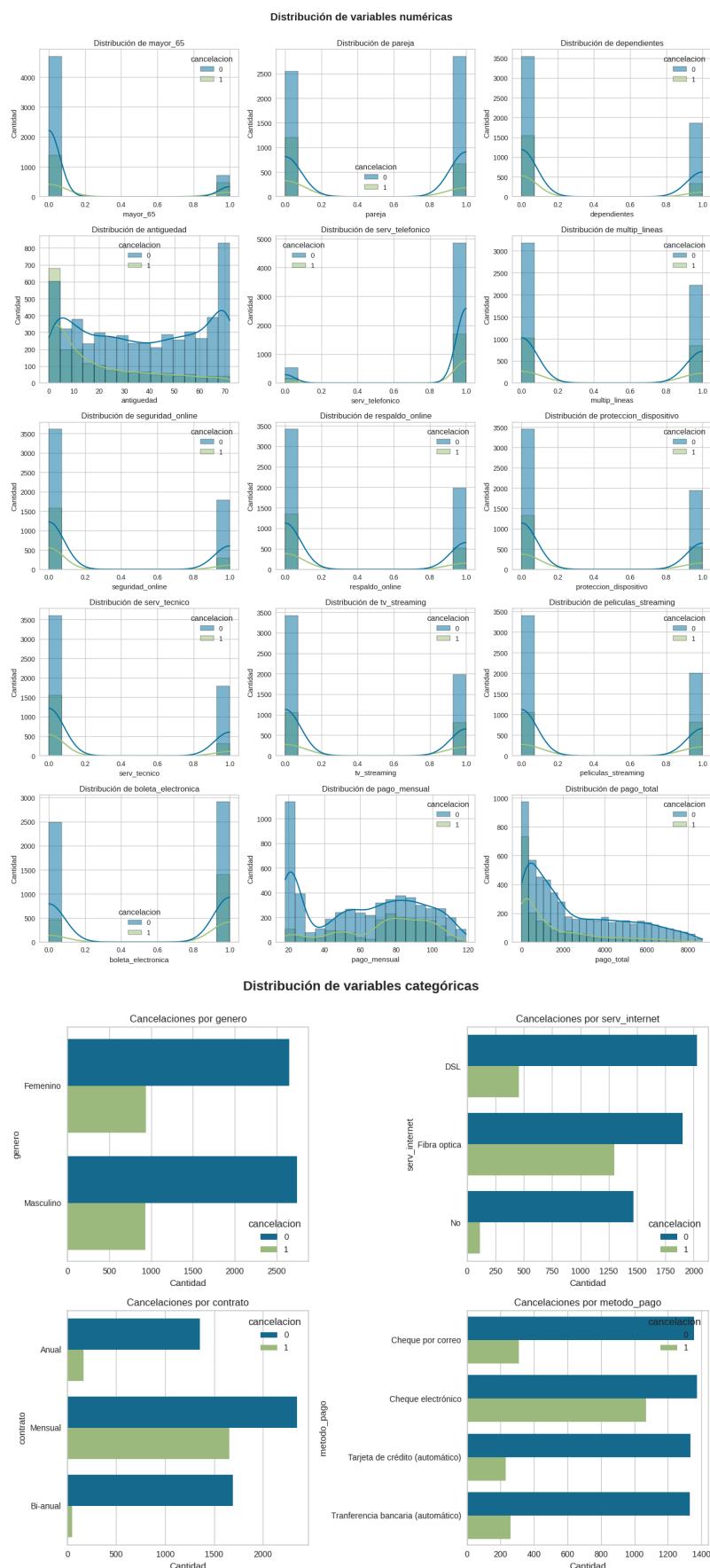
✗ ♦ Resumen

El proyecto tiene como objetivo desarrollar modelos predictivos para identificar clientes con alta probabilidad de cancelar sus servicios en Telecom X. Utilizando técnicas de machine learning, se analizaron datos históricos de clientes para predecir el churn y proporcionar insights estratégicos. El proceso incluyó preparación de datos, análisis exploratorio, modelado y evaluación de desempeño.

❖ ◆ Metodología

1. Preparación de Datos

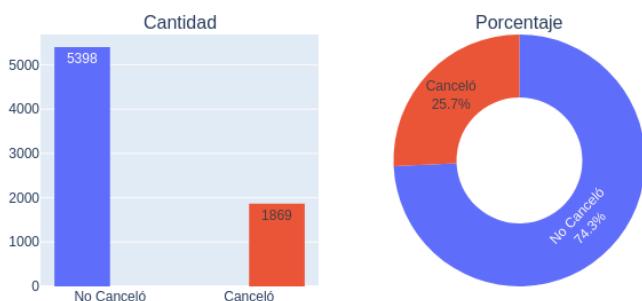
- Limpieza y Codificación:** Se eliminaron columnas irrelevantes (`id_cliente`, `cuentas_diarias`) y se aplicó One-Hot Encoding a variables categóricas (`genero`, `serv_internet`, `contrato`, `metodo_pago`).



- Balanceo de Clases:**

Se evaluó el desbalance entre clases (25.7% cancelaciones vs. 74.3% no cancelaciones) y se probaron técnicas como SMOTE (oversampling) y NearMiss (undersampling).

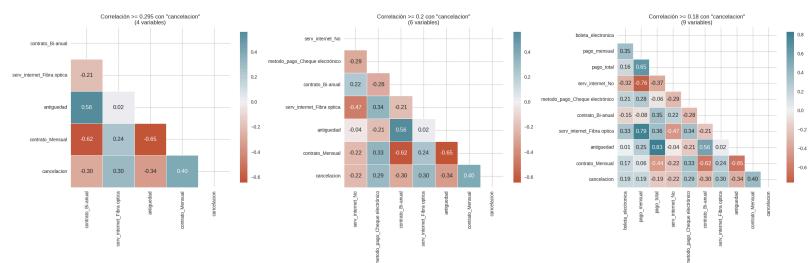
Proporción de Cancelación (Churn)



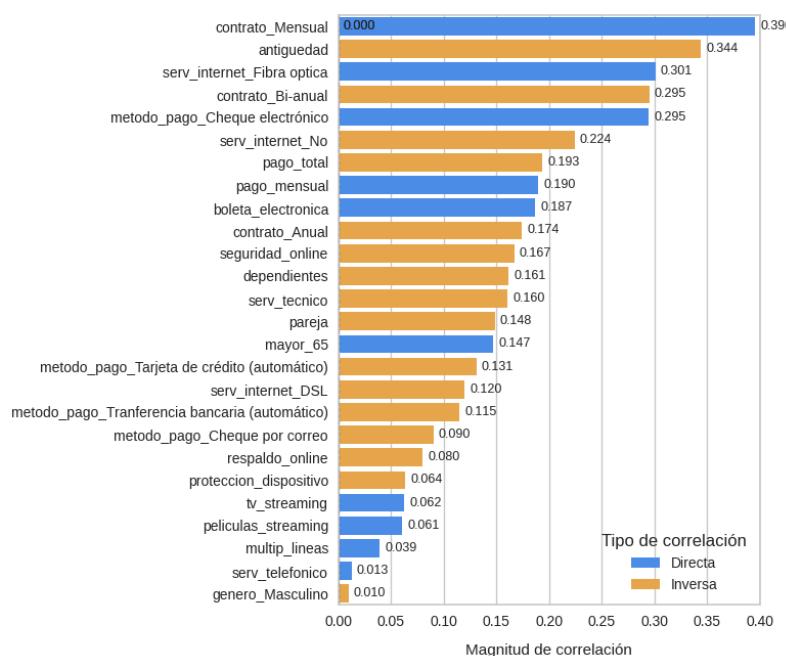
- Normalización:** Se estandarizaron los datos para modelos sensibles a la escala (Regresión Logística) usando StandardScaler y RobustScaler.

2. Análisis Exploratorio (EDA)

- Correlaciones:** Se identificaron variables con mayor correlación con el churn:
 - Directa: contrato_Mensual (0.396), serv_internet_Fibra optica (0.301).
 - Inversa: antiguedad (-0.344), contrato_Bi-anual (-0.295).



Correlación de cada variable con la cancelación



- Multicolinealidad:** Se calculó el VIF para detectar redundancias. Se eliminaron variables con VIF infinito (ej: serv_internet_DSL) y se corrigió la alta colinealidad entre pago_mensual y pago_total.

3. Modelado

- Modelos Evaluados:**

- Árbol de Decisión:** Precisión del 77% y AUC-ROC del 75%.

Evaluación del Modelo Decision Tree Classifier

