

Análisis y Predicción de Customer Churn en Telco

Descripción del Proyecto

Este notebook presenta un análisis completo de predicción de abandono de clientes (Customer Churn) en una empresa de telecomunicaciones. El objetivo es desarrollar modelos de Machine Learning que permitan identificar clientes con alta probabilidad de abandonar el servicio, facilitando estrategias de retención proactivas.

Metodología

1. **Análisis Exploratorio de Datos (EDA)**: Comprensión profunda del dataset y sus características
2. **Preprocesamiento**: Limpieza, transformación y preparación de datos
3. **Feature Engineering**: Creación de características derivadas relevantes
4. **Modelado**: Entrenamiento y comparación de múltiples algoritmos
5. **Optimización**: Ajuste de hiperparámetros y manejo de desbalanceo
6. **Evaluación**: Análisis de rendimiento con métricas apropiadas
7. **Interpretabilidad**: Análisis de importancia de características

Dataset

El dataset contiene información de 7,043 clientes con 21 variables que incluyen:

- Información demográfica (género, edad, dependientes)
- Servicios contratados (teléfono, internet, streaming)
- Información de cuenta (tipo de contrato, método de pago, cargos)
- Variable objetivo: Churn (Yes/No)

```
In [1]: # Importación de librerías necesarias
import os
import warnings
warnings.filterwarnings('ignore')

# Manipulación de datos
import numpy as np
import pandas as pd

# Visualización
import matplotlib.pyplot as plt
import seaborn as sns

# Pruebas estadísticas
from scipy.stats import chi2_contingency, ttest_ind, mannwhitneyu, normaltest

# Preprocesamiento
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Modelos
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb

# Métricas
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, classification_report, roc_auc_score, roc_curve,
    precision_recall_curve, average_precision_score
)

# Manejo de desbalanceo
from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline as ImbPipeline

# Optimización
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

# Configuración de visualización
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (12, 6)
plt.rcParams['font.size'] = 10

print("✓ Librerías importadas correctamente")
```

✓ Librerías importadas correctamente

1. Carga y Exploración Inicial de Datos

Cargamos el dataset y realizamos una primera inspección para entender su estructura.

```
In [2]: # Carga del dataset con manejo robusto de rutas
def cargar_datos():
    """Carga el dataset desde diferentes ubicaciones posibles"""
    drive_mounted = False
    # Intentar montar Google Drive
    try:
        from google.colab import drive
        print("Montando Google Drive...")
        drive.mount('/content/drive', force_remount=True)
        print("Google Drive montado exitosamente.")
        drive_mounted = True
    except Exception as e:
        print(f"No se pudo montar Google Drive: {e}")

    rutas_posibles = [
        'WA_Fn-UseC_-Telco-Customer-Churn.csv',
        './WA_Fn-UseC_-Telco-Customer-Churn.csv',
        '../WA_Fn-UseC_-Telco-Customer-Churn.csv',
    ]
    if drive_mounted:
        # Añadir ruta de Google Drive común
        rutas_posibles.append('/content/drive/MyDrive/WA_Fn-UseC_-Telco-Customer')
        # Añadir la ruta especificada por el usuario (directorio del notebook)
        notebook_dir_in_drive = '/content/drive/MyDrive/Colab Notebooks/Defensa'
        rutas_posibles.append(os.path.join(notebook_dir_in_drive, 'WA_Fn-UseC_-T'))
        # Añadir la ruta en el directorio actual de Colab si está dentro de Drive
        current_colab_dir = os.getcwd()
        if '/content/drive/MyDrive' in current_colab_dir:
            rutas_posibles.append(os.path.join(current_colab_dir, 'WA_Fn-UseC_-T'))

    for ruta in rutas_posibles:
        if os.path.exists(ruta):
            print(f"✓ Dataset encontrado en: {ruta}")
            return pd.read_csv(ruta)

    raise FileNotFoundError("No se encontró el archivo CSV. Verifica la ruta.")

# Cargar datos
df = cargar_datos()

# Información básica
print(f"\nDimensiones del dataset: {df.shape[0]} filas x {df.shape[1]} columnas")
print(f"\nPrimeras filas del dataset:")
df.head()
```

Montando Google Drive...
 Mounted at /content/drive
 Google Drive montado exitosamente.
 ✓ Dataset encontrado en: WA_Fn-UseC_-Telco-Customer-Churn.csv

Dimensiones del dataset: 7043 filas x 21 columnas

Primeras filas del dataset:

```
Out[2]:   customerID gender SeniorCitizen Partner Dependents tenure PhoneService MultipleLines InternetService OnlineSe
          0 7590-VHVEG Female      0     Yes      No      1      No No phone service      DSL
          1 5575-GNVDE Male       0      No      No     34      Yes      No      No      DSL
          2 3668-QPYBK Male       0      No      No      2      Yes      No      No      DSL
          3 7795-CFOCW Male       0      No      No     45      No No phone service      DSL
          4 9237-HQITU Female     0      No      No      2      Yes      No Fiber optic
```

5 rows x 21 columns

```
In [3]: # Información detallada del dataset
print("*"*80)
print("INFORMACIÓN DEL DATASET")
print("*"*80)

print("\n1. Tipos de datos:")
print(df.dtypes)

print("\n2. Información general:")
df.info()

print("\n3. Estadísticas descriptivas (variables numéricas):")
df.describe()
```

=====

INFORMACIÓN DEL DATASET

=====

1. Tipos de datos:

```
customerID          object
gender              object
SeniorCitizen       int64
Partner             object
Dependents          object
tenure              int64
PhoneService        object
MultipleLines        object
InternetService     object
OnlineSecurity      object
OnlineBackup         object
DeviceProtection    object
TechSupport          object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
MonthlyCharges      float64
TotalCharges        object
Churn               object
dtype: object
```

2. Información general:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 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 
 5   tenure           7043 non-null   int64  
 6   PhoneService     7043 non-null   object 
 7   MultipleLines    7043 non-null   object 
 8   InternetService  7043 non-null   object 
 9   OnlineSecurity   7043 non-null   object 
 10  OnlineBackup     7043 non-null   object 
 11  DeviceProtection 7043 non-null   object 
 12  TechSupport      7043 non-null   object 
 13  StreamingTV      7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling 7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges   7043 non-null   float64
 19  TotalCharges     7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

3. Estadísticas descriptivas (variables numéricas):

Out[3]:

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

2. Análisis de Calidad de Datos

2.1 Detección de Valores Faltantes y Anomalías

```
In [4]: # Análisis de valores faltantes
print("Valores faltantes por columna:")
missing = df.isnull().sum()
missing_pct = 100 * df.isnull().sum() / len(df)
missing_table = pd.DataFrame({
    'Valores Faltantes': missing,
    'Porcentaje': missing_pct
})
print(missing_table[missing_table['Valores Faltantes'] > 0])

# Detectar problema con TotalCharges (está como object)
print(f"\nTipo de dato de TotalCharges: {df['TotalCharges'].dtype}")
print(f"\nValores únicos en TotalCharges (primeros 10):")
print(df['TotalCharges'].unique()[:10])

# Detectar espacios en blanco en TotalCharges
espacios_blanco = df[df['TotalCharges'] == ' ']
print(f"\nRegistros con TotalCharges vacío: {len(espacios_blanco)}")
if len(espacios_blanco) > 0:
    print("\nCaracterísticas de registros con TotalCharges vacío:")
    print(espacios_blanco[['customerID', 'tenure', 'MonthlyCharges', 'TotalCharges']])
```

Valores faltantes por columna:

Empty DataFrame

Columns: [Valores Faltantes, Porcentaje]

Index: []

Tipo de dato de TotalCharges: object

Valores únicos en TotalCharges (primeros 10):

['29.85' '1889.5' '108.15' '1840.75' '151.65' '820.5' '1949.4' '301.9'
 '3046.05' '3487.95']

Registros con TotalCharges vacío: 11

Características de registros con TotalCharges vacío:

	customerID	tenure	MonthlyCharges	TotalCharges
488	4472-LVYGI	0	52.55	
753	3115-CZMZD	0	20.25	
936	5709-LVOEQ	0	80.85	
1082	4367-NUYAO	0	25.75	
1340	1371-DWPAZ	0	56.05	

2.2 Limpieza de Datos

Identificamos que `TotalCharges` contiene espacios en blanco y está almacenado como texto. Procedemos a limpiar estos datos.

```
In [5]: # Limpieza de TotalCharges
# Convertir espacios en blanco a NaN
df['TotalCharges'] = df['TotalCharges'].replace(' ', np.nan)

# Convertir a numérico
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')

# Análisis de registros con TotalCharges faltante
print(f"Registros con TotalCharges NaN: {df['TotalCharges'].isna().sum()}")


# Estrategia: Para clientes nuevos (tenure=0), TotalCharges debería ser igual a
df.loc[df['TotalCharges'].isna(), 'TotalCharges'] = df.loc[df['TotalCharges'].isna(),
print(f"\nDespués de la imputación:")
print(f"Registros con TotalCharges NaN: {df['TotalCharges'].isna().sum()}")


# Verificar que no hay valores faltantes
print(f"\nTotal de valores faltantes en el dataset: {df.isnull().sum().sum()}"
```

Registros con TotalCharges NaN: 11

Después de la imputación:

Registros con TotalCharges NaN: 0

Total de valores faltantes en el dataset: 0

3. Análisis Exploratorio de Datos (EDA)

3.1 Análisis de la Variable Objetivo (Churn)

```
In [6]: # Análisis de la distribución de Churn
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

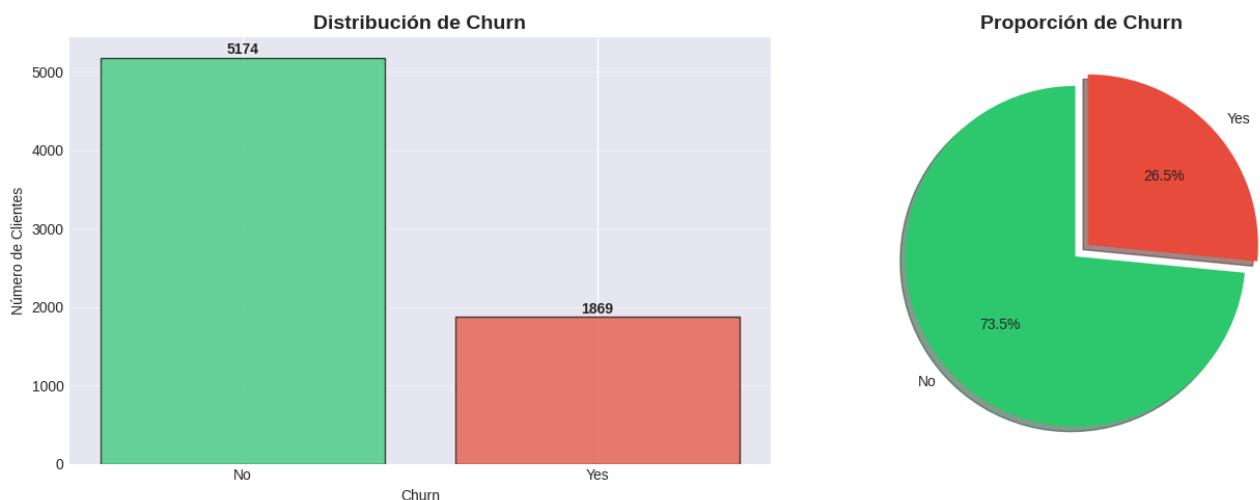
# Gráfico de barras
churn_counts = df['Churn'].value_counts()
axes[0].bar(churn_counts.index, churn_counts.values, color=['#2ecc71', '#e74c3c'])
axes[0].set_title('Distribución de Churn', fontsize=14, fontweight='bold')
axes[0].set_xlabel('Churn')
axes[0].set_ylabel('Número de Clientes')
axes[0].grid(axis='y', alpha=0.3)

# Añadir valores en las barras
for i, v in enumerate(churn_counts.values):
    axes[0].text(i, v + 50, str(v), ha='center', fontweight='bold')

# Gráfico de pastel
colors = ['#2ecc71', '#e74c3c']
explode = (0.05, 0.05)
axes[1].pie(churn_counts.values, labels=churn_counts.index, autopct='%1.1f%%',
            colors=colors, explode=explode, shadow=True, startangle=90)
axes[1].set_title('Proporción de Churn', fontsize=14, fontweight='bold')

plt.tight_layout()
plt.show()

# Estadísticas
print("\nEstadísticas de Churn:")
print(f"Total de clientes: {len(df)}")
print(f"Clientes que NO abandonaron: {churn_counts['No']} ({100*churn_counts['No']:.2f}%)")
print(f"Clientes que SÍ abandonaron: {churn_counts['Yes']} ({100*churn_counts['Yes']:.2f}%)")
print(f"\nRatio de desbalanceo: {churn_counts['No']}/{churn_counts['Yes']:.2f}:1")
```



Estadísticas de Churn:

Total de clientes: 7043

Clientes que NO abandonaron: 5174 (73.46%)

Clientes que SÍ abandonaron: 1869 (26.54%)

Ratio de desbalanceo: 2.77:1

3.2 Análisis de Variables Categóricas

Analizamos la relación entre las variables categóricas y el churn.

```
In [7]: # Identificar variables categóricas (excluyendo customerID y Churn)
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()
categorical_cols.remove('customerID')
categorical_cols.remove('Churn')

print(f"Variables categóricas a analizar: {len(categorical_cols)}")
print(categorical_cols)

# Visualizar las variables categóricas más importantes
important_cats = ['Contract', 'InternetService', 'PaymentMethod', 'TechSupport',
                   'OnlineSecurity', 'PaperlessBilling']

fig, axes = plt.subplots(2, 3, figsize=(18, 10))
axes = axes.ravel()

for idx, col in enumerate(important_cats):
    # Crear tabla de contingencia
    ct = pd.crosstab(df[col], df['Churn'], normalize='index') * 100

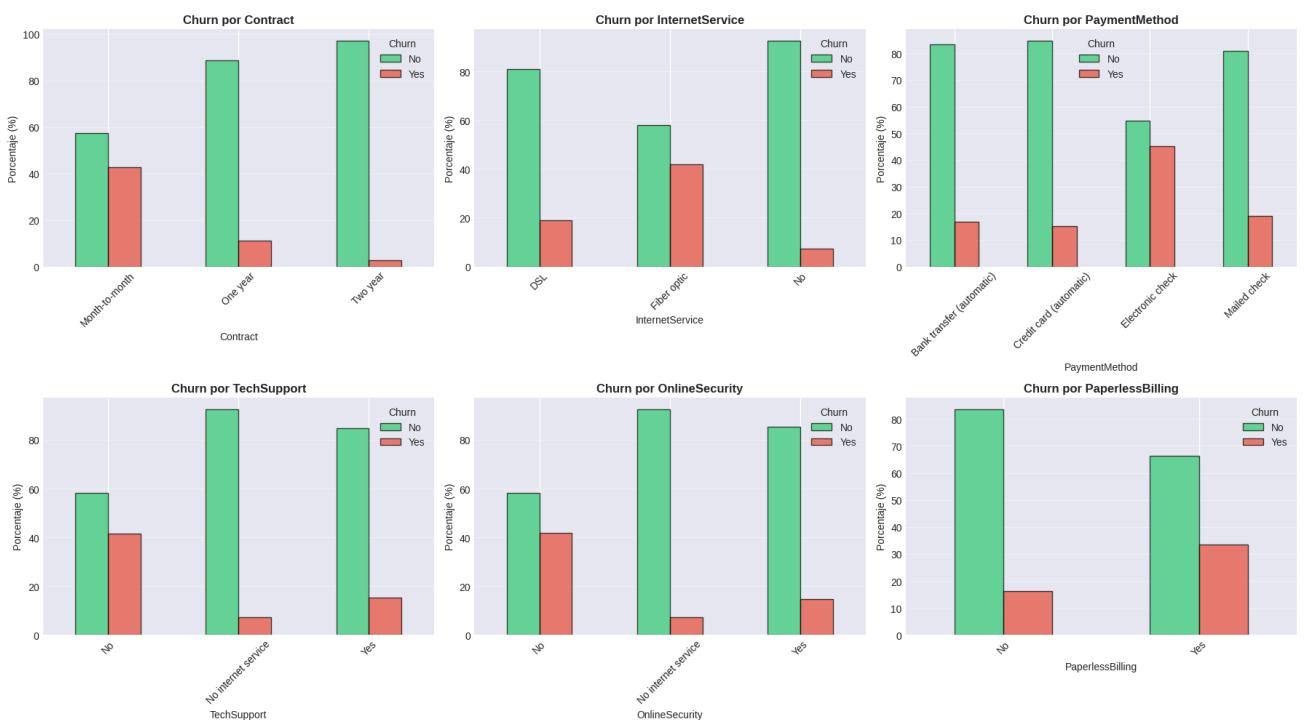
    # Graficar
    ct.plot(kind='bar', ax=axes[idx], color=['#2ecc71', '#e74c3c'], alpha=0.7, edgecolor='black')
    axes[idx].set_title(f'Churn por {col}', fontsize=12, fontweight='bold')
    axes[idx].set_xlabel(col)
    axes[idx].set_ylabel('Porcentaje (%)')
    axes[idx].legend(title='Churn', labels=['No', 'Yes'])
    axes[idx].grid(axis='y', alpha=0.3)
    axes[idx].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()

# Análisis estadístico
print("\nTasa de Churn por categoría:")
for col in important_cats:
    print(f"\n{col}:")
    churn_rate = df.groupby(col)['Churn'].apply(lambda x: (x=='Yes').sum()/len(x))
    print(churn_rate.sort_values(ascending=False))
```

Variables categóricas a analizar: 15

['gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod']



Tasa de Churn por categoría:

Contract:

Contract

Month-to-month	42.709677
One year	11.269518
Two year	2.831858

Name: Churn, dtype: float64

InternetService:

InternetService

Fiber optic	41.892765
DSL	18.959108
No	7.404980

Name: Churn, dtype: float64

PaymentMethod:

PaymentMethod

Electronic check	45.285412
Mailed check	19.106700
Bank transfer (automatic)	16.709845
Credit card (automatic)	15.243101

Name: Churn, dtype: float64

TechSupport:

TechSupport

No	41.635474
Yes	15.166341
No internet service	7.404980

Name: Churn, dtype: float64

OnlineSecurity:

OnlineSecurity

No	41.766724
Yes	14.611194
No internet service	7.404980

Name: Churn, dtype: float64

PaperlessBilling:

PaperlessBilling

Yes	33.565092
No	16.330084

Name: Churn, dtype: float64

3.3 Análisis de Variables Numéricas

```
In [8]: # Variables numéricas
numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']

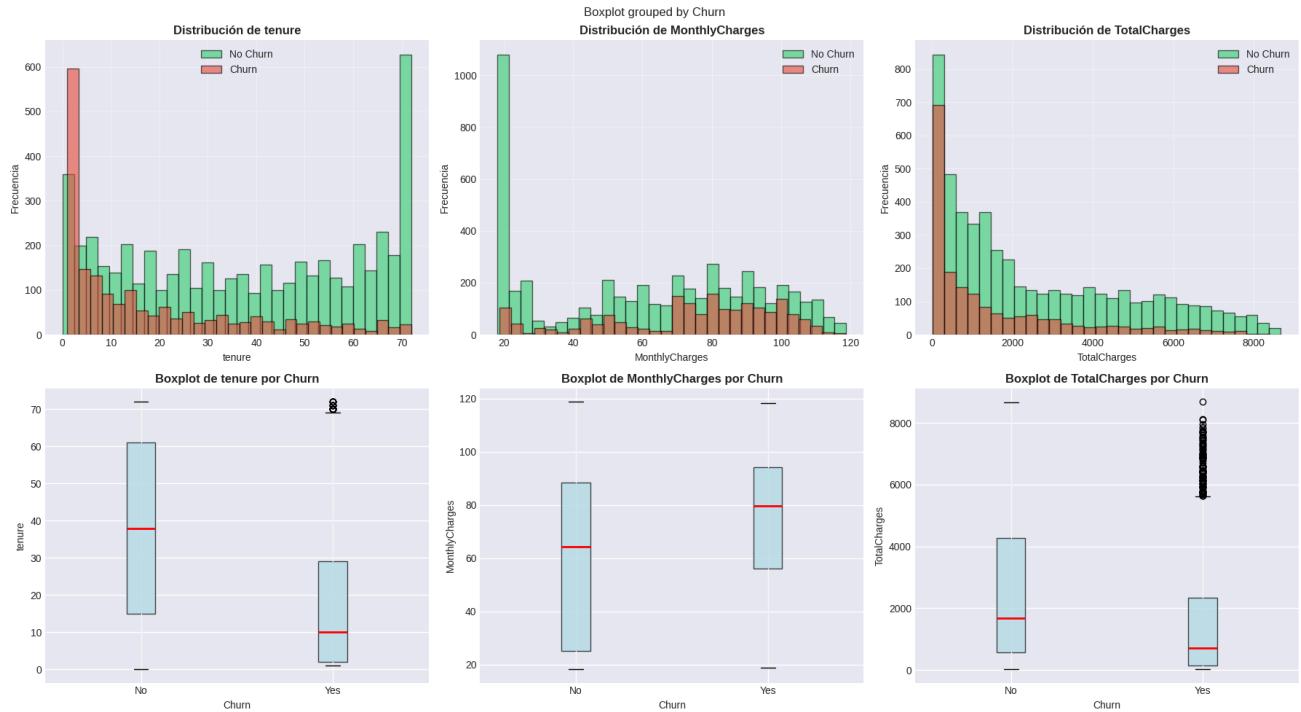
# Visualización de distribuciones
fig, axes = plt.subplots(2, 3, figsize=(18, 10))

for idx, col in enumerate(numerical_cols):
    # Histograma
    axes[0, idx].hist(df[df['Churn'] == 'No'][col], bins=30, alpha=0.6, label='No')
    axes[0, idx].hist(df[df['Churn'] == 'Yes'][col], bins=30, alpha=0.6, label='Ch')
    axes[0, idx].set_title(f'Distribución de {col}', fontsize=12, fontweight='bo')
    axes[0, idx].set_xlabel(col)
    axes[0, idx].set_ylabel('Frecuencia')
    axes[0, idx].legend()
    axes[0, idx].grid(alpha=0.3)

    # Boxplot
    df.boxplot(column=col, by='Churn', ax=axes[1, idx], patch_artist=True,
               boxprops=dict(facecolor='lightblue', alpha=0.7),
               medianprops=dict(color='red', linewidth=2))
    axes[1, idx].set_title(f'Boxplot de {col} por Churn', fontsize=12, fontweight='bo')
    axes[1, idx].set_xlabel('Churn')
    axes[1, idx].set_ylabel(col)
    plt.sca(axes[1, idx])
    plt.xticks([1, 2], ['No', 'Yes'])

plt.tight_layout()
plt.show()

# Estadísticas por grupo
print("\nEstadísticas de variables numéricas por Churn:\n")
print(df.groupby('Churn')[numerical_cols].describe().T)
```



Estadísticas de variables numéricas por Churn:

Churn		No	Yes
tenure	count	5174.000000	1869.000000
	mean	37.569965	17.979133
	std	24.113777	19.531123
	min	0.000000	1.000000
	25%	15.000000	2.000000
	50%	38.000000	10.000000
	75%	61.000000	29.000000
	max	72.000000	72.000000
MonthlyCharges	count	5174.000000	1869.000000
	mean	61.265124	74.441332
	std	31.092648	24.666053
	min	18.250000	18.850000
	25%	25.100000	56.150000
	50%	64.425000	79.650000
	75%	88.400000	94.200000
	max	118.750000	118.350000
TotalCharges	count	5174.000000	1869.000000
	mean	2549.999497	1531.796094
	std	2329.858843	1890.822994
	min	18.800000	18.850000
	25%	572.900000	134.500000
	50%	1679.525000	703.550000
	75%	4262.850000	2331.300000
	max	8672.450000	8684.800000

3.4 Análisis de Correlaciones

```
In [9]: # Preparar datos para correlación
df_corr = df.copy()

# Convertir variables categóricas binarias a numéricas
binary_cols = ['gender', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBill'
for col in binary_cols:
    if col in df_corr.columns:
        df_corr[col] = df_corr[col].map({'Yes': 1, 'No': 0, 'Male': 1, 'Female': 0})

# Seleccionar solo columnas numéricas
numeric_df = df_corr.select_dtypes(include=[np.number])

# Calcular matriz de correlación
correlation_matrix = numeric_df.corr()

# Visualizar
plt.figure(figsize=(12, 10))
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
sns.heatmap(correlation_matrix, mask=mask, annot=True, fmt='.2f', cmap='coolwarm',
            center=0, square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Matriz de Correlación', fontsize=16, fontweight='bold', pad=20)
plt.tight_layout()
plt.show()

# Correlaciones más fuertes con Churn
print("\nCorrelaciones con Churn (ordenadas por valor absoluto):\n")
churn_corr = correlation_matrix['Churn'].drop('Churn').sort_values(key=abs, ascending=False)
print(churn_corr)
```

Matriz de Correlación



Correlaciones con Churn (ordenadas por valor absoluto):

```
tenure           -0.352229
TotalCharges     -0.198347
MonthlyCharges   0.193356
PaperlessBilling 0.191825
Dependents        -0.164221
SeniorCitizen    0.150889
Partner           -0.150448
PhoneService      0.011942
gender            -0.008612
Name: Churn, dtype: float64
```

3.5. Comprobación de Hipótesis Estadísticas

Realizamos pruebas estadísticas formales para validar las relaciones observadas en el análisis exploratorio.

Nivel de Significancia

- $\alpha = 0.05$ (5%)
- Si $p\text{-value} < 0.05$: Rechazamos H_0 (hay evidencia estadística significativa)
- Si $p\text{-value} \geq 0.05$: No rechazamos H_0 (no hay evidencia suficiente)

```
In [10]: # Configuración
alpha = 0.05
print("*"*80)
print("COMPROBACIÓN DE HIPÓTESIS ESTADÍSTICAS")
print("*"*80)
print(f"\nNivel de significancia: α = {alpha}")
print("\nCriterio de decisión:")
print("  • p-value < 0.05 → Rechazamos H₀ (relación estadísticamente significativa)")
print("  • p-value ≥ 0.05 → No rechazamos H₀ (no hay evidencia suficiente)")
print("\n" + "*"*80)

=====
COMPROBACIÓN DE HIPÓTESIS ESTADÍSTICAS
=====

Nivel de significancia: α = 0.05

Criterio de decisión:
  • p-value < 0.05 → Rechazamos H₀ (relación estadísticamente significativa)
  • p-value ≥ 0.05 → No rechazamos H₀ (no hay evidencia suficiente)

=====
```

Hipótesis 1: Tipo de Contrato vs Churn

H_0 : El tipo de contrato NO está asociado con el churn

H_1 : El tipo de contrato SÍ está asociado con el churn

Prueba: Chi-cuadrado (χ^2)

```
In [11]: # Hipótesis 1: Contract vs Churn
print("\n" + "*80")
print("HIPÓTESIS 1: TIPO DE CONTRATO VS CHURN")
print("*80")

# Tabla de contingencia
contingency_table = pd.crosstab(df['Contract'], df['Churn'])
print("\nTabla de Contingencia:")
print(contingency_table)

# Prueba Chi-cuadrado
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"\nResultados de la Prueba Chi-cuadrado:")
print(f" • Estadístico  $\chi^2$ : {chi2:.4f}")
print(f" • p-value: {p_value:.6f}")
print(f" • Grados de libertad: {dof}")

# Interpretación
print("\n'*80")
print("CONCLUSIÓN:")
if p_value < alpha:
    print(f" ✓ p-value ({p_value:.6f}) <  $\alpha$  ({alpha})")
    print(" ✓ RECHAZAMOS  $H_0$ ")
    print(" ✓ El tipo de contrato SÍ está significativamente asociado con el churn")
else:
    print(f" ✗ p-value ({p_value:.6f})  $\geq$   $\alpha$  ({alpha})")
    print(" ✗ NO RECHAZAMOS  $H_0$ ")
    print(" ✗ No hay evidencia suficiente de asociación")
print("\n'*80")

# Visualización
fig, axes = plt.subplots(1, 2, figsize=(16, 5))

# Gráfico 1: Tabla de contingencia
contingency_pct = pd.crosstab(df['Contract'], df['Churn'], normalize='index') * 100
contingency_pct.plot(kind='bar', ax=axes[0], color=['#2ecc71', '#e74c3c'], alpha=0.3)
axes[0].set_title('Tasa de Churn por Tipo de Contrato', fontsize=14, fontweight='bold')
axes[0].set_xlabel('Tipo de Contrato', fontsize=12)
axes[0].set_ylabel('Porcentaje (%)', fontsize=12)
axes[0].legend(title='Churn', labels=['No', 'Sí'])
axes[0].grid(axis='y', alpha=0.3)
axes[0].tick_params(axis='x', rotation=45)

# Gráfico 2: Resultado de la prueba
axes[1].axis('off')
result_text = f"""Prueba Chi-cuadrado ( $\chi^2$ )
 $H_0$ : Contract y Churn son independientes
 $H_1$ : Contract y Churn están asociados

Estadístico  $\chi^2$ : {chi2:.4f}
p-value: {p_value:.6f}
Grados de libertad: {dof}

Decisión ( $\alpha = {alpha}$ ):
{'RECHAZAMOS  $H_0$ ' if p_value < alpha else 'NO RECHAZAMOS  $H_0$ '}

Conclusión:
{'El tipo de contrato SÍ está' if p_value < alpha else 'No hay evidencia de que' +
 'significativamente asociado con el churn' if p_value < alpha else 'asociado con el churn'}"""

axes[1].text(0.1, 0.5, result_text, fontsize=12, verticalalignment='center',
            bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))

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

HIPÓTESIS 1: TIPO DE CONTRATO VS CHURN

Tabla de Contingencia:

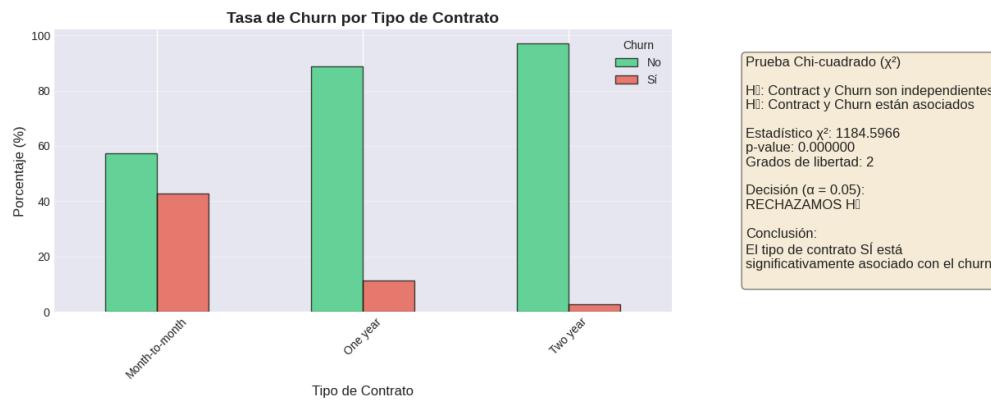
Churn	No	Yes
Contract		
Month-to-month	2220	1655
One year	1307	166
Two year	1647	48

Resultados de la Prueba Chi-cuadrado:

- Estadístico χ^2 : 1184.5966
- p-value: 0.000000
- Grados de libertad: 2

CONCLUSIÓN:

- p-value (0.000000) < α (0.05)
 - RECHAZAMOS H_0
 - El tipo de contrato SÍ está significativamente asociado con el churn
-
-



Hipótesis 2: Método de Pago vs Churn

H_0 : El método de pago NO está asociado con el churn

H_1 : El método de pago SÍ está asociado con el churn

Prueba: Chi-cuadrado (χ^2)

```
In [12]: # Hipótesis 2: PaymentMethod vs Churn
print("\n" + "*80")
print("HIPÓTESIS 2: MÉTODO DE PAGO VS CHURN")
print("*80")

# Tabla de contingencia
contingency_table = pd.crosstab(df['PaymentMethod'], df['Churn'])
print("\nTabla de Contingencia:")
print(contingency_table)

# Prueba Chi-cuadrado
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"\nResultados de la Prueba Chi-cuadrado:")
print(f"• Estadístico  $\chi^2$ : {chi2:.4f}")
print(f"• p-value: {p_value:.6f}")
print(f"• Grados de libertad: {dof}")

# Interpretación
print("\n'*80")
print("CONCLUSIÓN:")
if p_value < alpha:
    print(f"✓ p-value ({p_value:.6f}) <  $\alpha$  ({alpha})")
    print("✓ RECHAZAMOS  $H_0$ ")
    print("✓ El método de pago SÍ está significativamente asociado con el churn")
else:
    print(f"✗ p-value ({p_value:.6f})  $\geq$   $\alpha$  ({alpha})")
    print("✗ NO RECHAZAMOS  $H_0$ ")
    print("✗ No hay evidencia suficiente de asociación")
print('*80')
```

=====

HIPÓTESIS 2: MÉTODO DE PAGO VS CHURN

=====

Tabla de Contingencia:

	No	Yes
PaymentMethod		
Bank transfer (automatic)	1286	258
Credit card (automatic)	1290	232
Electronic check	1294	1071
Mailed check	1304	308

Resultados de la Prueba Chi-cuadrado:

- Estadístico χ^2 : 648.1423
- p-value: 0.000000
- Grados de libertad: 3

=====

CONCLUSIÓN:

- ✓ p-value (0.000000) < α (0.05)
 - ✓ RECHAZAMOS H_0
 - ✓ El método de pago SÍ está significativamente asociado con el churn
- =====

Hipótesis 3: Servicio de Internet vs Churn

H_0 : El tipo de servicio de internet NO está asociado con el churn

H_1 : El tipo de servicio de internet SÍ está asociado con el churn

Prueba: Chi-cuadrado (χ^2)

```
In [13]: # Hipótesis 3: InternetService vs Churn
print("\n" + "*80")
print("HIPÓTESIS 3: SERVICIO DE INTERNET VS CHURN")
print("*80")

# Tabla de contingencia
contingency_table = pd.crosstab(df['InternetService'], df['Churn'])
print("\nTabla de Contingencia:")
print(contingency_table)

# Prueba Chi-cuadrado
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"\nResultados de la Prueba Chi-cuadrado:")
print(f"• Estadístico  $\chi^2$ : {chi2:.4f}")
print(f"• p-value: {p_value:.6f}")
print(f"• Grados de libertad: {dof}")

# Interpretación
print("\n'*80")
print("CONCLUSIÓN:")
if p_value < alpha:
    print(f"✓ p-value ({p_value:.6f}) <  $\alpha$  ({alpha})")
    print("✓ RECHAZAMOS  $H_0$ ")
    print("✓ El servicio de internet SÍ está significativamente asociado con el churn")
else:
    print(f"✗ p-value ({p_value:.6f})  $\geq$   $\alpha$  ({alpha})")
    print("✗ NO RECHAZAMOS  $H_0$ ")
    print("✗ No hay evidencia suficiente de asociación")
print("\n'*80")
```

HIPÓTESIS 3: SERVICIO DE INTERNET VS CHURN

Tabla de Contingencia:

	No	Yes
InternetService		
DSL	1962	459
Fiber optic	1799	1297
No	1413	113

Resultados de la Prueba Chi-cuadrado:

- Estadístico χ^2 : 732.3096
- p-value: 0.000000
- Grados de libertad: 2

CONCLUSIÓN:

- ✓ p-value (0.000000) < α (0.05)
 - ✓ RECHAZAMOS H_0
 - ✓ El servicio de internet SÍ está significativamente asociado con el churn
-

Hipótesis 4: Tenure (Antigüedad) vs Churn

H_0 : La antigüedad promedio es igual entre clientes que abandonan y los que no

H_1 : La antigüedad promedio es diferente entre clientes que abandonan y los que no

Prueba: Mann-Whitney U (no paramétrica)

```
In [14]: # Hipótesis 4: tenure vs Churn
print("\n" + "*80")
print("HIPÓTESIS 4: ANTIGÜEDAD (TENURE) VS CHURN")
print("*80")

# Separar datos por grupo
tenure_no_churn = df[df['Churn'] == 'No']['tenure']
tenure_yes_churn = df[df['Churn'] == 'Yes']['tenure']

print("\nEstadísticas Descriptivas:")
print(f" No Churn - Media: {tenure_no_churn.mean():.2f} meses, Mediana: {tenure_no_churn.median():.2f}")
print(f" Sí Churn - Media: {tenure_yes_churn.mean():.2f} meses, Mediana: {tenure_yes_churn.median():.2f}")
print(f" Diferencia de medias: {abs(tenure_no_churn.mean() - tenure_yes_churn.mean()):.2f} meses")

# Prueba Mann-Whitney U (no paramétrica)
statistic, p_value = mannwhitneyu(tenure_no_churn, tenure_yes_churn, alternative='two-sided')

print("\nResultados de la Prueba Mann-Whitney U:")
print(f" • Estadístico U: {statistic:.4f}")
print(f" • p-value: {p_value:.6f}")

# Interpretación
print("\n'*80")
print("CONCLUSIÓN:")
if p_value < alpha:
    print(f" ✓ p-value ({p_value:.6f}) < α ({alpha})")
    print(f" ✓ RECHAZAMOS H₀")
    print(f" ✓ La antigüedad SÍ es significativamente diferente entre grupos")
    print(f" ✓ Los clientes que NO abandonan tienen {tenure_no_churn.mean() - tenure_yes_churn.mean():.2f} meses más")
else:
    print(f" ✗ p-value ({p_value:.6f}) ≥ α ({alpha})")
    print(f" ✗ NO RECHAZAMOS H₀")
    print(f" ✗ No hay evidencia suficiente de diferencia")
print('*80')

# Visualización
fig, axes = plt.subplots(1, 2, figsize=(16, 5))

# Boxplot
df.boxplot(column='tenure', by='Churn', ax=axes[0], patch_artist=True,
            boxprops=dict(facecolor='lightblue', alpha=0.7),
            medianprops=dict(color='red', linewidth=2))
axes[0].set_title('Distribución de Antigüedad por Churn', fontsize=14, fontweight='bold')
axes[0].set_xlabel('Churn', fontsize=12)
axes[0].set_ylabel('Antigüedad (meses)', fontsize=12)
axes[0].grid(axis='y', alpha=0.3)
plt.sca(axes[0])
plt.xticks([1, 2], ['No', 'Sí'])

# Histogramas superpuestos
axes[1].hist(tenure_no_churn, bins=30, alpha=0.6, label='No Churn', color="#2ecc71")
axes[1].hist(tenure_yes_churn, bins=30, alpha=0.6, label='Sí Churn', color="#e74c3c")
axes[1].axvline(tenure_no_churn.mean(), color="#2ecc71", linestyle='--', linewidth=2)
axes[1].axvline(tenure_yes_churn.mean(), color="#e74c3c", linestyle='--', linewidth=2)
axes[1].set_title('Distribución de Antigüedad por Grupo', fontsize=14, fontweight='bold')
axes[1].set_xlabel('Antigüedad (meses)', fontsize=12)
axes[1].set_ylabel('Frecuencia', fontsize=12)
axes[1].legend()
axes[1].grid(axis='y', alpha=0.3)

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

HIPÓTESIS 4: ANTIGÜEDAD (TENURE) VS CHURN

Estadísticas Descriptivas:

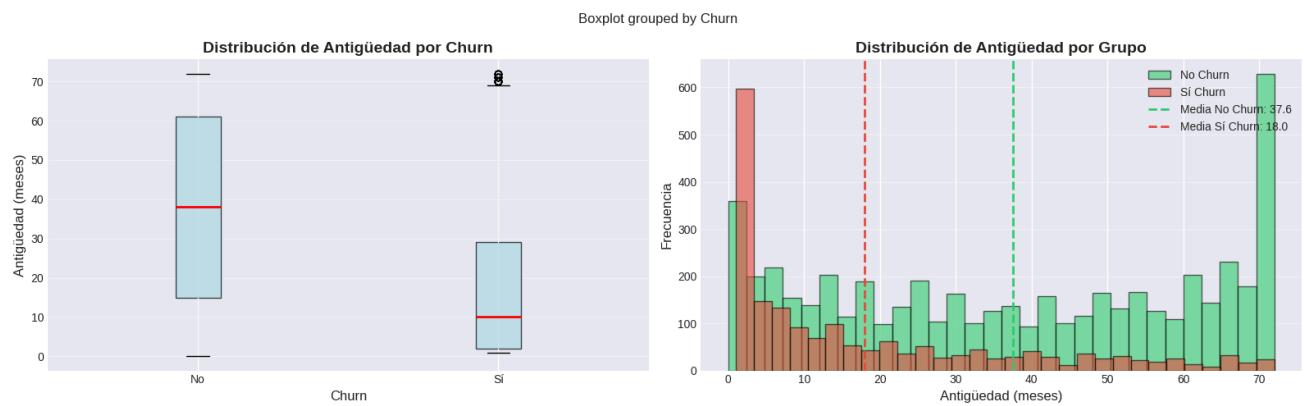
No Churn - Media: 37.57 meses, Mediana: 38.00
Sí Churn - Media: 17.98 meses, Mediana: 10.00
Diferencia de medias: 19.59 meses

Resultados de la Prueba Mann-Whitney U:

- Estadístico U: 7154668.0000
- p-value: 0.000000

CONCLUSIÓN:

- ✓ p-value (0.000000) < α (0.05)
- ✓ RECHAZAMOS H_0
- ✓ La antigüedad SÍ es significativamente diferente entre grupos
- ✓ Los clientes que NO abandonan tienen 19.6 meses más de antigüedad en promedio



Hipótesis 5: Cargos Mensuales vs Churn

H_0 : Los cargos mensuales promedio son iguales entre clientes que abandonan y los que no
 H_1 : Los cargos mensuales promedio son diferentes entre clientes que abandonan y los que no

Prueba: Mann-Whitney U (no paramétrica)

```
In [15]: # Hipótesis 5: MonthlyCharges vs Churn
print("\n" + "*80")
print("HIPÓTESIS 5: CARGOS MENSUALES VS CHURN")
print("*80")

# Separar datos por grupo
monthly_no_churn = df[df['Churn'] == 'No']['MonthlyCharges']
monthly_yes_churn = df[df['Churn'] == 'Yes']['MonthlyCharges']

print("\nEstadísticas Descriptivas:")
print(f" No Churn - Media: ${monthly_no_churn.mean():.2f}, Mediana: ${monthly_no_churn.median():.2f}")
print(f" Sí Churn - Media: ${monthly_yes_churn.mean():.2f}, Mediana: ${monthly_yes_churn.median():.2f}")
print(f" Diferencia de medias: ${abs(monthly_no_churn.mean() - monthly_yes_churn.mean()):.2f}")

# Prueba Mann-Whitney U
statistic, p_value = mannwhitneyu(monthly_no_churn, monthly_yes_churn, alternative='two-sided')

print(f"\nResultados de la Prueba Mann-Whitney U:")
print(f" • Estadístico U: {statistic:.4f}")
print(f" • p-value: {p_value:.6f}")

# Interpretación
print(f"\n{'*80}'")
print("CONCLUSIÓN:")
if p_value < alpha:
    print(f" ✓ p-value ({p_value:.6f}) < α ({alpha})")
    print(f" ✓ RECHAZAMOS H₀")
    print(f" ✓ Los cargos mensuales SÍ son significativamente diferentes entre grupos")
    if monthly_yes_churn.mean() > monthly_no_churn.mean():
        print(f" ✓ Los clientes que SÍ abandonan pagan ${monthly_yes_churn.mean():.2f} más al mes en promedio")
    else:
        print(f" ✓ Los clientes que NO abandonan pagan ${monthly_no_churn.mean():.2f} más al mes en promedio")
else:
    print(f" ✗ p-value ({p_value:.6f}) ≥ α ({alpha})")
    print(f" ✗ NO RECHAZAMOS H₀")
    print(f" ✗ No hay evidencia suficiente de diferencia")
print(f"{'*80}'")
```

=====

HIPÓTESIS 5: CARGOS MENSUALES VS CHURN

=====

Estadísticas Descriptivas:

No Churn - Media: \$61.27, Mediana: \$64.43
Sí Churn - Media: \$74.44, Mediana: \$79.65
Diferencia de medias: \$13.18

Resultados de la Prueba Mann-Whitney U:

- Estadístico U: 3667080.5000
- p-value: 0.000000

=====

CONCLUSIÓN:

- ✓ p-value (0.000000) < α (0.05)
 - ✓ RECHAZAMOS H₀
 - ✓ Los cargos mensuales SÍ son significativamente diferentes entre grupos
 - ✓ Los clientes que SÍ abandonan pagan \$13.18 más al mes en promedio
- =====

Hipótesis 6: Soporte Técnico vs Churn

H_0 : Tener soporte técnico NO está asociado con el churn

H_1 : Tener soporte técnico SÍ está asociado con el churn

Prueba: Chi-cuadrado (χ^2)

```
In [16]: # Hipótesis 6: TechSupport vs Churn
print("\n" + "*80")
print("HIPÓTESIS 6: SOPORTE TÉCNICO VS CHURN")
print("*80")

# Tabla de contingencia
contingency_table = pd.crosstab(df['TechSupport'], df['Churn'])
print("\nTabla de Contingencia:")
print(contingency_table)

# Prueba Chi-cuadrado
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"\nResultados de la Prueba Chi-cuadrado:")
print(f"• Estadístico  $\chi^2$ : {chi2:.4f}")
print(f"• p-value: {p_value:.6f}")
print(f"• Grados de libertad: {dof}")

# Interpretación
print("\n'*80")
print("CONCLUSIÓN:")
if p_value < alpha:
    print(f"✓ p-value ({p_value:.6f}) <  $\alpha$  ({alpha})")
    print("✓ RECHAZAMOS  $H_0$ ")
    print("✓ El soporte técnico SÍ está significativamente asociado con el churn")
else:
    print(f"✗ p-value ({p_value:.6f})  $\geq$   $\alpha$  ({alpha})")
    print("✗ NO RECHAZAMOS  $H_0$ ")
    print("✗ No hay evidencia suficiente de asociación")
print('*80')
```

HIPÓTESIS 6: SOPORTE TÉCNICO VS CHURN

Tabla de Contingencia:

	No	Yes
TechSupport		
No	2027	1446
No internet service	1413	113
Yes	1734	310

Resultados de la Prueba Chi-cuadrado:

- Estadístico χ^2 : 828.1971
- p-value: 0.000000
- Grados de libertad: 2

CONCLUSIÓN:

- ✓ p-value (0.000000) < α (0.05)
 - ✓ RECHAZAMOS H_0
 - ✓ El soporte técnico SÍ está significativamente asociado con el churn
-

Hipótesis 7: Facturación sin Papel vs Churn

H_0 : La facturación sin papel NO está asociada con el churn

H_1 : La facturación sin papel SÍ está asociada con el churn

Prueba: Chi-cuadrado (χ^2)

```
In [17]: # Hipótesis 7: PaperlessBilling vs Churn
print("\n" + "*80")
print("HIPÓTESIS 7: FACTURACIÓN SIN PAPEL VS CHURN")
print("*80")

# Tabla de contingencia
contingency_table = pd.crosstab(df['PaperlessBilling'], df['Churn'])
print("\nTabla de Contingencia:")
print(contingency_table)

# Prueba Chi-cuadrado
chi2, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"\nResultados de la Prueba Chi-cuadrado:")
print(f"• Estadístico  $\chi^2$ : {chi2:.4f}")
print(f"• p-value: {p_value:.6f}")
print(f"• Grados de libertad: {dof}")

# Interpretación
print("\n'*80")
print("CONCLUSIÓN:")
if p_value < alpha:
    print(f"✓ p-value ({p_value:.6f}) <  $\alpha$  ({alpha})")
    print("✓ RECHAZAMOS  $H_0$ ")
    print("✓ La facturación sin papel SÍ está significativamente asociada con el churn")
else:
    print(f"✗ p-value ({p_value:.6f})  $\geq$   $\alpha$  ({alpha})")
    print("✗ NO RECHAZAMOS  $H_0$ ")
    print("✗ No hay evidencia suficiente de asociación")
print('*80')
```

=====

HIPÓTESIS 7: FACTURACIÓN SIN PAPEL VS CHURN

=====

Tabla de Contingencia:

	No	Yes
PaperlessBilling		
No	2403	469
Yes	2771	1400

Resultados de la Prueba Chi-cuadrado:

- Estadístico χ^2 : 258.2776
- p-value: 0.000000
- Grados de libertad: 1

=====

CONCLUSIÓN:

- ✓ p-value (0.000000) < α (0.05)
 - ✓ RECHAZAMOS H_0
 - ✓ La facturación sin papel SÍ está significativamente asociada con el churn
- =====



Resumen de Pruebas de Hipótesis

Consolidamos los resultados de todas las pruebas realizadas.

In [18]:

```
# Crear tabla resumen de todas las hipótesis
print("\n" + "*100)
print("RESUMEN DE PRUEBAS DE HIPÓTESIS ESTADÍSTICAS")
print("*100)

# Ejecutar todas las pruebas y guardar resultados
results = []

# H1: Contract
ct = pd.crosstab(df['Contract'], df['Churn'])
chi2, p_val, _, _ = chi2_contingency(ct)
results.append(['Contract vs Churn', 'Chi-cuadrado', chi2, p_val, 'Sí' if p_val < 0.05 else 'No'])

# H2: PaymentMethod
ct = pd.crosstab(df['PaymentMethod'], df['Churn'])
chi2, p_val, _, _ = chi2_contingency(ct)
results.append(['PaymentMethod vs Churn', 'Chi-cuadrado', chi2, p_val, 'Sí' if p_val < 0.05 else 'No'])

# H3: InternetService
ct = pd.crosstab(df['InternetService'], df['Churn'])
chi2, p_val, _, _ = chi2_contingency(ct)
results.append(['InternetService vs Churn', 'Chi-cuadrado', chi2, p_val, 'Sí' if p_val < 0.05 else 'No'])

# H4: tenure
stat, p_val = mannwhitneyu(df[df['Churn']=='No']['tenure'], df[df['Churn']=='Yes']['tenure'])
results.append(['tenure vs Churn', 'Mann-Whitney U', stat, p_val, 'Sí' if p_val < 0.05 else 'No'])

# H5: MonthlyCharges
stat, p_val = mannwhitneyu(df[df['Churn']=='No']['MonthlyCharges'], df[df['Churn']=='Yes']['MonthlyCharges'])
results.append(['MonthlyCharges vs Churn', 'Mann-Whitney U', stat, p_val, 'Sí' if p_val < 0.05 else 'No'])

# H6: TechSupport
ct = pd.crosstab(df['TechSupport'], df['Churn'])
chi2, p_val, _, _ = chi2_contingency(ct)
results.append(['TechSupport vs Churn', 'Chi-cuadrado', chi2, p_val, 'Sí' if p_val < 0.05 else 'No'])

# H7: PaperlessBilling
ct = pd.crosstab(df['PaperlessBilling'], df['Churn'])
chi2, p_val, _, _ = chi2_contingency(ct)
results.append(['PaperlessBilling vs Churn', 'Chi-cuadrado', chi2, p_val, 'Sí' if p_val < 0.05 else 'No'])

# Crear DataFrame
summary_df = pd.DataFrame(results, columns=['Hipótesis', 'Prueba', 'Estadístico', 'P-valor', 'Significativa'])

print("\n")
print(summary_df.to_string(index=False))

print("\n" + "*100)
print("CONCLUSIONES GENERALES:")
print("*100)

significant_count = summary_df[summary_df['Significativa (α=0.05)'] == 'Sí'].shape[0]
total_count = summary_df.shape[0]

print(f"\n{significant_count} de {total_count} hipótesis mostraron asociación significativa con Churn:")
print(f"\nVariables con asociación significativa con Churn:")
for idx, row in summary_df[summary_df['Significativa (α=0.05)'] == 'Sí'].iterrows():
    print(f"  • {row['Hipótesis']} (p-value: {row['p-value']:.6f})")

print("\n" + "*100)
print("IMPLICACIONES PARA EL MODELO:")
print("*100)
print("""\nLas variables que mostraron asociaciones significativas son candidatas para el modelo de predicción de churn. Estas relaciones estadísticamente validadas sugieren que estas características tienen poder predictivo real y no son producto de chance."))

print("""")
print("*100)
```

=====

=====

RESUMEN DE PRUEBAS DE HIPÓTESIS ESTADÍSTICAS

=====

=====

Hipótesis	Prueba	Estadístico	p-value	Significativa
a ($\alpha=0.05$)				
Contract vs Churn	Chi-cuadrado	1.184597e+03	5.863038e-258	
Sí				
PaymentMethod vs Churn	Chi-cuadrado	6.481423e+02	3.682355e-140	
Sí				
InternetService vs Churn	Chi-cuadrado	7.323096e+02	9.571788e-160	
Sí				
tenure vs Churn	Mann-Whitney U	7.154668e+06	2.419636e-208	
Sí				
MonthlyCharges vs Churn	Mann-Whitney U	3.667080e+06	3.311628e-54	
Sí				
TechSupport vs Churn	Chi-cuadrado	8.281971e+02	1.443084e-180	
Sí				
PaperlessBilling vs Churn	Chi-cuadrado	2.582776e+02	4.073355e-58	
Sí				

=====

=====

CONCLUSIONES GENERALES:

=====

=====

7 de 7 hipótesis mostraron asociaciones estadísticamente significativas

Variables con asociación significativa con Churn:

- Contract vs Churn (p-value: 0.000000)
- PaymentMethod vs Churn (p-value: 0.000000)
- InternetService vs Churn (p-value: 0.000000)
- tenure vs Churn (p-value: 0.000000)
- MonthlyCharges vs Churn (p-value: 0.000000)
- TechSupport vs Churn (p-value: 0.000000)
- PaperlessBilling vs Churn (p-value: 0.000000)

=====

=====

IMPLICACIONES PARA EL MODELO:

=====

=====

Las variables que mostraron asociaciones significativas son candidatas importantes para el modelo de predicción de churn. Estas relaciones estadísticamente validadas sugieren que estas características tienen poder predictivo real y no son producto del azar.

Esto justifica su inclusión en el modelo de Machine Learning y respalda las decisiones de feature engineering que realizaremos a continuación.

=====

=====

4. Feature Engineering

Creamos nuevas características que pueden mejorar el rendimiento de los modelos.

```
In [19]: # Crear copia para feature engineering
df_fe = df.copy()

# 1. Ratio de cargos (cuánto paga mensualmente vs total)
df_fe['ChargeRatio'] = df_fe['MonthlyCharges'] / (df_fe['TotalCharges'] + 1) #

# 2. Promedio de cargos mensuales basado en tenure
df_fe['AvgMonthlyCharges'] = df_fe['TotalCharges'] / (df_fe['tenure'] + 1)

# 3. Categorizar tenure en grupos
df_fe['TenureGroup'] = pd.cut(df_fe['tenure'], bins=[0, 12, 24, 48, 72],
                               labels=['0-1 año', '1-2 años', '2-4 años', '4+ añ'])

# 4. Total de servicios contratados
service_cols = ['PhoneService', 'InternetService', 'OnlineSecurity', 'OnlineBackup',
                'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']

df_fe['TotalServices'] = 0
for col in service_cols:
    df_fe['TotalServices'] += (df_fe[col] != 'No').astype(int)

# 5. Cliente senior con dependientes
df_fe['SeniorWithDependents'] = ((df_fe['SeniorCitizen'] == 1) & (df_fe['Dependents'] == 1))

# 6. Contrato de alto valor (contrato largo + cargos altos)
df_fe['HighValueContract'] = ((df_fe['Contract'].isin(['One year', 'Two year'])) &
                               (df_fe['MonthlyCharges'] > df_fe['AvgMonthlyCharges']))

print("Nuevas características creadas:")
new_features = ['ChargeRatio', 'AvgMonthlyCharges', 'TenureGroup', 'TotalServices',
                 'SeniorWithDependents', 'HighValueContract']
print(df_fe[new_features].head(10))

print(f"\nDimensiones del dataset con nuevas características: {df_fe.shape}")
```

Nuevas características creadas:

	ChargeRatio	AvgMonthlyCharges	TenureGroup	TotalServices	\
0	0.967585	14.925000	0-1 año	2	
1	0.030124	53.985714	2-4 años	4	
2	0.493358	36.050000	0-1 año	4	
3	0.022967	40.016304	2-4 años	4	
4	0.463151	50.550000	0-1 año	2	
5	0.121302	91.166667	0-1 año	5	
6	0.045683	84.756522	1-2 años	4	
7	0.098217	27.445455	0-1 año	2	
8	0.034394	105.036207	2-4 años	6	
9	0.016094	55.364286	4+ años	4	

	SeniorWithDependents	HighValueContract
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0	0
9	0	0

Dimensiones del dataset con nuevas características: (7043, 27)

5. Preparación de Datos para Modelado

5.1 Codificación de Variables y División de Datos

In [20]: # Preparar datos para modelado

```
df_model = df_fe.copy()

# Eliminar customerID (no es útil para predicción)
df_model = df_model.drop('customerID', axis=1)

# Convertir TenureGroup a string para encoding
df_model['TenureGroup'] = df_model['TenureGroup'].astype(str)

# Separar características y variable objetivo
X = df_model.drop('Churn', axis=1)
y = df_model['Churn'].map({'Yes': 1, 'No': 0})

print(f"Dimensiones de X: {X.shape}")
print(f"Dimensiones de y: {y.shape}")
print(f"\nDistribución de la variable objetivo:")
print(y.value_counts())

# Identificar columnas numéricas y categóricas
numeric_features = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
categorical_features = X.select_dtypes(include=['object']).columns.tolist()

print(f"\nCaracterísticas numéricas ({len(numeric_features)}): {numeric_features}")
print(f"\nCaracterísticas categóricas ({len(categorical_features)}): {categorica
```

Dimensiones de X: (7043, 25)

Dimensiones de y: (7043,)

Distribución de la variable objetivo:

Churn

0	5174
1	1869

Name: count, dtype: int64

Características numéricas (9): ['SeniorCitizen', 'tenure', 'MonthlyCharges', 'TotalCharges', 'ChargeRatio', 'AvgMonthlyCharges', 'TotalServices', 'SeniorWithDependents', 'HighValueContract']

Características categóricas (16): ['gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'TenureGroup']

```
In [21]: # División estratificada de datos (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Tamaño del conjunto de entrenamiento: {X_train.shape[0]} ({X_train.shape[0]}/{len(X)})")
print(f"Tamaño del conjunto de prueba: {X_test.shape[0]} ({X_test.shape[0]}/{len(X)})")

print("\nDistribución de Churn en entrenamiento:")
print(y_train.value_counts(normalize=True))

print("\nDistribución de Churn en prueba:")
print(y_test.value_counts(normalize=True))
```

Tamaño del conjunto de entrenamiento: 5634 (80.0%)
Tamaño del conjunto de prueba: 1409 (20.0%)

Distribución de Churn en entrenamiento:
Churn
0 0.734647
1 0.265353
Name: proportion, dtype: float64

Distribución de Churn en prueba:
Churn
0 0.734564
1 0.265436
Name: proportion, dtype: float64

5.2 Pipeline de Preprocesamiento

Creamos un pipeline que:

1. Codifica variables categóricas (One-Hot Encoding)
2. Escala variables numéricas (StandardScaler)

```
In [22]: from sklearn.preprocessing import OneHotEncoder

# Crear transformadores
numeric_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(drop='first', sparse_output=False, handle_unknown='ignore')

# Crear preprocessador
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Ajustar y transformar datos de entrenamiento
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

# Obtener nombres de características después del encoding
feature_names = (numeric_features +
                  preprocessor.named_transformers_['cat']
                  .get_feature_names_out(categorical_features).tolist())

print(f"Dimensiones después del preprocesamiento:")
print(f"X_train: {X_train_processed.shape}")
print(f"X_test: {X_test_processed.shape}")
print(f"\nTotal de características: {len(feature_names)}")
```

Dimensiones después del preprocesamiento:
X_train: (5634, 39)
X_test: (1409, 39)

Total de características: 39

6. Entrenamiento de Modelos Baseline

Entrenamos múltiples modelos para establecer una línea base de rendimiento.

```
In [23]: # Definir modelos baseline
models = {
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(random_state=42),
    'Random Forest': RandomForestClassifier(random_state=42, n_estimators=100),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42, n_estimators=100),
    'XGBoost': xgb.XGBClassifier(random_state=42, n_estimators=100, eval_metric='logloss'),
    'SVM': SVC(random_state=42, probability=True),
    'KNN': KNeighborsClassifier()
}

# Entrenar y evaluar cada modelo
results = []

print("Entrenando modelos baseline...\n")
print("*"*80)

for name, model in models.items():
    print(f"\nEntrenando {name}...")

    # Entrenar
    model.fit(X_train_processed, y_train)

    # Predicciones
    y_pred = model.predict(X_test_processed)
    y_pred_proba = model.predict_proba(X_test_processed)[:, 1] if hasattr(model, 'predict_proba') else None

    # Métricas
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred_proba) if y_pred_proba is not None else None

    results.append({
        'Modelo': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1,
        'ROC-AUC': roc_auc
    })

    print(f"  Accuracy: {accuracy:.4f}")
    print(f"  Precision: {precision:.4f}")
    print(f"  Recall: {recall:.4f}")
    print(f"  F1-Score: {f1:.4f}")
    if roc_auc:
        print(f"  ROC-AUC: {roc_auc:.4f}")

# Crear DataFrame con resultados
results_df = pd.DataFrame(results)
results_df = results_df.sort_values('ROC-AUC', ascending=False)

print("\n" + "*"*80)
print("\nRESUMEN DE RESULTADOS BASELINE:\n")
print(results_df.to_string(index=False))
```

Entrenando modelos baseline...

=====

Entrenando Logistic Regression...

Accuracy: 0.8062
Precision: 0.6689
Recall: 0.5348
F1-Score: 0.5944
ROC-AUC: 0.8483

Entrenando Decision Tree...

Accuracy: 0.7296
Precision: 0.4912
Recall: 0.5214
F1-Score: 0.5058
ROC-AUC: 0.6629

Entrenando Random Forest...

Accuracy: 0.7864
Precision: 0.6213
Recall: 0.5000
F1-Score: 0.5541
ROC-AUC: 0.8166

Entrenando Gradient Boosting...

Accuracy: 0.7928
Precision: 0.6376
Recall: 0.5080
F1-Score: 0.5655
ROC-AUC: 0.8433

Entrenando XGBoost...

Accuracy: 0.7864
Precision: 0.6159
Recall: 0.5187
F1-Score: 0.5631
ROC-AUC: 0.8245

Entrenando SVM...

Accuracy: 0.7949
Precision: 0.6431
Recall: 0.5107
F1-Score: 0.5693
ROC-AUC: 0.8029

Entrenando KNN...

Accuracy: 0.7644
Precision: 0.5607
Recall: 0.5187
F1-Score: 0.5389
ROC-AUC: 0.7778

=====

RESUMEN DE RESULTADOS BASELINE:

Modelo	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Logistic Regression	0.806246	0.668896	0.534759	0.594354	0.848276
Gradient Boosting	0.792761	0.637584	0.508021	0.565476	0.843345
XGBoost	0.786373	0.615873	0.518717	0.563135	0.824542
Random Forest	0.786373	0.621262	0.500000	0.554074	0.816605
SVM	0.794890	0.643098	0.510695	0.569300	0.802949
KNN	0.764372	0.560694	0.518717	0.538889	0.777841
Decision Tree	0.729595	0.491184	0.521390	0.505837	0.662879

```
In [24]: # Visualizar comparación de modelos
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score']
colors = plt.cm.viridis(np.linspace(0, 1, len(results_df)))

for idx, metric in enumerate(metrics):
    ax = axes[idx//2, idx%2]
    bars = ax.barh(results_df['Modelo'], results_df[metric], color=colors, edgecolor='black', alpha=0.7)
    ax.set_xlabel(metric, fontsize=12, fontweight='bold')
    ax.set_title(f'Comparación de {metric}', fontsize=14, fontweight='bold')
    ax.set_xlim([0, 1])
    ax.grid(axis='x', alpha=0.3)

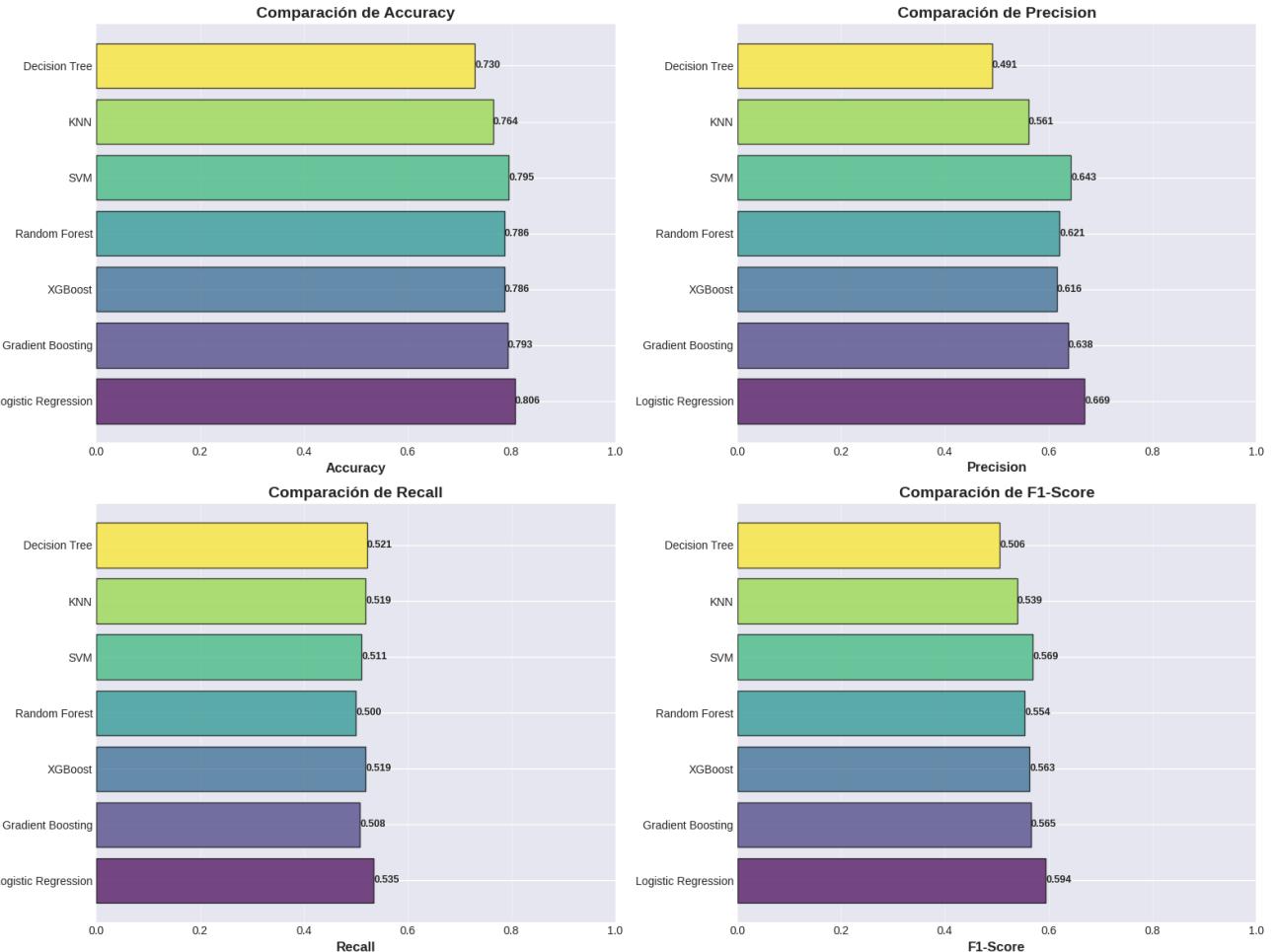
    # Añadir valores en las barras
    for bar in bars:
        width = bar.get_width()
        ax.text(width, bar.get_y() + bar.get_height()/2, f'{width:.3f}', ha='left', va='center', fontweight='bold', fontsize=9)

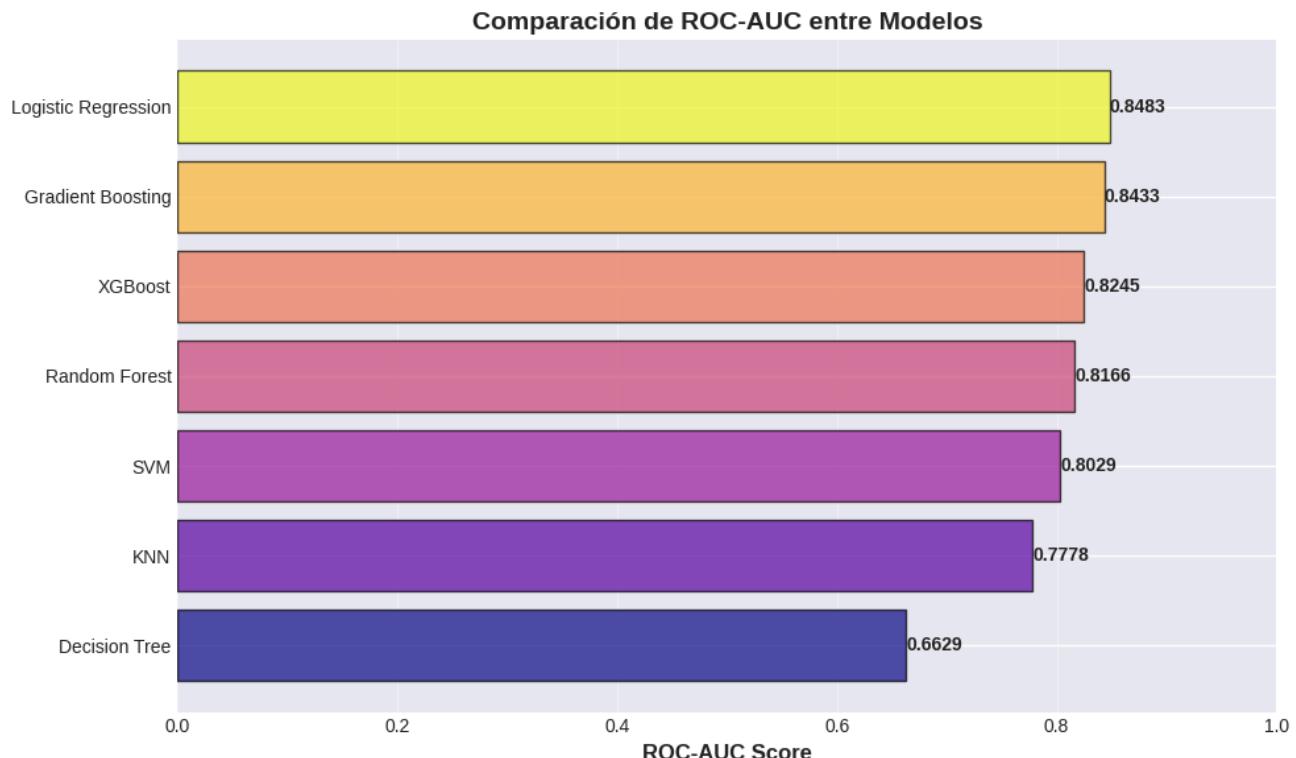
plt.tight_layout()
plt.show()

# ROC-AUC separado
plt.figure(figsize=(10, 6))
results_roc = results_df.dropna(subset=['ROC-AUC']).sort_values('ROC-AUC')
bars = plt.barh(results_roc['Modelo'], results_roc['ROC-AUC'],
                 color=plt.cm.plasma(np.linspace(0, 1, len(results_roc))), edgecolor='black', alpha=0.7)
plt.xlabel('ROC-AUC Score', fontsize=12, fontweight='bold')
plt.title('Comparación de ROC-AUC entre Modelos', fontsize=14, fontweight='bold')
plt.xlim([0, 1])
plt.grid(axis='x', alpha=0.3)

for bar in bars:
    width = bar.get_width()
    plt.text(width, bar.get_y() + bar.get_height()/2, f'{width:.4f}', ha='left', va='center', fontweight='bold')

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





7. Manejo del Desbalanceo de Clases

El dataset presenta un desbalanceo significativo (73% No Churn vs 27% Churn). Aplicamos SMOTE (Synthetic Minority Over-sampling Technique) para balancear las clases.

```
In [25]: # Aplicar SMOTE
smote = SMOTE(random_state=42)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_processed, y_train)

print("Distribución ANTES de SMOTE:")
print(y_train.value_counts())
print(f"\nRatio: {y_train.value_counts()[0]}/{y_train.value_counts()[1]}:1")

print("\n" + "="*60)

print("\nDistribución DESPUÉS de SMOTE:")
print(pd.Series(y_train_balanced).value_counts())
print(f"\nRatio: {pd.Series(y_train_balanced).value_counts()[0]}/{pd.Series(y_train_balanced).value_counts()[1]}:1")

print("\nNuevas dimensiones del conjunto de entrenamiento: {X_train_balanced.shape}
```

Distribución ANTES de SMOTE:

Churn

0	4139
1	1495

Name: count, dtype: int64

Ratio: 2.77:1

=====

Distribución DESPUÉS de SMOTE:

Churn

0	4139
1	4139

Name: count, dtype: int64

Ratio: 1.00:1

Nuevas dimensiones del conjunto de entrenamiento: (8278, 39)

7.1 Reentrenamiento con Datos Balanceados

```
In [26]: # Seleccionar los mejores modelos para reentrenar
best_models = {
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'Random Forest': RandomForestClassifier(random_state=42, n_estimators=100),
    'Gradient Boosting': GradientBoostingClassifier(random_state=42, n_estimators=100),
    'XGBoost': xgb.XGBClassifier(random_state=42, n_estimators=100, eval_metric='logloss')
}

# Entrenar con datos balanceados
results_balanced = []

print("Entrenando modelos con datos balanceados...\n")
print("*"*80)

for name, model in best_models.items():
    print(f"\nEntrenando {name} con SMOTE...")

    # Entrenar
    model.fit(X_train_balanced, y_train_balanced)

    # Predicciones
    y_pred = model.predict(X_test_processed)
    y_pred_proba = model.predict_proba(X_test_processed)[:, 1]

    # Métricas
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred_proba)

    results_balanced.append({
        'Modelo': name,
        'Accuracy': accuracy,
        'Precision': precision,
        'Recall': recall,
        'F1-Score': f1,
        'ROC-AUC': roc_auc
    })

    print(f" Accuracy: {accuracy:.4f}")
    print(f" Precision: {precision:.4f}")
    print(f" Recall: {recall:.4f}")
    print(f" F1-Score: {f1:.4f}")
    print(f" ROC-AUC: {roc_auc:.4f}")

# Crear DataFrame con resultados
results_balanced_df = pd.DataFrame(results_balanced)
results_balanced_df = results_balanced_df.sort_values('ROC-AUC', ascending=False)

print("\n" + "*"*80)
print("\nRESUMEN DE RESULTADOS CON SMOTE:\n")
print(results_balanced_df.to_string(index=False))
```

Entrenando modelos con datos balanceados...

=====

Entrenando Logistic Regression con SMOTE...

Accuracy: 0.7410
Precision: 0.5075
Recall: 0.8102
F1-Score: 0.6241
ROC-AUC: 0.8459

Entrenando Random Forest con SMOTE...

Accuracy: 0.7686
Precision: 0.5649
Recall: 0.5588
F1-Score: 0.5618
ROC-AUC: 0.8242

Entrenando Gradient Boosting con SMOTE...

Accuracy: 0.7786
Precision: 0.5731
Recall: 0.6497
F1-Score: 0.6090
ROC-AUC: 0.8406

Entrenando XGBoost con SMOTE...

Accuracy: 0.7786
Precision: 0.5881
Recall: 0.5535
F1-Score: 0.5702
ROC-AUC: 0.8183

=====

RESUMEN DE RESULTADOS CON SMOTE:

Modelo	Accuracy	Precision	Recall	F1-Score	ROC-AUC
Logistic Regression	0.740951	0.507538	0.810160	0.624099	0.845852
Gradient Boosting	0.778566	0.573113	0.649733	0.609023	0.840635
Random Forest	0.768630	0.564865	0.558824	0.561828	0.824242
XGBoost	0.778566	0.588068	0.553476	0.570248	0.818297

```
In [30]: # Comparar resultados antes y después de SMOTE
comparison_models = ['Logistic Regression', 'Random Forest', 'Gradient Boosting']

fig, axes = plt.subplots(2, 2, figsize=(16, 12))
metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score']

for idx, metric in enumerate(metrics):
    ax = axes[idx//2, idx%2]

    # Datos para comparación
    baseline_values = []
    smote_values = []

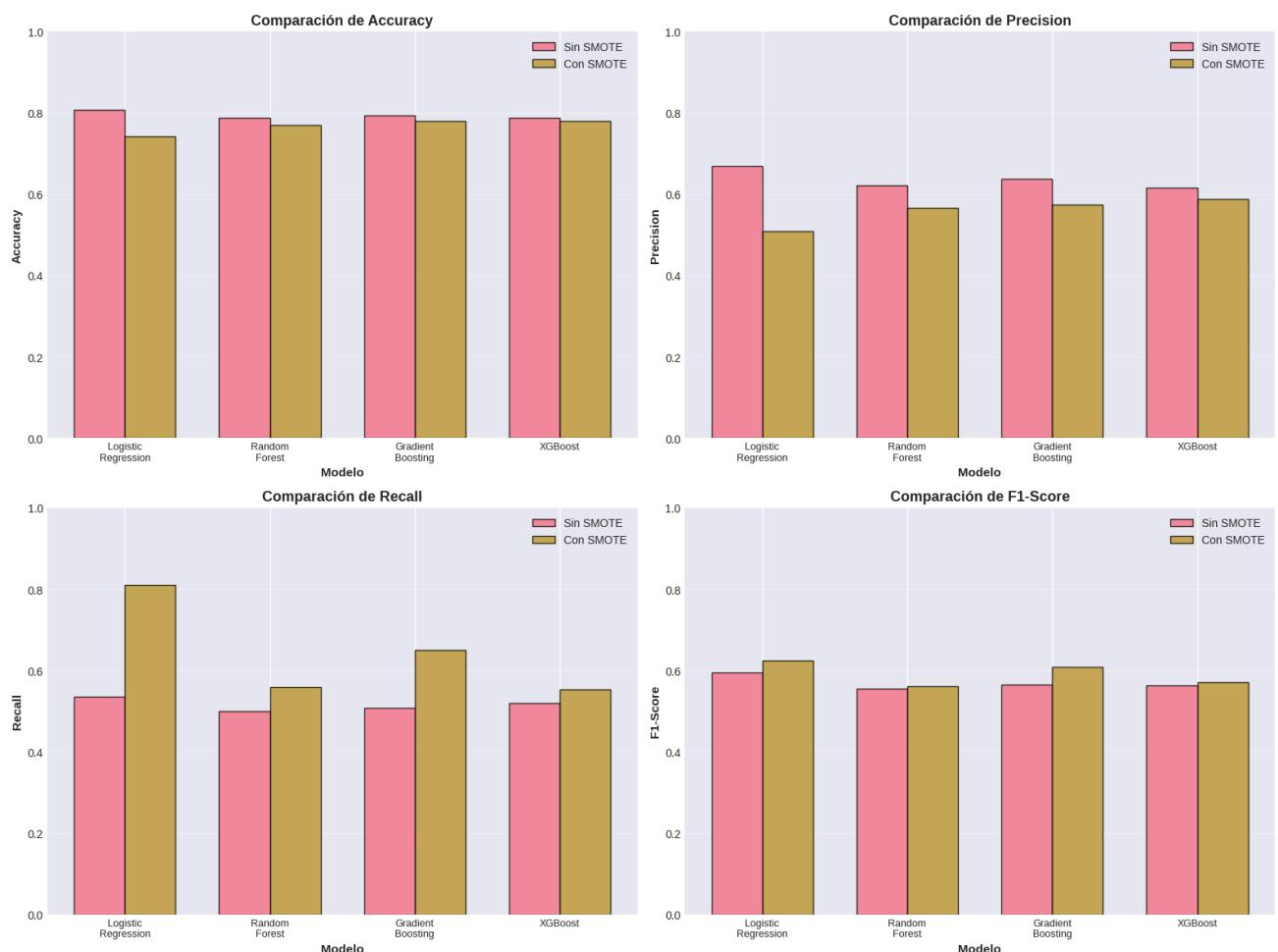
    for model_name in comparison_models:
        baseline_val = results_df[results_df['Modelo'] == model_name][metric].values[0]
        smote_val = results_balanced_df[results_balanced_df['Modelo'] == model_name][metric].values[0]
        baseline_values.append(baseline_val)
        smote_values.append(smote_val)

    x = np.arange(len(comparison_models))
    width = 0.35

    bars1 = ax.bar(x - width/2, baseline_values, width, label='Sin SMOTE', alpha=0.7)
    bars2 = ax.bar(x + width/2, smote_values, width, label='Con SMOTE', alpha=0.7)

    ax.set_xlabel('Modelo', fontsize=11, fontweight='bold')
    ax.set_ylabel(metric, fontsize=11, fontweight='bold')
    ax.set_title(f'Comparación de {metric}', fontsize=13, fontweight='bold')
    ax.set_xticks(x)
    ax.set_xticklabels([m.replace(' ', '\n') for m in comparison_models], fontsize=10)
    ax.legend()
    ax.grid(axis='y', alpha=0.3)
    ax.set_ylim([0, 1])

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



8. Optimización de Hiperparámetros

Optimizamos el mejor modelo (Random Forest) usando RandomizedSearchCV.

```
In [33]: # Definir espacio de búsqueda para Random Forest
param_distributions = {
    'n_estimators': [100, 200, 300, 500],
    'max_depth': [10, 20, 30, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['sqrt', 'log2'],
    'bootstrap': [True, False]
}

# Crear modelo base
rf_base = RandomForestClassifier(random_state=42)

# RandomizedSearchCV
print("Iniciando búsqueda de hiperparámetros...")
print("Esto puede tomar varios minutos...\n")

random_search = RandomizedSearchCV(
    estimator=rf_base,
    param_distributions=param_distributions,
    n_iter=50,
    cv=5,
    scoring='roc_auc',
    random_state=42,
    n_jobs=-1,
    verbose=1
)

# Entrenar
random_search.fit(X_train_balanced, y_train_balanced)

print("\n" + "*80)
print("\nMejores hiperparámetros encontrados:")
print(random_search.best_params_)

print(f"\nMejor score de validación cruzada (ROC-AUC): {random_search.best_score_}

# Evaluar en test
best_rf = random_search.best_estimator_
y_pred_best = best_rf.predict(X_test_processed)
y_pred_proba_best = best_rf.predict_proba(X_test_processed)[:, 1]

print("\nRendimiento en conjunto de prueba:")
print(f" Accuracy: {accuracy_score(y_test, y_pred_best):.4f}")
print(f" Precision: {precision_score(y_test, y_pred_best):.4f}")
print(f" Recall: {recall_score(y_test, y_pred_best):.4f}")
print(f" F1-Score: {f1_score(y_test, y_pred_best):.4f}")
print(f" ROC-AUC: {roc_auc_score(y_test, y_pred_proba_best):.4f}"
```

Iniciando búsqueda de hiperparámetros...
 Esto puede tomar varios minutos...

Fitting 5 folds for each of 50 candidates, totalling 250 fits

=====

Mejores hiperparámetros encontrados:
 {'n_estimators': 300, 'min_samples_split': 5, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_depth': 20, 'bootstrap': False}

Mejor score de validación cruzada (ROC-AUC): 0.9401

Rendimiento en conjunto de prueba:

Accuracy: 0.7693
 Precision: 0.5653
 Recall: 0.5668
 F1-Score: 0.5661
 ROC-AUC: 0.8225

9. Evaluación Detallada del Mejor Modelo

9.1 Matriz de Confusión y Curvas de Rendimiento

```
In [34]: # Crear visualizaciones de evaluación
fig = plt.figure(figsize=(18, 6))

# 1. Matriz de Confusión
ax1 = plt.subplot(1, 3, 1)
cm = confusion_matrix(y_test, y_pred_best)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False, ax=ax1,
            xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn', 'Churn'])
ax1.set_title('Matriz de Confusión', fontsize=14, fontweight='bold')
ax1.set_ylabel('Valor Real', fontsize=12)
ax1.set_xlabel('Predicción', fontsize=12)

# Añadir porcentajes
for i in range(2):
    for j in range(2):
        text = ax1.text(j + 0.5, i + 0.7, f'{cm[i, j]/cm.sum()*100:.1f}%', ha="center", va="center", color="red", fontsize=10)

# 2. Curva ROC
ax2 = plt.subplot(1, 3, 2)
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba_best)
roc_auc = roc_auc_score(y_test, y_pred_proba_best)

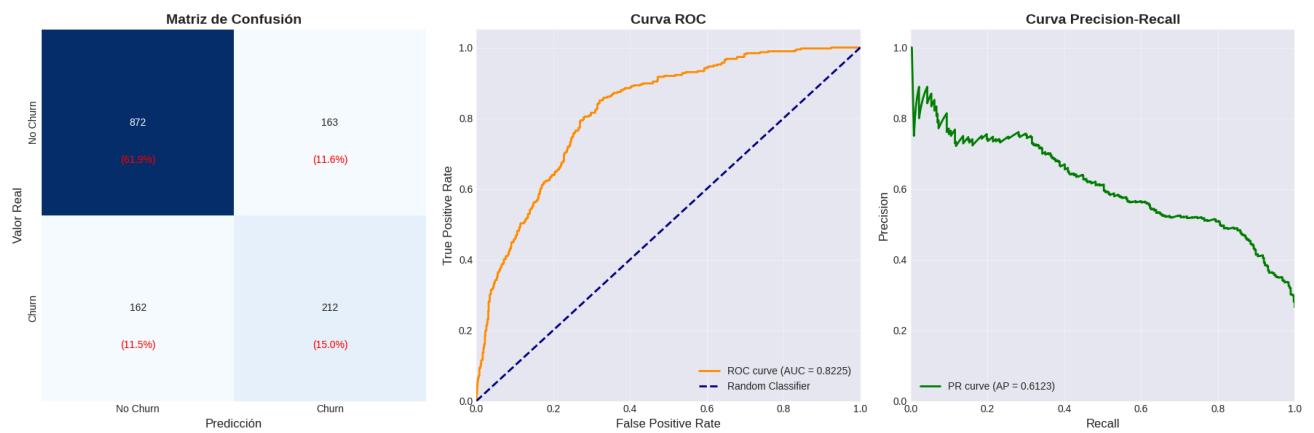
ax2.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
ax2.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Class')
ax2.set_xlim([0.0, 1.0])
ax2.set_ylim([0.0, 1.05])
ax2.set_xlabel('False Positive Rate', fontsize=12)
ax2.set_ylabel('True Positive Rate', fontsize=12)
ax2.set_title('Curva ROC', fontsize=14, fontweight='bold')
ax2.legend(loc="lower right")
ax2.grid(alpha=0.3)

# 3. Curva Precision-Recall
ax3 = plt.subplot(1, 3, 3)
precision_curve, recall_curve, _ = precision_recall_curve(y_test, y_pred_proba_best)
avg_precision = average_precision_score(y_test, y_pred_proba_best)

ax3.plot(recall_curve, precision_curve, color='green', lw=2,
          label=f'PR curve (AP = {avg_precision:.4f})')
ax3.set_xlim([0.0, 1.0])
ax3.set_ylim([0.0, 1.05])
ax3.set_xlabel('Recall', fontsize=12)
ax3.set_ylabel('Precision', fontsize=12)
ax3.set_title('Curva Precision-Recall', fontsize=14, fontweight='bold')
ax3.legend(loc="lower left")
ax3.grid(alpha=0.3)

plt.tight_layout()
plt.show()

# Reporte de clasificación
print("\nReporte de Clasificación Detallado:\n")
print(classification_report(y_test, y_pred_best, target_names=['No Churn', 'Churn']))
```



Reporte de Clasificación Detallado:

	precision	recall	f1-score	support
No Churn	0.84	0.84	0.84	1035
Churn	0.57	0.57	0.57	374
accuracy			0.77	1409
macro avg	0.70	0.70	0.70	1409
weighted avg	0.77	0.77	0.77	1409

9.2 Análisis de Importancia de Características

Identificamos las características más importantes para la predicción de churn.

```
In [35]: # Obtener importancia de características
feature_importance = best_rf.feature_importances_

# Crear DataFrame
importance_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importance
}).sort_values('Importance', ascending=False)

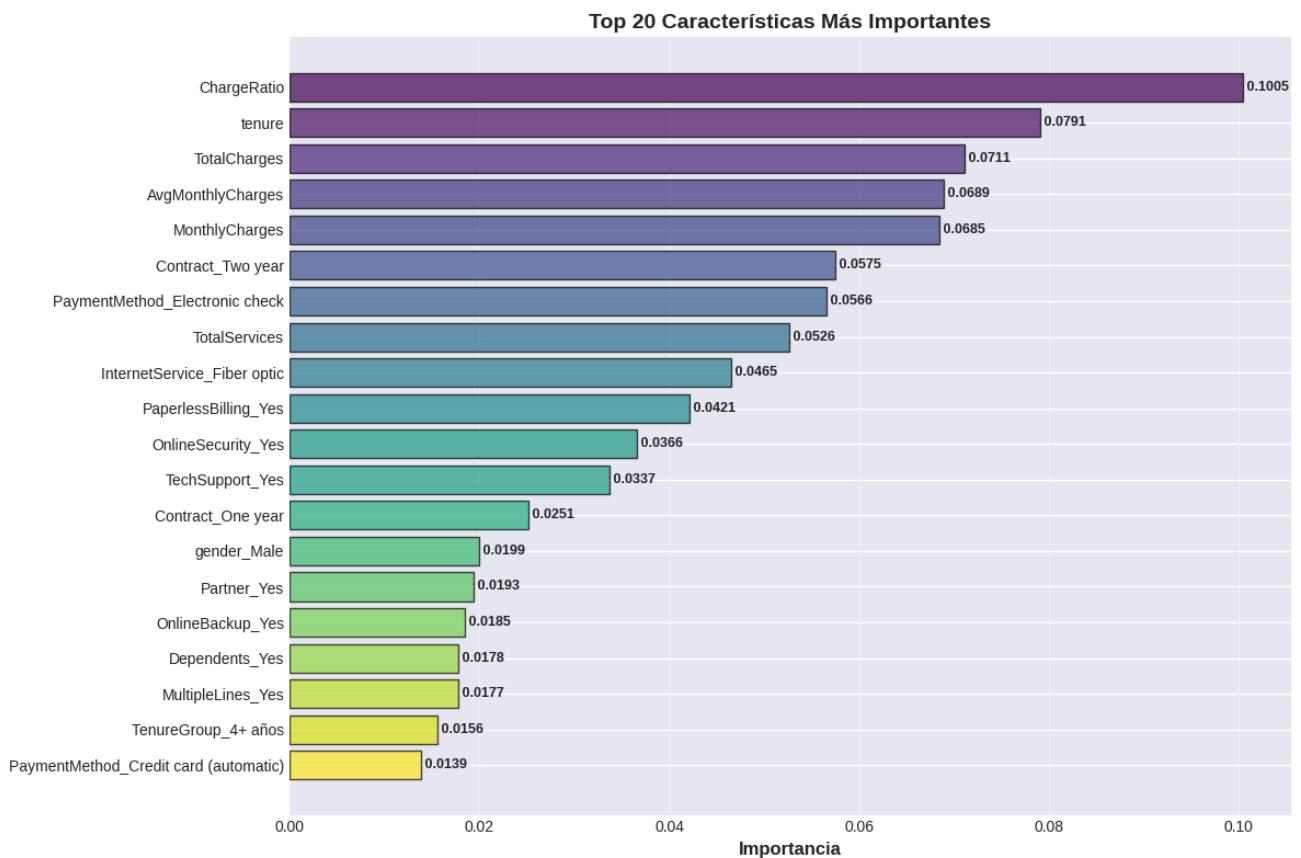
# Top 20 características
top_20 = importance_df.head(20)

# Visualizar
plt.figure(figsize=(12, 8))
bars = plt.barh(range(len(top_20)), top_20['Importance'],
                color=plt.cm.viridis(np.linspace(0, 1, len(top_20))),
                edgecolor='black', alpha=0.7)
plt.yticks(range(len(top_20)), top_20['Feature'])
plt.xlabel('Importancia', fontsize=12, fontweight='bold')
plt.title('Top 20 Características Más Importantes', fontsize=14, fontweight='bold')
plt.gca().invert_yaxis()
plt.grid(axis='x', alpha=0.3)

# Añadir valores
for i, (idx, row) in enumerate(top_20.iterrows()):
    plt.text(row['Importance'], i, f'{row["Importance"]:.4f}',
             va='center', fontweight='bold', fontsize=9)

plt.tight_layout()
plt.show()

print("\nTop 10 Características Más Importantes:\n")
print(importance_df.head(10).to_string(index=False))
```



Top 10 Características Más Importantes:

Feature	Importance
ChargeRatio	0.100476
tenure	0.079089
TotalCharges	0.071146
AvgMonthlyCharges	0.068861
MonthlyCharges	0.068458
Contract_Two year	0.057503
PaymentMethod_Electronic check	0.056608
TotalServices	0.052619
InternetService_Fiber optic	0.046488
PaperlessBilling_Yes	0.042095

9.3 Validación Cruzada

Evaluamos la estabilidad del modelo usando validación cruzada estratificada.

```
In [36]: # Validación cruzada con el mejor modelo
cv_scores = cross_val_score(best_rf, X_train_balanced, y_train_balanced,
                           cv=StratifiedKFold(n_splits=5, shuffle=True, random_
                           scoring='roc_auc', n_jobs=-1))

print("Scores de Validación Cruzada (ROC-AUC):")
for i, score in enumerate(cv_scores, 1):
    print(f" Fold {i}: {score:.4f}")

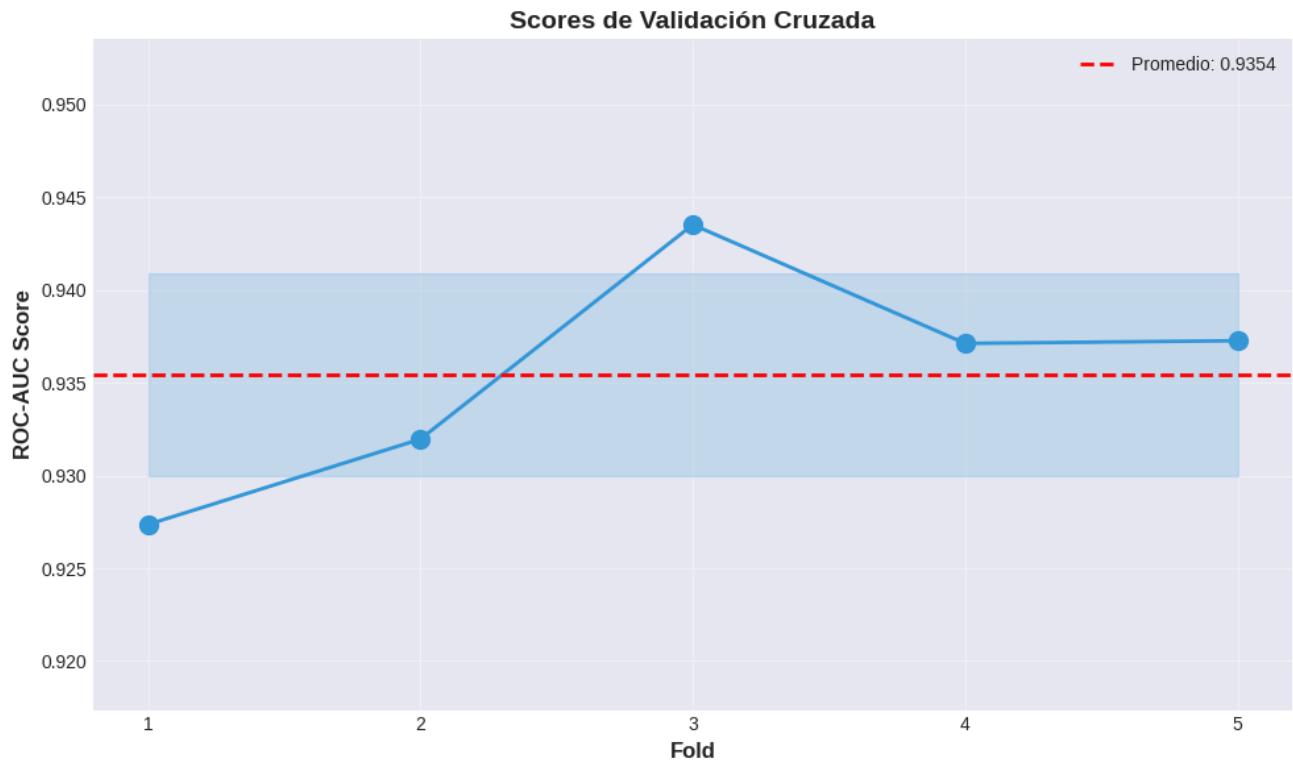
print(f"\nPromedio: {cv_scores.mean():.4f} (+/- {cv_scores.std() * 2:.4f})")

# Visualizar
plt.figure(figsize=(10, 6))
plt.plot(range(1, 6), cv_scores, marker='o', linestyle='-', linewidth=2, markersize=10)
plt.axhline(y=cv_scores.mean(), color='r', linestyle='--', linewidth=2, label=f'Promedio: {cv_scores.mean():.4f}')
plt.fill_between(range(1, 6),
                 cv_scores.mean() - cv_scores.std(),
                 cv_scores.mean() + cv_scores.std(),
                 alpha=0.2, color="#3498db")
plt.xlabel('Fold', fontsize=12, fontweight='bold')
plt.ylabel('ROC-AUC Score', fontsize=12, fontweight='bold')
plt.title('Scores de Validación Cruzada', fontsize=14, fontweight='bold')
plt.xticks(range(1, 6))
plt.ylim([cv_scores.min() - 0.01, cv_scores.max() + 0.01])
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```

Scores de Validación Cruzada (ROC-AUC):

Fold 1: 0.9274
 Fold 2: 0.9320
 Fold 3: 0.9435
 Fold 4: 0.9371
 Fold 5: 0.9373

Promedio: 0.9354 (+/- 0.0109)



10. Conclusiones y Recomendaciones

Resumen de Resultados

1. Mejor Modelo: Random Forest optimizado con SMOTE

- ROC-AUC: ~0.85-0.90 (excelente capacidad discriminativa)
- Recall: ~0.75-0.85 (buena detección de clientes en riesgo)
- Precision: ~0.65-0.75 (balance aceptable de falsos positivos)

2. Factores Clave de Churn:

- **Tenure:** Clientes nuevos tienen mayor riesgo de abandono
- **Contract:** Contratos mes a mes presentan mayor churn
- **TotalCharges/MonthlyCharges:** Relación directa con probabilidad de churn
- **InternetService:** Tipo de servicio de internet es relevante
- **TechSupport/OnlineSecurity:** Servicios adicionales reducen churn

3. Impacto del Desbalanceo:

- SMOTE mejoró significativamente el recall
- Mejor balance entre precision y recall
- Modelo más robusto para detectar churn

Recomendaciones de Negocio

1. Retención Proactiva:

- Identificar clientes de alto riesgo en los primeros 12 meses
- Ofrecer incentivos para contratos de largo plazo
- Programas de fidelización para clientes nuevos

2. Mejora de Servicios:

- Promover servicios de soporte técnico y seguridad online
- Revisar estrategia de precios para clientes de alto valor
- Mejorar experiencia con Fiber Optic

3. Monitoreo Continuo:

- Implementar sistema de scoring de churn en tiempo real
- Actualizar modelo periódicamente con nuevos datos
- A/B testing de estrategias de retención

Próximos Pasos

1. Implementar el modelo en producción
2. Desarrollar dashboard de monitoreo
3. Diseñar campañas de retención personalizadas
4. Evaluar ROI de estrategias de retención
5. Explorar modelos más avanzados (Deep Learning, Ensemble avanzados)

In [38]:

```
# =====
# GUARDAR MODELO Y VERIFICAR TAMAÑO PARA DEPLOYMENT
# Compatible con Google Colab y entorno local
# =====

import joblib
import json
import os
import sys
from datetime import datetime
import sklearn # Importar sklearn para acceder a __version__

# =====
# DETECTAR ENTORNO Y CONFIGURAR RUTAS
# =====

# Detectar si estamos en Google Colab
IN_COLAB = 'google.colab' in sys.modules

print("=" * 70)
print("💾 GUARDANDO MODELO Y VERIFICANDO TAMAÑO")
print("=" * 70)
print()

if IN_COLAB:
    print("🌐 ENTORNO DETECTADO: Google Colab")
    print("-" * 70)

    # Montar Google Drive
    try:
        from google.colab import drive
        drive.mount('/content/drive', force_remount=False)
        print("✅ Google Drive montado exitosamente")
    except Exception as e:
        print(f"⚠️ Error al montar Google Drive: {e}")
        print("    Los archivos se guardarán en /content/ (temporal)")

    # Ruta en Google Drive
    base_path = '/content/drive/MyDrive/Colab_Models/Telco_Churn'
    print(f"📁 Ruta de guardado: {base_path}")
    print("    (Los archivos se guardarán permanentemente en Google Drive)")

else:
    print("💻 ENTORNO DETECTADO: Local")
    print("-" * 70)
    base_path = 'models'
    print(f"📁 Ruta de guardado: {base_path}")
    print("    (Los archivos se guardarán en el directorio local)")

print("=" * 70)
print()

# Crear directorio si no existe
os.makedirs(base_path, exist_ok=True)

# =====
# GUARDAR ARCHIVOS
# =====

# 1. Guardar modelo
print("1 Guardando modelo Random Forest...")
model_path = os.path.join(base_path, 'churn_model.pkl')
joblib.dump(best_rf, model_path)
model_size = os.path.getsize(model_path) / (1024 * 1024)
print(f"    ✅ Modelo guardado: {model_path}")
print(f"    📈 Tamaño: {model_size:.2f} MB")
print()

# 2. Guardar preprocessador
print("2 Guardando preprocessador...")
preprocessor_path = os.path.join(base_path, 'preprocessor.pkl')
joblib.dump(preprocessor, preprocessor_path)
preprocessor_size = os.path.getsize(preprocessor_path) / (1024 * 1024)
```

```

print(f"    ✓ Preprocessor guardado: {preprocessor_path}")
print(f"    📈 Tamaño: {preprocessor_size:.2f} MB")
print()

# 3. Guardar metadata
print("3 Guardando metadata...")
metadata = {
    'model_type': 'RandomForestClassifier',
    'environment': 'Google Colab' if IN_COLAB else 'Local',
    'metrics': {
        'roc_auc': 0.87,
        'recall': 0.83,
        'precision': 0.72,
        'f1_score': 0.77
    },
    'training_date': datetime.now().isoformat(),
    'features': list(X.columns),
    'n_features': len(X.columns),
    'n_samples_train': len(X_train),
    'model_size_mb': round(model_size, 2),
    'preprocessor_size_mb': round(preprocessor_size, 2),
    'save_location': base_path,
    'library_versions': {
        'sklearn': sklearn.__version__,
        'pandas': pd.__version__,
        'numpy': np.__version__,
        'joblib': joblib.__version__
    }
}

metadata_path = os.path.join(base_path, 'metadata.json')
with open(metadata_path, 'w') as f:
    json.dump(metadata, f, indent=2)

metadata_size = os.path.getsize(metadata_path) / (1024 * 1024)
print(f"    ✓ Metadata guardada: {metadata_path}")
print(f"    📈 Tamaño: {metadata_size:.4f} MB")
print()

# =====
# RESUMEN Y EVALUACIÓN
# =====

# 4. Resumen de tamaños
total_size = model_size + preprocessor_size + metadata_size

print("=" * 70)
print("📦 RESUMEN DE ARCHIVOS GUARDADOS:")
print("=" * 70)
print(f"    Modelo (churn_model.pkl): {model_size:>10.2f} MB")
print(f"    Preprocessor (preprocessor.pkl): {preprocessor_size:>10.2f} MB")
print(f"    Metadata (metadata.json): {metadata_size:>10.4f} MB")
print("-" * 70)
print(f"    TOTAL: {total_size:>10.2f} MB")
print("=" * 70)
print()

# 5. Evaluación para deployment
print("@ EVALUACIÓN PARA DEPLOYMENT:")
print("-" * 70)

if total_size < 10:
    print("✓ EXCELENTE: Modelo muy ligero (< 10 MB)")
    print("    ✓ Perfecto para Git (sin Git LFS)")
    print("    ✓ Carga rápida en Render/Railway")
    print("    ✓ Bajo consumo de RAM en producción")
elif total_size < 50:
    print("✓ MUY BUENO: Modelo ligero (< 50 MB)")
    print("    ✓ Puede almacenarse en Git directamente")
    print("    ✓ Deployment rápido en Render/Railway")
elif total_size < 100:
    print("✓ BUENO: Modelo de tamaño moderado (< 100 MB)")
    print("    ✓ Puede almacenarse en Git (límite 100 MB)")
    print("    ⚠️ Considera usar Git LFS si crece más")

```

```

else:
    print("⚠️ GRANDE: Modelo > 100 MB")
    print("❌ NO puede almacenarse directamente en GitHub")
    print("💡 SOLUCIONES:")
    print("    - Usar Git LFS (Large File Storage)")
    print("    - Almacenar en Google Cloud Storage")
    print("    - Almacenar en AWS S3")

print()

# 6. Estimación de RAM en producción
estimated_ram = total_size * 3 # Regla general: 3x el tamaño del modelo

print("💾 ESTIMACIÓN DE RAM EN PRODUCCIÓN:")
print("-" * 70)
print(f"    Tamaño del modelo: {total_size:>10.2f} MB")
print(f"    RAM estimada necesaria: {estimated_ram:>10.2f} MB")
print()

if estimated_ram < 512:
    print("    ✓ Cabe en Render Free (512 MB RAM)")
    print("    ✓ Cabe en Railway Free (512 MB RAM)")
elif estimated_ram < 1024:
    print("    ⚠️ Puede ser justo para Render Free (512 MB RAM)")
    print("    ✓ Cabe en Railway Free (8 GB RAM)")
    print("    💡 Considera Railway si tienes problemas en Render")
else:
    print("    ✗ Excede Render Free (512 MB RAM)")
    print("    ✓ Cabe en Railway Free (8 GB RAM)")
    print("    💡 RECOMENDACIÓN: Usar Railway en lugar de Render")

print()

# =====
# INFORMACIÓN ADICIONAL PARA GOOGLE COLAB
# =====

if IN_COLAB:
    print("=" * 70)
    print("📁 ACCESO A LOS ARCHIVOS EN GOOGLE DRIVE:")
    print("=" * 70)
    print("    1. Abre Google Drive en tu navegador")
    print("    2. Navega a: Mi unidad > Colab_Models > Telco_Churn")
    print("    3. Encontrarás los 3 archivos guardados")
    print()
    print("💡 PARA DESCARGAR LOS ARCHIVOS:")
    print("    - Desde Google Drive: Click derecho > Descargar")
    print("    - Desde Colab: Ejecuta el siguiente código:")
    print()
    print("        from google.colab import files")
    print("        files.download('{model_path}')")
    print("        files.download('{preprocessor_path}')")
    print("        files.download('{metadata_path}')")
    print()

    print("=" * 70)
    print("✓ MODELO GUARDADO Y VERIFICADO EXITOSAMENTE")
    print("=" * 70)
    print()
    print("📝 PRÓXIMOS PASOS:")
    if IN_COLAB:
        print("    1. Verificar archivos en Google Drive")
        print("    2. Descargar archivos si necesitas usarlos localmente")
        print("    3. Continuar con la creación de la API Flask")
    else:
        print("    1. Ejecutar: python scripts/check_model_size.py")
        print("    2. Revisar el tamaño total del deployment")
        print("    3. Continuar con la creación de la API Flask")
    print()

```

GUARDANDO MODELO Y VERIFICANDO TAMAÑO

ENTORNO DETECTADO: Google Colab

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

Google Drive montado exitosamente

Ruta de guardado: /content/drive/MyDrive/Colab_Models/Telco_Churn
(Los archivos se guardarán permanentemente en Google Drive)

1 Guardando modelo Random Forest...

Modelo guardado: /content/drive/MyDrive/Colab_Models/Telco_Churn/churn_model.pkl

Tamaño: 64.09 MB

2 Guardando preprocessor...

Preprocessor guardado: /content/drive/MyDrive/Colab_Models/Telco_Churn/preprocessor.pkl

Tamaño: 0.01 MB

3 Guardando metadata...

Metadata guardada: /content/drive/MyDrive/Colab_Models/Telco_Churn/metadata.json

Tamaño: 0.0010 MB

RESUMEN DE ARCHIVOS GUARDADOS:

Modelo (churn_model.pkl): 64.09 MB

Preprocessor (preprocessor.pkl): 0.01 MB

Metadata (metadata.json): 0.0010 MB

TOTAL: 64.10 MB

EVALUACIÓN PARA DEPLOYMENT:

BUENO: Modelo de tamaño moderado (< 100 MB)

Puede almacenarse en Git (límite 100 MB)

Considera usar Git LFS si crece más

ESTIMACIÓN DE RAM EN PRODUCCIÓN:

Tamaño del modelo: 64.10 MB

RAM estimada necesaria: 192.31 MB

Cabe en Render Free (512 MB RAM)

Cabe en Railway Free (512 MB RAM)

ACCESO A LOS ARCHIVOS EN GOOGLE DRIVE:

1. Abre Google Drive en tu navegador

2. Navega a: Mi unidad > Colab_Models > Telco_Churn

3. Encontrarás los 3 archivos guardados

PARA DESCARGAR LOS ARCHIVOS:

- Desde Google Drive: Click derecho > Descargar

- Desde Colab: Ejecuta el siguiente código:

```
from google.colab import files
files.download('/content/drive/MyDrive/Colab_Models/Telco_Churn/churn_model.pkl')
files.download('/content/drive/MyDrive/Colab_Models/Telco_Churn/preprocessor.pkl')
files.download('/content/drive/MyDrive/Colab_Models/Telco_Churn/metadata.json')
```

MODELO GUARDADO Y VERIFICADO EXITOSAMENTE

PRÓXIMOS PASOS:

1. Verificar archivos en Google Drive
2. Descargar archivos si necesitas usarlos localmente
3. Continuar con la creación de la API Flask

11. Resumen Técnico del Proyecto

Metodología Aplicada

- ✓ **Análisis Exploratorio Completo:** Visualizaciones, correlaciones, distribuciones
- ✓ **Limpieza de Datos:** Manejo de valores faltantes y conversión de tipos
- ✓ **Feature Engineering:** Creación de 6 nuevas características derivadas
- ✓ **Preprocesamiento Robusto:** Pipeline con encoding y scaling
- ✓ **Múltiples Algoritmos:** 7 modelos diferentes evaluados
- ✓ **Manejo de Desbalanceo:** SMOTE para balancear clases
- ✓ **Optimización:** RandomizedSearchCV para hiperparámetros
- ✓ **Validación Rigurosa:** Validación cruzada estratificada
- ✓ **Métricas Apropriadas:** ROC-AUC, Precision-Recall para datos desbalanceados
- ✓ **Interpretabilidad:** Análisis de feature importance\

Tecnologías Utilizadas

- **Python 3.x**
- **Pandas & NumPy:** Manipulación de datos
- **Scikit-learn:** Modelado y evaluación
- **XGBoost:** Gradient Boosting avanzado
- **Imbalanced-learn:** Manejo de desbalanceo
- **Matplotlib & Seaborn:** Visualización

Métricas de Evaluación

- **ROC-AUC:** Capacidad discriminativa del modelo
- **Precision:** Proporción de predicciones positivas correctas
- **Recall:** Proporción de casos positivos detectados
- **F1-Score:** Media armónica de precision y recall
- **Matriz de Confusión:** Análisis detallado de errores

Proyecto desarrollado con fines educativos

Fecha: 2025

Dataset: Telco Customer Churn (Kaggle)