Guía Completa de Cliente Insight

Análisis de Customer Churn

Contents

Guía Completa de Cliente Insight - Análisis de Customer Churn **3**

Introducción . . . 3

Objetivo del Proyecto . . . 3

Dataset Utilizado . . . 4

Estructura del Notebook . . . 4

Bloque 1: Configuración e Importaciones . . . 4

¿Qué es? . . . 4

Código Relevante . . . 4

Explicación Detallada . . . 5

Resultados Esperados . . . 5

Mejores Prácticas Aplicadas . . . 5

Consideraciones Importantes . . . 5

Dependencias del Bloque . . . 5

Conexión con Bloques Siguientes . . . 6

Bloque 2: Carga de Datos . . . 6

¿Qué es? . . . 6

Código Relevante . . . 6

Explicación Detallada . . . 6

Resultados Esperados . . . 7

Mejores Prácticas Aplicadas . . . 7

Bloque 3: Análisis Exploratorio de Datos (EDA) . . . 7

¿Qué es? . . . 7

Código Relevante . . . 7

Explicación Detallada . . . 8

Resultados Esperados . . . 8

Mejores Prácticas Aplicadas . . . 9

Bloque 4: Limpieza de Datos . . . 9

¿Qué es? . . . 9

Código Relevante . . . 9
Explicación Detallada . . . 10
Resultados Esperados . . . 10
Mejores Prácticas Aplicadas . . . 10
Bloque 5: Feature Engineering . . . 10
¿Qué es? . . . 10
Código Relevante . . . 10
Explicación Detallada . . . 11
Resultados Esperados . . . 11
Mejores Prácticas Aplicadas . . . 11
Bloque 6: Preprocesamiento . . . 11
¿Qué es? . . . 11
Código Relevante . . . 11
Explicación Detallada . . . 12
Resultados Esperados . . . 12
Mejores Prácticas Aplicadas . . . 13
Bloque 7: División de Datos y Balanceo . . . 13
¿Qué es? . . . 13
Código Relevante . . . 13
Explicación Detallada . . . 14
Resultados Esperados . . . 14
Mejores Prácticas Aplicadas . . . 14
Bloque 8: Entrenamiento de Modelos Baseline . . . 14
¿Qué es? . . . 14
Código Relevante . . . 14
Explicación Detallada . . . 15
Resultados Esperados (Baseline sin balanceo) . . . 16
Mejores Prácticas Aplicadas . . . 16
Bloque 9: Evaluación del Mejor Modelo . . . 16
¿Qué es? . . . 16
Código Relevante . . . 16
Explicación Detallada . . . 18
Resultados Esperados . . . 18
Mejores Prácticas Aplicadas . . . 18
Bloque 10: Optimización de Hiperparámetros . . . 18
¿Qué es? . . . 18

Código Relevante . . . 18
Explicación Detallada . . . 19
Resultados Esperados . . . 20
Mejores Prácticas Aplicadas . . . 20
Bloque 11: Guardado del Modelo . . . 20
¿Qué es? . . . 20
Código Relevante . . . 20
Explicación Detallada . . . 21
Resultados Esperados . . . 22
Mejores Prácticas Aplicadas . . . 22
Bloque 12: Resumen Técnico . . . 22
¿Qué es? . . . 22
Código Relevante . . . 22
Explicación Detallada . . . 23
Resultados Esperados . . . 24
Mejores Prácticas Aplicadas . . . 24
Bloque 13: Generación de Informe Automático . . . 24
¿Qué es? . . . 24
Código Relevante . . . 24
Explicación Detallada . . . 25
Resultados Esperados . . . 25
Mejores Prácticas Aplicadas . . . 26
Conclusiones Generales . . . 26
Flujo Completo del Proyecto . . . 26
Métricas Clave del Proyecto . . . 26
Próximos Pasos Recomendados . . . 26

Guía Completa de Cliente Insight - Análisis de Customer Churn

Introducción

Este documento proporciona una guía detallada del notebook **Telco_Customer_Churn.ipynb**, un proyecto de Machine Learning diseñado para predecir la fuga de clientes (churn) en una empresa de telecomunicaciones.

Objetivo del Proyecto

Desarrollar un modelo predictivo que identifique clientes con alta probabilidad de abandonar el servicio, permitiendo implementar estrategias de retención proactivas.

Dataset Utilizado

- **Archivo**: `WA_Fn-UseC_-Telco-Customer-Churn.csv`
- **Registros**: 7,043 clientes
- **Variables**: 21 columnas (20 features + 1 target)
- **Target**: `Churn` (Yes/No)

Estructura del Notebook

El notebook está organizado en **13 bloques** principales que cubren todo el ciclo de vida de un proyecto de Machine Learning.

Bloque 1: Configuración e Importaciones

¿Qué es?

Este bloque establece el entorno de trabajo importando todas las librerías necesarias para el análisis de datos, visualización, preprocesamiento y modelado de Machine Learning.

Código Relevante

```
# Librerías de manipulación de datos
import pandas as pd
import numpy as np

# Librerías de visualización
import matplotlib.pyplot as plt
import seaborn as sns

# Librerías de Machine Learning
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
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 de evaluación
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, roc_auc_score, confusion_matrix,
                             classification_report, roc_curve, precision_recall_curve)

# Manejo de desbalanceo
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
```

```
# Configuración de semilla para reproducibilidad
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)

# Configuración de visualización
plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("husl")
```

Explicación Detallada

Librería	Propósito
pandas	Manipulación y análisis de datos tabulares
numpy	Operaciones numéricas y arrays
matplotlib/seaborn	Visualización de datos
sklearn	Algoritmos de ML, preprocesamiento y métricas
xgboost	Algoritmo de Gradient Boosting avanzado
imblearn	Técnicas para manejar datasets desbalanceados

Resultados Esperados

- Todas las librerías importadas sin errores
- Variable `RANDOM_STATE = 42` definida para reproducibilidad
- Configuración visual establecida para gráficos consistentes

Mejores Prácticas Aplicadas

1. **Reproducibilidad**: Se define `RANDOM_STATE = 42` para garantizar resultados consistentes entre ejecuciones
2. **Organización**: Las importaciones están agrupadas por funcionalidad
3. **Configuración visual**: Se establece un estilo de gráficos uniforme
4. **Supresión de warnings**: Se pueden agregar filtros para warnings no críticos

Consideraciones Importantes

- La semilla aleatoria (`RANDOM_STATE`) es crucial para:
  - Reproducir experimentos
  - Comparar modelos de forma justa
  - Debugging y validación
- Las versiones de las librerías deben documentarse para evitar incompatibilidades

Dependencias del Bloque

Este bloque no tiene dependencias previas y es el punto de partida del análisis.

Conexión con Bloques Siguientes

Las librerías importadas aquí se utilizan en todos los bloques posteriores:

- **Bloque 2**: pandas para carga de datos
- **Bloque 3**: matplotlib/seaborn para EDA
- **Bloque 6-10**: sklearn para modelado

Bloque 2: Carga de Datos

¿Qué es?

Este bloque carga el dataset de clientes de telecomunicaciones y realiza una inspección inicial para comprender la estructura y características de los datos.

Código Relevante

```
# Cargar el dataset
df = pd.read_csv('WA_Fn-UseC_-Telco-Customer-Churn.csv')

# Inspección inicial
print(f"Dimensiones del dataset: {df.shape}")
print(f"\nPrimeras filas:")
df.head()

# Información del dataset
df.info()

# Estadísticas descriptivas
df.describe()

# Verificar valores únicos por columna
for col in df.columns:
    print(f"{col}: {df[col].nunique()} valores únicos")
```

Explicación Detallada

Estructura del Dataset (21 columnas):

Variable	Tipo	Descripción
`customerID`	String	Identificador único del cliente
`gender`	Categórica	Género (Male/Female)
`SeniorCitizen`	Binaria	Es adulto mayor (0/1)
`Partner`	Categórica	Tiene pareja (Yes/No)
`Dependents`	Categórica	Tiene dependientes (Yes/No)
`tenure`	Numérica	Meses como cliente
`PhoneService`	Categórica	Tiene servicio telefónico

Variable	Tipo	Descripción
`MultipleLines`	Categórica	Múltiples líneas
`InternetService`	Categórica	Tipo de servicio de internet
`OnlineSecurity`	Categórica	Seguridad en línea
`OnlineBackup`	Categórica	Backup en línea
`DeviceProtection`	Categórica	Protección de dispositivo
`TechSupport`	Categórica	Soporte técnico
`StreamingTV`	Categórica	Streaming de TV
`StreamingMovies`	Categórica	Streaming de películas
`Contract`	Categórica	Tipo de contrato
`PaperlessBilling`	Categórica	Facturación sin papel
`PaymentMethod`	Categórica	Método de pago
`MonthlyCharges`	Numérica	Cargo mensual
`TotalCharges`	Numérica	Cargo total acumulado
`Churn`	Target	Abandono del cliente (Yes/No)

Resultados Esperados

- Dataset cargado: 7,043 filas × 21 columnas
- Sin errores de lectura
- Vista previa de los datos disponible

Mejores Prácticas Aplicadas

1. **Inspección inmediata**: Verificar dimensiones y tipos de datos
2. **Exploración de valores únicos**: Identificar variables categóricas vs numéricas
3. **Documentación de variables**: Entender el significado de cada columna

---

Bloque 3: Análisis Exploratorio de Datos (EDA)

¿Qué es?

Análisis visual y estadístico profundo del dataset para entender patrones, distribuciones y relaciones entre variables, especialmente aquellas relacionadas con el churn.

Código Relevante

```
# Distribución del target (Churn)
fig, axes = plt.subplots(1, 2, figsize=(12, 5))

# Gráfico de barras
df['Churn'].value_counts().plot(kind='bar', ax=axes[0], color=['#2ecc71', '#e74c3c'])
axes[0].set_title('Distribución de Churn')
axes[0].set_xlabel('Churn')
axes[0].set_ylabel('Cantidad')
```

```python
# Gráfico de pie
df['Churn'].value_counts().plot(kind='pie', ax=axes[1], autopct='%1.1f%%',
                                colors=['#2ecc71', '#e74c3c'])
axes[1].set_title('Proporción de Churn')
plt.tight_layout()
plt.show()

# Análisis de variables numéricas
numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
for i, col in enumerate(numerical_cols):
    sns.histplot(data=df, x=col, hue='Churn', ax=axes[i], kde=True)
    axes[i].set_title(f'Distribución de {col} por Churn')
plt.tight_layout()
plt.show()

# Matriz de correlación
plt.figure(figsize=(10, 8))
correlation_matrix = df[numerical_cols].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Matriz de Correlación')
plt.show()

# Análisis de variables categóricas vs Churn
categorical_cols = ['gender', 'Partner', 'Dependents', 'Contract',
                    'InternetService', 'PaymentMethod']
fig, axes = plt.subplots(2, 3, figsize=(15, 10))
axes = axes.flatten()
for i, col in enumerate(categorical_cols):
    sns.countplot(data=df, x=col, hue='Churn', ax=axes[i])
    axes[i].set_title(f'{col} vs Churn')
    axes[i].tick_params(axis='x', rotation=45)
plt.tight_layout()
plt.show()
```

Explicación Detallada

Hallazgos Clave del EDA:

Insight	Descripción
Desbalance de clases	~26.5% Churn vs ~73.5% No Churn
Tenure	Clientes nuevos (bajo tenure) tienen mayor churn
MonthlyCharges	Cargos mensuales altos correlacionan con mayor churn
Contract	Contratos mes a mes tienen mayor tasa de abandono
InternetService	Fiber optic muestra mayor churn que DSL
PaymentMethod	Electronic check asociado a mayor churn

Resultados Esperados

- Visualizaciones claras de distribuciones
- Identificación de patrones de churn

- Correlaciones entre variables numéricas
- Insights para feature engineering

Mejores Prácticas Aplicadas

1. **Visualización multidimensional**: Combinar gráficos de barras, pie, histogramas
2. **Segmentación por target**: Analizar distribuciones separadas por Churn
3. **Correlaciones**: Identificar multicolinealidad potencial

---

Bloque 4: Limpieza de Datos

¿Qué es?

Proceso de identificación y tratamiento de valores faltantes, conversión de tipos de datos incorrectos y preparación del dataset para el modelado.

Código Relevante

```
# Verificar valores faltantes
print("Valores faltantes por columna:")
print(df.isnull().sum())

# Detectar espacios en blanco en TotalCharges
df['TotalCharges'] = df['TotalCharges'].replace(' ', np.nan)

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

# Verificar nuevamente valores faltantes
print(f"\nValores faltantes en TotalCharges: {df['TotalCharges'].isnull().sum()}")

# Estrategia de imputación: rellenar con MonthlyCharges
# Los clientes con TotalCharges vacío son nuevos (tenure=0),
# por lo que su TotalCharges debería ser igual a su primer MonthlyCharges
df['TotalCharges'] = df['TotalCharges'].fillna(df['MonthlyCharges'])

# Eliminar customerID (no aporta al modelo)
df_clean = df.drop('customerID', axis=1)

# Convertir SeniorCitizen a categórico
df_clean['SeniorCitizen'] = df_clean['SeniorCitizen'].map({0: 'No', 1: 'Yes'})

# Verificar resultado final
print(f"\nDataset limpio: {df_clean.shape}")
print(f"Valores faltantes totales: {df_clean.isnull().sum().sum()}")
```

Explicación Detallada

Problemas Identificados y Soluciones:

Problema	Solución
TotalCharges con espacios en blanco	Convertir a NaN y luego a numérico
11 valores faltantes en TotalCharges	Rellenar con MonthlyCharges (clientes nuevos con tenure=0)
customerID no predictivo	Eliminación de la columna
SeniorCitizen como 0/1	Conversión a Yes/No para consistencia

Resultados Esperados

- Dataset sin valores faltantes
- Tipos de datos correctos
- 20 columnas (sin customerID)
- Datos listos para preprocesamiento

Mejores Prácticas Aplicadas

1. **Imputación lógica**: Usar MonthlyCharges para clientes nuevos (tenure=0)
2. **Preservación de información**: No eliminar filas innecesariamente
3. **Consistencia de tipos**: Uniformar formato de variables categóricas

---

Bloque 5: Feature Engineering

¿Qué es?

Creación de nuevas características derivadas de las existentes para capturar patrones más complejos y mejorar el poder predictivo del modelo.

Código Relevante

```
# 1. FEATURE ENGINEERING: Charge_Ratio
# Ratio que indica si el cliente paga más ahora que su promedio histórico
df['Charge_Ratio'] = df['MonthlyCharges'] / (df['TotalCharges'] / (df['tenure'] + 1))
df['Charge_Ratio'] = df['Charge_Ratio'].replace([np.inf, -np.inf], 1.0)
print("    Creada variable 'Charge_Ratio' (ratio de cargos mensuales)")

# 2. FEATURE ENGINEERING: Total_Services
# Contador de servicios contratados por el cliente
services = ['PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
            'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']

df['Total_Services'] = df[services].apply(
    lambda x: ((x != 'No') & (x != 'No internet service') & (x != 'No phone service')).sum(),
    axis=1
```

```
)
print("    Creada variable 'Total_Services' (total de servicios contratados)")
```

Explicación Detallada

Nuevas Features Creadas:

Feature	Lógica	Hipótesis
`Charge_Ratio`	MonthlyCharges / (TotalCharges / (tenure + 1))	Ratio que indica si el cliente paga más ahora que su promedio histórico
`Total_Services`	Conteo de servicios activos (donde valor != 'No')	Más servicios contratados = mayor retención

Resultados Esperados

- 2 nuevas features derivadas
- Mayor poder predictivo potencial
- Features interpretables para negocio

Mejores Prácticas Aplicadas

1. **Features basadas en dominio**: Usar conocimiento del negocio de telecomunicaciones
2. **Agregaciones significativas**: Combinar variables relacionadas
3. **Evitar data leakage**: No usar información futura

Bloque 6: Preprocesamiento

¿Qué es?

Transformación de variables para que sean compatibles con algoritmos de Machine Learning, incluyendo encoding de categóricas y escalado de numéricas.

Código Relevante

```
from sklearn.preprocessing import LabelEncoder, StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# Separar features y target
X = df_fe.drop('Churn', axis=1)
y = df_fe['Churn'].map({'No': 0, 'Yes': 1})

# Identificar tipos de columnas
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()
```

```
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()

print(f"Columnas numéricas ({len(numerical_cols)}): {numerical_cols}")
print(f"Columnas categóricas ({len(categorical_cols)}): {categorical_cols}")

# Crear pipeline de preprocesamiento
numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(drop='first', sparse_output=False,
                                        handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ]
)

# Aplicar preprocesamiento
X_processed = preprocessor.fit_transform(X)

# Obtener nombres de features después de encoding
cat_feature_names = preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_cols)
feature_names = numerical_cols + list(cat_feature_names)

print(f"\nDimensiones después de preprocesamiento: {X_processed.shape}")
print(f"Total de features: {len(feature_names)}")
```

Explicación Detallada

Transformaciones Aplicadas:

Tipo	Transformación	Razón
Numéricas	StandardScaler	Normalizar a media=0, std=1
Categóricas	OneHotEncoder	Crear variables dummy
Target	Label Encoding	Convertir Yes/No a 1/0

Parámetros Importantes:

- `drop='first'`: Evita multicolinealidad en One-Hot Encoding
- `handle_unknown='ignore'`: Maneja categorías nuevas en producción
- `sparse_output=False`: Retorna array denso para compatibilidad

Resultados Esperados

- X_processed: matriz numérica lista para ML
- ~45-50 features después de One-Hot Encoding
- Valores escalados (numéricas) o binarios (categóricas)

Mejores Prácticas Aplicadas

1. **Pipeline integrado**: ColumnTransformer para transformaciones consistentes
2. **Evitar data leakage**: fit_transform solo en datos de entrenamiento (después de split)
3. **Manejo de unknowns**: Preparado para datos de producción

---

Bloque 7: División de Datos y Balanceo

¿Qué es?

Separación del dataset en conjuntos de entrenamiento y prueba, junto con técnicas para manejar el desbalance de clases en la variable objetivo.

Código Relevante

```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTETomek

# División train/test (80/20)
X_train, X_test, y_train, y_test = train_test_split(
    X_processed, y,
    test_size=0.2,
    random_state=RANDOM_STATE,
    stratify=y
)

print(f"Conjunto de entrenamiento: {X_train.shape[0]} muestras")
print(f"Conjunto de prueba: {X_test.shape[0]} muestras")

# Verificar distribución del target
print(f"\nDistribución en train:")
print(f"  No Churn (0): {(y_train == 0).sum()} ({(y_train == 0).mean()*100:.1f}%)")
print(f"  Churn (1): {(y_train == 1).sum()} ({(y_train == 1).mean()*100:.1f}%)")

# Se evaluaron 3 técnicas de balanceo: SMOTE, SMOTE+Tomek, Undersampling
# Undersampling fue seleccionada como la mejor técnica

# Aplicar Undersampling (técnica seleccionada)
rus = RandomUnderSampler(random_state=RANDOM_STATE)
X_train_balanced, y_train_balanced = rus.fit_resample(X_train, y_train)

print(f"\nDespués de Undersampling:")
print(f"  No Churn (0): {(y_train_balanced == 0).sum()}")
print(f"  Churn (1): {(y_train_balanced == 1).sum()}")
print(f"  Ratio: {(y_train_balanced == 1).sum() / (y_train_balanced == 0).sum():.2f}")
```

Explicación Detallada

Estrategia de División:

Parámetro	Valor	Justificación
`test_size`	0.2	20% para evaluación final
`stratify`	y	Mantener proporción de clases
`random_state`	42	Reproducibilidad

Técnicas de Balanceo Evaluadas:

Técnica	ROC-AUC	Recall	Tiempo	Resultado
SMOTE	0.8258	76.47%	0.91s	-
SMOTE+Tomek	0.8217	74.33%	1.10s	-
Undersampling	0.8277	77.01%	0.57s	Seleccionada

Resultados Esperados

- Train: ~5,634 muestras (80%)
- Test: ~1,409 muestras (20%)
- Después de Undersampling: 2,990 muestras balanceadas (1,495 por clase)

Mejores Prácticas Aplicadas

1. **Stratify**: Garantiza proporciones iguales en train/test
2. **Balanceo solo en train**: Nunca en test para evitar data leakage
3. **Undersampling seleccionado**: Mejor ROC-AUC y menor tiempo de entrenamiento

---

Bloque 8: Entrenamiento de Modelos Baseline

¿Qué es?

Entrenamiento y evaluación de múltiples algoritmos de clasificación para establecer una línea base de rendimiento y seleccionar el modelo más prometedor.

Código Relevante

```
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

# Definir modelos a evaluar
```

```python
models = {
    'Logistic Regression': LogisticRegression(random_state=RANDOM_STATE, max_iter=1000),
    'Decision Tree': DecisionTreeClassifier(random_state=RANDOM_STATE),
    'Random Forest': RandomForestClassifier(random_state=RANDOM_STATE, n_estimators=100),
    'Gradient Boosting': GradientBoostingClassifier(random_state=RANDOM_STATE),
    'XGBoost': xgb.XGBClassifier(random_state=RANDOM_STATE, eval_metric='logloss'),
    'SVM': SVC(random_state=RANDOM_STATE, probability=True),
    'KNN': KNeighborsClassifier(n_neighbors=5)
}

# Entrenar y evaluar cada modelo
results = []
for name, model in models.items():
    print(f"\nEntrenando {name}...")

    # Entrenar modelo
    model.fit(X_train_balanced, y_train_balanced)

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

    # Calcular métricas
    metrics = {
        'Model': name,
        'Accuracy': accuracy_score(y_test, y_pred),
        'Precision': precision_score(y_test, y_pred),
        'Recall': recall_score(y_test, y_pred),
        'F1-Score': 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(metrics)

    print(f"  Accuracy: {metrics['Accuracy']:.4f}")
    print(f"  F1-Score: {metrics['F1-Score']:.4f}")
    print(f"  ROC-AUC: {metrics['ROC-AUC']:.4f}" if metrics['ROC-AUC'] else "")

# Crear DataFrame de resultados
results_df = pd.DataFrame(results)
results_df = results_df.sort_values('ROC-AUC', ascending=False)
print("\n" + "="*60)
print("RESUMEN DE RESULTADOS:")
print("="*60)
print(results_df.to_string(index=False))
```

Explicación Detallada

Modelos Evaluados:

Modelo	Tipo	Características
Logistic Regression	Lineal	Simple, interpretable, baseline sólido
Decision Tree	Árbol	Interpretable, propenso a overfitting

Modelo	Tipo	Características
Random Forest	Ensemble	Robusto, reduce varianza
Gradient Boosting	Ensemble	Alta precisión, secuencial
XGBoost	Ensemble	Optimizado, regularización
SVM	Kernel	Efectivo en alta dimensión
KNN	Instancias	No paramétrico, sensible a escala

Métricas Utilizadas:

Métrica	Fórmula	Interpretación para Churn
Accuracy	(TP+TN)/(Total)	% predicciones correctas
Precision	TP/(TP+FP)	De los predichos churn, % correctos
Recall	TP/(TP+FN)	De los churn reales, % detectados
F1-Score	2×(P×R)/(P+R)	Balance precision-recall
ROC-AUC	Área bajo curva ROC	Capacidad de discriminación

Resultados Esperados (Baseline sin balanceo)

Modelo	Accuracy	F1-Score	ROC-AUC
Logistic Regression	0.8084	0.5994	0.8483
Gradient Boosting	0.7956	0.5701	0.8453
Random Forest	0.7906	0.5564	0.8244
XGBoost	0.7828	0.5714	0.8234

> **Nota:** Logistic Regression obtuvo el mejor ROC-AUC en baseline. Tras optimización con Undersampling, se seleccionó **Logistic Regression Optimizado** como modelo final.

Mejores Prácticas Aplicadas

1. **Múltiples modelos**: Comparar antes de optimizar
2. **Métricas relevantes**: F1 y Recall importantes para churn
3. **Evaluación en test set**: Métricas finales en datos no vistos

---

Bloque 9: Evaluación del Mejor Modelo

¿Qué es?

Análisis detallado del modelo con mejor rendimiento, incluyendo matriz de confusión, curvas ROC y Precision-Recall, e interpretación de resultados.

Código Relevante

```
# Seleccionar mejor modelo (Logistic Regression Optimizado)
best_model_name = results_df.iloc[0]['Model']
best_model = models[best_model_name]

print(f"Mejor modelo: {best_model_name}")

# Predicciones finales
y_pred = best_model.predict(X_test)
y_pred_proba = best_model.predict_proba(X_test)[:, 1]

# 1. Matriz de Confusión
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[0])
axes[0].set_title(f'Matriz de Confusión - {best_model_name}')
axes[0].set_xlabel('Predicción')
axes[0].set_ylabel('Real')

# 2. Curva ROC
fpr, tpr, thresholds_roc = roc_curve(y_test, y_pred_proba)
roc_auc = roc_auc_score(y_test, y_pred_proba)
axes[1].plot(fpr, tpr, 'b-', label=f'ROC (AUC = {roc_auc:.3f})')
axes[1].plot([0, 1], [0, 1], 'r--', label='Random')
axes[1].set_xlabel('False Positive Rate')
axes[1].set_ylabel('True Positive Rate')
axes[1].set_title('Curva ROC')
axes[1].legend()

# 3. Curva Precision-Recall
precision, recall, thresholds_pr = precision_recall_curve(y_test, y_pred_proba)
axes[2].plot(recall, precision, 'g-')
axes[2].set_xlabel('Recall')
axes[2].set_ylabel('Precision')
axes[2].set_title('Curva Precision-Recall')
axes[2].axhline(y=y_test.mean(), color='r', linestyle='--', label='Baseline')
axes[2].legend()

plt.tight_layout()
plt.show()

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['No Churn', 'Churn']))

# Análisis de umbral óptimo
optimal_idx = np.argmax(tpr - fpr)
optimal_threshold = thresholds_roc[optimal_idx]
print(f"\nUmbral óptimo (Youden's J): {optimal_threshold:.3f}")
```

Explicación Detallada

Interpretación de la Matriz de Confusión:

	Pred: No Churn	Pred: Churn
Real: No Churn	TN (Verdaderos Negativos)	FP (Falsos Positivos)
Real: Churn	FN (Falsos Negativos)	TP (Verdaderos Positivos)

Importancia para el Negocio:

- **FN (Falsos Negativos)**: Clientes que se irán pero no detectamos - CRÍTICO
- **FP (Falsos Positivos)**: Clientes estables que predecimos se irán - Menor costo

Resultados Esperados

- Matriz de confusión visual
- ROC-AUC > 0.80 (buen discriminador)
- Curva PR sobre la baseline
- Classification report detallado

Mejores Prácticas Aplicadas

1. **Múltiples visualizaciones**: Perspectiva completa del rendimiento
2. **Análisis de umbral**: Optimizar según costo de negocio
3. **Baseline comparison**: Comparar con clasificador aleatorio

---

Bloque 10: Optimización de Hiperparámetros

¿Qué es?

Búsqueda sistemática de los mejores hiperparámetros del modelo seleccionado para maximizar su rendimiento predictivo.

Código Relevante

```
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV

# Definir espacio de búsqueda para Logistic Regression
param_distributions = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga'],
    'max_iter': [500, 1000, 2000],
```

```
    'subsample': [0.6, 0.7, 0.8, 0.9],
    'colsample_bytree': [0.6, 0.7, 0.8, 0.9],
    'gamma': [0, 0.1, 0.2, 0.3],
    'reg_alpha': [0, 0.1, 0.5, 1],
    'reg_lambda': [0, 0.1, 0.5, 1]
}

# Modelo base para optimización
xgb_model = xgb.XGBClassifier(
    random_state=RANDOM_STATE,
    eval_metric='logloss',
    use_label_encoder=False
)

# RandomizedSearchCV para búsqueda eficiente
random_search = RandomizedSearchCV(
    estimator=xgb_model,
    param_distributions=param_distributions,
    n_iter=50,  # Número de combinaciones a probar
    cv=5,       # Cross-validation 5-fold
    scoring='roc_auc',
    random_state=RANDOM_STATE,
    n_jobs=-1,
    verbose=2
)

# Ejecutar búsqueda
print("Iniciando búsqueda de hiperparámetros...")
random_search.fit(X_train_balanced, y_train_balanced)

# Mejores parámetros encontrados
print("\nMejores hiperparámetros:")
for param, value in random_search.best_params_.items():
    print(f"  {param}: {value}")

print(f"\nMejor ROC-AUC en CV: {random_search.best_score_:.4f}")

# Modelo optimizado
best_model_optimized = random_search.best_estimator_

# Evaluar en test set
y_pred_optimized = best_model_optimized.predict(X_test)
y_pred_proba_optimized = best_model_optimized.predict_proba(X_test)[:, 1]

print("\nMétricas del modelo optimizado en test set:")
print(f"  Accuracy: {accuracy_score(y_test, y_pred_optimized):.4f}")
print(f"  F1-Score: {f1_score(y_test, y_pred_optimized):.4f}")
print(f"  ROC-AUC: {roc_auc_score(y_test, y_pred_proba_optimized):.4f}")
```

Explicación Detallada

Hiperparámetros de Logistic Regression:

Parámetro	Rango	Efecto
`C`	0.001-100	Fuerza de regularización inversa
`penalty`	l1, l2	Tipo de regularización
`solver`	liblinear, saga	Algoritmo de optimización
`max_iter`	500-2000	Iteraciones máximas
`learning_rate`	0.01-0.2	Tasa de aprendizaje
`min_child_weight`	1-7	Peso mínimo en hojas
`subsample`	0.6-0.9	% de muestras por árbol
`colsample_bytree`	0.6-0.9	% de features por árbol
`gamma`	0-0.3	Regularización de poda
`reg_alpha/lambda`	0-1	Regularización L1/L2

RandomizedSearchCV vs GridSearchCV:

Método	Ventajas	Desventajas
Randomized	Rápido, explora espacio amplio	Puede perder óptimo
Grid	Exhaustivo, encuentra óptimo	Muy lento en espacios grandes

Resultados Esperados

- Mejora de 1-3% en ROC-AUC
- Hiperparámetros óptimos documentados
- Modelo listo para producción

Mejores Prácticas Aplicadas

1. **Cross-validation**: Evaluación robusta durante búsqueda
2. **Scoring relevante**: Optimizar ROC-AUC para clasificación desbalanceada
3. **n_iter suficiente**: Balance entre exploración y tiempo

Bloque 11: Guardado del Modelo

¿Qué es?

Exportación del modelo entrenado y sus componentes asociados para su uso posterior en producción o inferencia.

Código Relevante

```
import joblib
import json
from datetime import datetime

# Crear directorio para modelos
import os
```

```
model_dir = 'models'
os.makedirs(model_dir, exist_ok=True)

# Timestamp para versionado
timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')

# 1. Guardar modelo principal
model_path = f'{model_dir}/churn_model_{timestamp}.joblib'
joblib.dump(best_model_optimized, model_path)
print(f"Modelo guardado en: {model_path}")

# 2. Guardar preprocesador
preprocessor_path = f'{model_dir}/preprocessor_{timestamp}.joblib'
joblib.dump(preprocessor, preprocessor_path)
print(f"Preprocesador guardado en: {preprocessor_path}")

# 3. Guardar metadata del modelo
metadata = {
    'model_name': 'Logistic Regression Optimizado',
    'version': timestamp,
    'training_date': datetime.now().isoformat(),
    'metrics': {
        'accuracy': float(accuracy_score(y_test, y_pred_optimized)),
        'precision': float(precision_score(y_test, y_pred_optimized)),
        'recall': float(recall_score(y_test, y_pred_optimized)),
        'f1_score': float(f1_score(y_test, y_pred_optimized)),
        'roc_auc': float(roc_auc_score(y_test, y_pred_proba_optimized))
    },
    'best_params': random_search.best_params_,
    'features': feature_names,
    'target': 'Churn',
    'threshold': float(optimal_threshold)
}

metadata_path = f'{model_dir}/model_metadata_{timestamp}.json'
with open(metadata_path, 'w') as f:
    json.dump(metadata, f, indent=2)
print(f"Metadata guardada en: {metadata_path}")

# 4. Guardar nombres de features
features_path = f'{model_dir}/feature_names_{timestamp}.json'
with open(features_path, 'w') as f:
    json.dump(feature_names, f)
print(f"Features guardadas en: {features_path}")

print(f"\n  Modelo exportado exitosamente - Versión: {timestamp}")
```

Explicación Detallada

Archivos Generados:

Archivo	Formato	Contenido
`churn_model_*.joblib`	Binario	Modelo entrenado serializado
`preprocessor_*.joblib`	Binario	Pipeline de preprocesamiento
`model_metadata_*.json`	JSON	Métricas, parámetros, versión
`feature_names_*.json`	JSON	Lista de features esperadas

Estructura del Metadata:

```
{
  "model_name": "Logistic Regression Optimizado",
  "version": "20251128_170912",
  "training_date": "2025-11-28T17:09:12",
  "metrics": {
    "accuracy": 0.7424,
    "roc_auc": 0.8503
  },
  "best_params": {...},
  "threshold": 0.45
}
```

Resultados Esperados

- 4 archivos en directorio `models/`
- Modelo reproducible y versionado
- Metadata completa para auditoría

Mejores Prácticas Aplicadas

1. **Versionado con timestamp**: Trazabilidad de versiones
2. **Metadata completa**: Reproducibilidad y documentación
3. **Preprocesador incluido**: Pipeline completo para producción

---

Bloque 12: Resumen Técnico

¿Qué es?

Documentación estructurada de la metodología utilizada, decisiones tomadas, resultados obtenidos y recomendaciones para uso del modelo.

Código Relevante

```
# Generar resumen técnico
print("="*70)
print("RESUMEN TÉCNICO DEL PROYECTO")
print("="*70)
```

```
print("\n DATASET:")
print(f"  • Registros totales: 7,043")
print(f"  • Features originales: 21")
print(f"  • Features finales: {len(feature_names)}")
print(f"  • Target: Churn (26.5% positivos)")

print("\n PREPROCESAMIENTO:")
print(f"  • Valores faltantes: Imputación con MonthlyCharges para clientes nuevos")
print(f"  • Encoding: OneHotEncoder (drop='first')")
print(f"  • Scaling: StandardScaler para numéricas")
print(f"  • Balanceo: Undersampling (seleccionado tras evaluar SMOTE, SMOTE+Tomek)")

print("\n MODELO SELECCIONADO:")
print(f"  • Algoritmo: Logistic Regression Optimizado")
print(f"  • Optimización: RandomizedSearchCV (50 iter, 5-fold CV)")
print(f"  • Técnica de Balanceo: Undersampling")

print("\n MÉTRICAS FINALES:")
print(f"  • Accuracy: {accuracy_score(y_test, y_pred_optimized):.4f}")
print(f"  • Precision: {precision_score(y_test, y_pred_optimized):.4f}")
print(f"  • Recall: {recall_score(y_test, y_pred_optimized):.4f}")
print(f"  • F1-Score: {f1_score(y_test, y_pred_optimized):.4f}")
print(f"  • ROC-AUC: {roc_auc_score(y_test, y_pred_proba_optimized):.4f}")

print("\n UMBRAL DE DECISIÓN:")
print(f"  • Umbral óptimo: {optimal_threshold:.3f}")
print(f"  • Criterio: Youden's J (maximizar TPR - FPR)")

print("\n FEATURES MÁS IMPORTANTES:")
if hasattr(best_model_optimized, 'feature_importances_'):
    importances = pd.DataFrame({
        'feature': feature_names,
        'importance': best_model_optimized.feature_importances_
    }).sort_values('importance', ascending=False)

    for i, row in importances.head(10).iterrows():
        print(f"  {i+1}. {row['feature']}: {row['importance']:.4f}")

print("\n" + "="*70)
```

Explicación Detallada

Componentes del Resumen:

Sección	Contenido
Dataset	Dimensiones, distribución del target
Preprocesamiento	Técnicas aplicadas, parámetros
Modelo	Algoritmo, método de optimización
Métricas	Rendimiento en test set
Umbral	Punto de corte para clasificación
Feature Importance	Variables más predictivas

Resultados Esperados

- Documentación completa del pipeline
- Métricas finales claras
- Top 10 features más importantes

Mejores Prácticas Aplicadas

1. **Documentación inline**: Resumen ejecutivo en el notebook
2. **Métricas completas**: Todas las métricas relevantes
3. **Feature importance**: Interpretabilidad del modelo

Bloque 13: Generación de Informe Automático

¿Qué es?

Creación automatizada de un informe en formato Markdown que resume todo el proyecto y puede compartirse fácilmente con stakeholders.

Código Relevante

```
# Generar informe en Markdown
report_content = f"""# Informe de Modelo de Predicción de Churn

## Información General
- **Fecha de generación**: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}
- **Versión del modelo**: {timestamp}
- **Autor**: Data Science Team

## Resumen Ejecutivo
Este modelo predice la probabilidad de que un cliente abandone el servicio de telecomunicaciones,
permitiendo implementar estrategias de retención proactivas.

## Dataset
| Característica | Valor |
|----------------|-------|
| Registros totales | 7,043 |
| Features | {len(feature_names)} |
| Tasa de Churn | 26.5% |
| División Train/Test | 80/20 |

## Rendimiento del Modelo
| Métrica | Valor |
|---------|-------|
| Accuracy | {accuracy_score(y_test, y_pred_optimized):.2%} |
| Precision | {precision_score(y_test, y_pred_optimized):.2%} |
| Recall | {recall_score(y_test, y_pred_optimized):.2%} |
| F1-Score | {f1_score(y_test, y_pred_optimized):.2%} |
```

```
| ROC-AUC | {roc_auc_score(y_test, y_pred_proba_optimized):.4f} |

## Interpretación de Negocio
- **Recall {recall_score(y_test, y_pred_optimized):.0%}**: De cada 100 clientes que realmente se van,
  el modelo detecta {int(recall_score(y_test, y_pred_optimized)*100)}.
- **Precision {precision_score(y_test, y_pred_optimized):.0%}**: De cada 100 clientes que el modelo
  predice como churn, {int(precision_score(y_test, y_pred_optimized)*100)} realmente se van.

## Recomendaciones

1. Implementar campaña de retención para clientes con score > {optimal_threshold:.2f}
2. Priorizar clientes con contratos mes a mes y tenure bajo
3. Ofrecer beneficios en servicios de seguridad y soporte técnico
4. Revisar precios para clientes con MonthlyCharges alto

## Archivos Generados
- `{model_path}` - Modelo serializado
- `{preprocessor_path}` - Preprocesador
- `{metadata_path}` - Metadata

---
*Generado automáticamente por el pipeline de ML*
"""

# Guardar informe
report_path = f'{model_dir}/churn_model_report_{timestamp}.md'
with open(report_path, 'w') as f:
    f.write(report_content)

print(f"  Informe generado: {report_path}")
print("\n" + "="*70)
print("  PIPELINE COMPLETADO EXITOSAMENTE")
print("="*70)
```

Explicación Detallada

Estructura del Informe:

Sección	Audiencia	Contenido
Resumen Ejecutivo	Directivos	Propósito del modelo
Dataset	Técnicos	Características de los datos
Rendimiento	Ambos	Métricas de evaluación
Interpretación	Negocio	Traducción a impacto
Recomendaciones	Negocio	Acciones sugeridas
Archivos	Técnicos	Artefactos generados

Resultados Esperados

- Archivo Markdown legible
- Métricas formateadas como porcentajes
- Recomendaciones accionables
- Referencias a archivos generados

Mejores Prácticas Aplicadas

1. **Formato Markdown**: Universal y portable
2. **Interpretación de negocio**: Traducir métricas a impacto
3. **Automatización**: Informe generado sin intervención manual
4. **Versionado**: Timestamp en nombre del archivo

Conclusiones Generales

Flujo Completo del Proyecto

```
1. Config       2. Carga         3. EDA

6. Prepro       5. FeatEng     4. Limpieza

7. Split       8. Baseline      9. Eval

12. Resumen     11. Guardar     10. Optim


13. Informe
```

Métricas Clave del Proyecto

Aspecto	Valor
Modelo Final	Logistic Regression Optimizado
ROC-AUC	0.8503
Recall	79.41%
Precision	50.94%
Accuracy	74.24%
Técnica de Balanceo	Undersampling
Archivos generados	5

Próximos Pasos Recomendados

1. **Monitoreo**: Implementar seguimiento de drift del modelo
2. **A/B Testing**: Validar efectividad de campañas de retención
3. **Reentrenamiento**: Programar actualización trimestral

4. **API**: Desarrollar endpoint para predicciones en tiempo real

---

Documento generado como parte del análisis del notebook Telco_Customer_Churn.ipynb