Desafío: Telecom X: **"Análisis de Evasión de Clientes"**

Has sido contratado como asistente de análisis de datos en Telecom X y formarás parte del proyecto "Churn de Clientes". La empresa enfrenta una alta tasa de cancelaciones y necesita comprender los factores que llevan a la pérdida de clientes.

Tu desafío será recopilar, procesar y analizar los datos, utilizando Python y sus principales bibliotecas para extraer información valiosa.

A partir de tu análisis, el equipo de Data Science podrá avanzar en modelos predictivos y desarrollar estrategias para reducir la evasión.

1. Importar y manipular datos desde una API

Para iniciar tu análisis, necesitarás importar los datos de la API de Telecom X. Estos datos están disponibles en formato JSON y contienen información esencial sobre los clientes, incluyendo datos demográficos, tipo de servicio contratado y estado de evasión.

Enlaces:

- **JSON:** https://github.com/ingridcristh/challenge2-data-science-LATAM/blob/main/TelecomX_Data.json
- **Diccionario**: https://github.com/ingridcristh/challenge2-data-science-LATAM/blob/main/TelecomX_diccionario.md

Pandas es una de las bibliotecas más populares y poderosas en Python para el análisis y manipulación de datos.

Se utiliza principalmente para:

- **Manipulación de datos:** Permite cargar, limpiar, transformar y reorganizar datos de manera eficiente. Puedes manejar datos faltantes, cambiar formatos, combinar conjuntos de datos, etc.
- **Estructuras de datos:** Introduce dos estructuras de datos principales: Series (similar a una columna en una hoja de cálculo) y **DataFrame** (similar a una tabla o una hoja de cálculo, con filas y columnas).
- **Análisis exploratorio de datos (EDA):** Facilita la exploración inicial de los datos para entender su distribución, identificar patrones, anomalías y relaciones entre variables.
- **Carga y guardado de datos:** Puede leer y escribir datos en diversos formatos como **CSV, Excel, SQL, JSON y muchos otros.**
- **Preparación para modelado:** Es una herramienta fundamental para preparar los datos antes de aplicar algoritmos de machine learning o modelos estadísticos.

```
# Importar Pandas #
import pandas as pd
```

2. ETL (Extracción, Transformación y Carga)

2.1 Extracción

```
# Importar Json y MD #

# Cargar los datos desde la url en formato Json #
# Y los convierte en un DATAFRAME
TelecomX_Data = pd.read_json("https://raw.githubusercontent.com/ingridcristh/challenge2-data-science-LATAM/main/Telec
```

```
# URL del archivo Markdown
url_diccionario = "https://raw.githubusercontent.com/ingridcristh/challenge2-data-science-LATAM/main/TelecomX_diccion
```

Mostrando el Diccionario

- **IPython.display** es un módulo de IPython que contiene utilidades para mostrar objetos en notebooks (Jupyter, Colab) con formato HTML, imágenes, audio, vídeo, Markdown, etc.
  - **Display**: Es una función general para mostrar objetos de Python con la mejor representación disponible (HTML, tablas, Markdown, etc.), más flexible que print.
  - **MarkDown**: Es una clase que recibe texto en formato Markdown.
    - **por ejemplo**: con **títulos, listas, tablas** y lo envuelve como un objeto **"mostrable"** en el notebook.

```
# Ver el contenido de md #

#Importar request#
import requests
from IPython.display import display, Markdown
# Descarga el contenido del archivo desde la web #
# Se manda una solicitud tipo get a la url y devuelve un oobjeto#
# que contiene el archivo, .text extrae esa respuesta como cadena #
# De texto
texto = requests.get(url_diccionario).text
# Muestra en Colab como Markdown ya con formato #
# Markdown (texto) crea un objeto especial "texto en formato markdown"
# Display muestra esa salida.
display(Markdown(texto))
```

Diccionario de datos

- `customerID`: número de identificación único de cada cliente
- `Churn`: si el cliente dejó o no la empresa
- `gender`: género (masculino y femenino)
- `SeniorCitizen`: información sobre si un cliente tiene o no una edad igual o mayor a 65 años
- `Partner`: si el cliente tiene o no una pareja
- `Dependents`: si el cliente tiene o no dependientes
- `tenure`: meses de contrato del cliente
- `PhoneService`: suscripción al servicio telefónico
- `MultipleLines`: suscripción a más de una línea telefónica
- `InternetService`: suscripción a un proveedor de internet
- `OnlineSecurity`: suscripción adicional de seguridad en línea
- `OnlineBackup`: suscripción adicional de respaldo en línea
- `DeviceProtection`: suscripción adicional de protección del dispositivo
- `TechSupport`: suscripción adicional de soporte técnico, menor tiempo de espera
- `StreamingTV`: suscripción de televisión por cable
- `StreamingMovies`: suscripción de streaming de películas
- `Contract`: tipo de contrato
- `PaperlessBilling`: si el cliente prefiere recibir la factura en línea
- `PaymentMethod`: forma de pago
- `Charges.Monthly`: total de todos los servicios del cliente por mes
- `Charges.Total`: total gastado por el cliente

```
# ver contenido del json #
TelecomX_Data
```

	customerID	Churn	customer	phone	internet	account
0	0002-ORFBO	No	{'gender': 'Female', 'SeniorCitizen': 0, 'Part...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'DSL', 'OnlineSecurity': '...	{'Contract': 'One year', 'PaperlessBilling': '...
1	0003-MKNFE	No	{'gender': 'Male', 'SeniorCitizen': 0, 'Partne...	{'PhoneService': 'Yes', 'MultipleLines': 'Yes'}	{'InternetService': 'DSL', 'OnlineSecurity': '...	{'Contract': 'Month-to-month', 'PaperlessBilli...
2	0004-TLHLJ	Yes	{'gender': 'Male', 'SeniorCitizen': 0, 'Partne...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'Fiber optic', 'OnlineSecu...	{'Contract': 'Month-to-month', 'PaperlessBilli...
3	0011-IGKFF	Yes	{'gender': 'Male', 'SeniorCitizen': 1, 'Partne...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'Fiber optic', 'OnlineSecu...	{'Contract': 'Month-to-month', 'PaperlessBilli...
4	0013-EXCHZ	Yes	{'gender': 'Female', 'SeniorCitizen': 1, 'Part...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'Fiber optic', 'OnlineSecu...	{'Contract': 'Month-to-month', 'PaperlessBilli...
...	...	...	...	...	...	...
7262	9987-LUTYD	No	{'gender': 'Female', 'SeniorCitizen': 0, 'Part...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'DSL', 'OnlineSecurity': '...	{'Contract': 'One year', 'PaperlessBilling': '...
7263	9992-RRAMN	Yes	{'gender': 'Male', 'SeniorCitizen': 0, 'Partne...	{'PhoneService': 'Yes', 'MultipleLines': 'Yes'}	{'InternetService': 'Fiber optic', 'OnlineSecu...	{'Contract': 'Month-to-month', 'PaperlessBilli...
7264	9992-UJOEL	No	{'gender': 'Male', 'SeniorCitizen': 0, 'Partne...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'DSL', 'OnlineSecurity': '...	{'Contract': 'Month-to-month', 'PaperlessBilli...
7265	9993-LHIEB	No	{'gender': 'Male', 'SeniorCitizen': 0, 'Partne...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'DSL', 'OnlineSecurity': '...	{'Contract': 'Two year', 'PaperlessBilling': '...
7266	9995-HOTOH	No	{'gender': 'Male', 'SeniorCitizen': 0,	{'PhoneService': 'No', 'MultipleLines': 'No ph	{'InternetService': 'DSL', 'OnlineSecurity': '	{'Contract': 'Two year', 'PaperlessBilling': '

Pasos siguientes: New interactive sheet

2.2 Transformación

Flattening o Aplanado

Obteniendo Columnas

Al visualizar el dataFrame anterior nos damos cuenta de que existen Json anidados.
Podemos usar normalize para obtener mas columnas.

pd.json_normalize(columna) toma cada diccionario de esa serie y lo convierte en columnas, una por clave del JSON.

```
from pandas.tseries.offsets import CustomBusinessDay
# Quitando los JSON aniados #
# Las Columnas Objetivo SON:

# customer #
custommer_df=pd.json_normalize(TelecomX_Data['customer'])
# phone #
phone_df=pd.json_normalize(TelecomX_Data['phone'])
# internet #
internet_df=pd.json_normalize(TelecomX_Data['internet'])
# account#
account_df=pd.json_normalize(TelecomX_Data['account'])
```

Uniendo las Columnas en Un solo DataFrame

Vamos a unir las columnas simples originales y las nuevas obtenidas en el paso anterior.

```
#Nuevo dataFrame Completo #
TelecomX_Data_Final=pd.concat([TelecomX_Data,custommer_df,phone_df,internet_df,account_df],axis=1)
TelecomX_Data_Final.head()
```

	customerID	Churn	customer	phone	internet	account	gender	SeniorCitizen	Partner	Depe
0	0002-ORFBO	No	{'gender': 'Female', 'SeniorCitizen': 0, 'Part...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'DSL', 'OnlineSecurity': '...	{'Contract': 'One year', 'PaperlessBilling': '...	Female	0	Yes	
1	0003-MKNFE	No	{'gender': 'Male', 'SeniorCitizen': 0, 'Partne...	{'PhoneService': 'Yes', 'MultipleLines': 'Yes'}	{'InternetService': 'DSL', 'OnlineSecurity': '...	{'Contract': 'Month-to-month', 'PaperlessBilli...	Male	0	No	
2	0004-TLHLJ	Yes	{'gender': 'Male', 'SeniorCitizen': 0, 'Partne...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'Fiber optic', 'OnlineSecu...	{'Contract': 'Month-to-month', 'PaperlessBilli...	Male	0	No	
3	0011-IGKFF	Yes	{'gender': 'Male', 'SeniorCitizen': 1, 'Partne...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'Fiber optic', 'OnlineSecu...	{'Contract': 'Month-to-month', 'PaperlessBilli...	Male	1	Yes	
4	0013-EXCHZ	Yes	{'gender': 'Female', 'SeniorCitizen': 1, 'Part...	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'Fiber optic', 'OnlineSecu...	{'Contract': 'Month-to-month', 'PaperlessBilli...	Female	1	Yes	

5 rows × 25 columns

Analizando: Las columnas vemos que aun hay columnas con **JSON** anidados.

Debemos de quitar los json anidados de las columnas:

- **customer**
- **phone**
- **internet**
- **account**

```
# Analizando columnas con json anidados #
df=TelecomX_Data_Final

# Ver lo Json anidados, ver si aun hay diccionarios #
for col in df.columns:
  if df[col].apply(lambda x: isinstance(x,dict)).any():
    print("Columna con JSON anidado",col)
```

```
Columna con JSON anidado customer
Columna con JSON anidado phone
Columna con JSON anidado internet
Columna con JSON anidado account
```

Explicación:

if df[col].apply(lambda x: isinstance(x, dict)).any():

- **df[col]:** selecciona la columna col (por ejemplo, customer_2).
- **.apply(lambda x:** isinstance(x, dict)): aplica una función lambda a cada celda de la columna. Para cada valor x (que puede ser str, int, dict, etc.), isinstance(x, dict) devuelve True si es un diccionario, False si no. El resultado es una serie de booleanos (uno por fila).
- **.any():** devuelve True si al menos una de esas filas es True (hay al menos un dict en esa columna).

Expandiendo las columnas: **"flattening" ó "Aplanado"**

- **customer**
- **phone**
- **internet**
- **account**

```
# Expandiendo customer #
# Expandir customer

# Normalizamos el df = data frame #
nuevas_cols = pd.json_normalize(df["customer"])
# Agregamos el prefijo para saber que es una sub col de customer#
nuevas_cols = nuevas_cols.add_prefix("customer_")
# eliminas columa customer y concatenas con columna "nuevas_cols"#
df = pd.concat([df.drop(columns=["customer"]), nuevas_cols], axis=1)

print("✅ customer expandido")
df.head()
```

```
✅ customer expandido
```

	customerID	Churn	phone	internet	account	gender	SeniorCitizen	Partner	Dependents	tenure
0	0002-ORFBO	No	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'DSL', 'OnlineSecurity': '...	{'Contract': 'One year', 'PaperlessBilling': '...	Female	0	Yes	Yes	9
1	0003-MKNFE	No	{'PhoneService': 'Yes', 'MultipleLines': 'Yes'}	{'InternetService': 'DSL', 'OnlineSecurity': '...	{'Contract': 'Month-to-month', 'PaperlessBilli...	Male	0	No	No	9
2	0004-TLHLJ	Yes	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'Fiber optic', 'OnlineSecu...	{'Contract': 'Month-to-month', 'PaperlessBilli...	Male	0	No	No	4
3	0011-IGKFF	Yes	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'Fiber optic', 'OnlineSecu...	{'Contract': 'Month-to-month', 'PaperlessBilli...	Male	1	Yes	No	13
4	0013-EXCHZ	Yes	{'PhoneService': 'Yes', 'MultipleLines': 'No'}	{'InternetService': 'Fiber optic', 'OnlineSecu...	{'Contract': 'Month-to-month', 'PaperlessBilli...	Female	1	Yes	No	3

5 rows × 29 columns

```
# Expandiendo phone#
# Normalizar #
nuevas_cols = pd.json_normalize(df["phone"])
# agregas el prefijo#
nuevas_cols = nuevas_cols.add_prefix("phone_")
# Eliminas columna phone y concatenas con nuevas_cols#
df = pd.concat([df.drop(columns=["phone"]), nuevas_cols], axis=1)

print("✅ phone expandido")
```

```
✅ phone expandido
```

```
# Expandiendo internet #
# Normalizar #
nuevas_cols = pd.json_normalize(df["internet"])
# Agregas prefijo #
nuevas_cols = nuevas_cols.add_prefix("internet_")
# Eliminas columna internet y concatenas con nuevas_cols #
df = pd.concat([df.drop(columns=["internet"]), nuevas_cols], axis=1)
```

```
print("✅ internet expandido")
```

```
✅ internet expandido
```

```
# Expandiendo account #
# Normalizar #
nuevas_cols = pd.json_normalize(df["account"])
# Agregas prefijo #
nuevas_cols = nuevas_cols.add_prefix("account_")
# # Eliminas columna account y concatenas con nuevas_cols #
df = pd.concat([df.drop(columns=["account"]), nuevas_cols], axis=1)

print("✅ account expandido")
```

```
✅ account expandido
```

Verificando si aun existen JSON anidados:

```
# Analizando columnas con json anidados #

# Ver lo Json anidados, ver si aun hay diccionarios #
for col in df.columns:
  if df[col].apply(lambda x: isinstance(x,dict)).any():
    print("Columna con JSON anidado",col)
print("No hay JSON anidados")
```

```
No hay JSON anidados
```

```
# Mostrando El dataFrame final #
df.head()
```

	customerID	Churn	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService
0	0002-ORFBO	No	Female	0	Yes	Yes	9	Yes	No	DSL
1	0003-MKNFE	No	Male	0	No	No	9	Yes	Yes	DSL
2	0004-TLHLJ	Yes	Male	0	No	No	4	Yes	No	Fiber optic
3	0011-IGKFF	Yes	Male	1	Yes	No	13	Yes	No	Fiber optic
4	0013-EXCHZ	Yes	Female	1	Yes	No	3	Yes	No	Fiber optic

5 rows × 40 columns

Analizamos el DataFrame que tenemos ya aplanado.

Necesitamos conocer el propio dataFrame.

```
# Analizando los datos #

#1 Tipos de Datos#
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7267 entries, 0 to 7266
Data columns (total 40 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   customerID                7267 non-null   object
 1   Churn                     7267 non-null   object
 2   gender                    7267 non-null   object
 3   SeniorCitizen             7267 non-null   int64
 4   Partner                   7267 non-null   object
```

```
 5   Dependents                7267 non-null   object
 6   tenure                    7267 non-null   int64
 7   PhoneService              7267 non-null   object
 8   MultipleLines             7267 non-null   object
 9   InternetService           7267 non-null   object
 10  OnlineSecurity            7267 non-null   object
 11  OnlineBackup              7267 non-null   object
 12  DeviceProtection          7267 non-null   object
 13  TechSupport               7267 non-null   object
 14  StreamingTV               7267 non-null   object
 15  StreamingMovies           7267 non-null   object
 16  Contract                  7267 non-null   object
 17  PaperlessBilling          7267 non-null   object
 18  PaymentMethod             7267 non-null   object
 19  Charges.Monthly           7267 non-null   float64
 20  Charges.Total             7267 non-null   object
 21  customer_gender           7267 non-null   object
 22  customer_SeniorCitizen    7267 non-null   int64
 23  customer_Partner          7267 non-null   object
 24  customer_Dependents       7267 non-null   object
 25  customer_tenure           7267 non-null   int64
 26  phone_PhoneService        7267 non-null   object
 27  phone_MultipleLines       7267 non-null   object
 28  internet_InternetService  7267 non-null   object
 29  internet_OnlineSecurity   7267 non-null   object
 30  internet_OnlineBackup     7267 non-null   object
 31  internet_DeviceProtection 7267 non-null   object
 32  internet_TechSupport      7267 non-null   object
 33  internet_StreamingTV      7267 non-null   object
 34  internet_StreamingMovies  7267 non-null   object
 35  account_Contract          7267 non-null   object
 36  account_PaperlessBilling  7267 non-null   object
 37  account_PaymentMethod     7267 non-null   object
 38  account_Charges.Monthly   7267 non-null   float64
 39  account_Charges.Total     7267 non-null   object
dtypes: float64(2), int64(4), object(34)
memory usage: 2.2+ MB
```

```
# 2 Contar valores nulos por columna #
print(df.isnull().sum())
```

```
customerID                0
Churn                     0
gender                    0
SeniorCitizen             0
Partner                   0
Dependents                0
tenure                    0
PhoneService              0
MultipleLines             0
InternetService           0
OnlineSecurity            0
OnlineBackup              0
DeviceProtection          0
TechSupport               0
StreamingTV               0
StreamingMovies           0
Contract                  0
PaperlessBilling          0
PaymentMethod             0
Charges.Monthly           0
Charges.Total             0
customer_gender           0
customer_SeniorCitizen    0
customer_Partner          0
customer_Dependents       0
customer_tenure           0
phone_PhoneService        0
phone_MultipleLines       0
internet_InternetService  0
internet_OnlineSecurity   0
internet_OnlineBackup     0
internet_DeviceProtection 0
internet_TechSupport      0
internet_StreamingTV      0
internet_StreamingMovies  0
account_Contract          0
account_PaperlessBilling  0
```

```
account_PaymentMethod        0
account_Charges.Monthly      0
account_Charges.Total        0
dtype: int64
```

```
# 3. Limpiar nombres de Columanas #
# cambiar . por _ #
df.columns = df.columns.str.lower().str.replace('.', '_')
df.head()
```

	customerid	churn	gender	seniorcitizen	partner	dependents	tenure	phoneservice	multiplelines	internetservice
0	0002-ORFBO	No	Female	0	Yes	Yes	9	Yes	No	DSL
1	0003-MKNFE	No	Male	0	No	No	9	Yes	Yes	DSL
2	0004-TLHLJ	Yes	Male	0	No	No	4	Yes	No	Fiber optic
3	0011-IGKFF	Yes	Male	1	Yes	No	13	Yes	No	Fiber optic
4	0013-EXCHZ	Yes	Female	1	Yes	No	3	Yes	No	Fiber optic

5 rows × 40 columns

Mostramos Todas las columnas: **pd.set_option()**

```
# Muestra aleatoria #
pd.set_option('display.max_columns', None)
df.sample(5)
```

	customerid	churn	gender	seniorcitizen	partner	dependents	tenure	phoneservice	multiplelines	internetservi
6149	8410-BGQXN	No	Male	0	No	No	4	Yes	No	
6677	9178-JHUVJ	No	Male	0	Yes	Yes	24	Yes	Yes	
302	0436-TWFFZ	No	Female	0	No	No	67	Yes	Yes	D
6621	9102-IAYHT	Yes	Female	0	Yes	Yes	17	Yes	Yes	Fiber op
4467	6127-ISGTU	Yes	Female	0	Yes	No	16	Yes	Yes	Fiber op

DataFrame Aplanado ó Normalizado

```
# Mostrar Datos Alineados Muestra de 5.
df.sample(8).style.set_properties(**{'text-align': 'left'})
```

	customerid	churn	gender	seniorcitizen	partner	dependents	tenure	phoneservice	multiplelines	internetservi
2673	3717-LNXKW	No	Male	0	Yes	No	38	Yes	Yes	Fiber optic
5318	7266-GSSJX	No	Male	0	Yes	Yes	11	Yes	No	No
3235	4521-WFJAI	No	Male	0	No	No	56	Yes	Yes	No
5483	7516-GMHUV	No	Male	1	Yes	No	50	Yes	Yes	Fiber optic
3494	4824-GUCBY	No	Female	1	No	No	22	Yes	No	Fiber optic
782	1104-FEJAM	No	Male	0	Yes	Yes	28	Yes	Yes	DSL
3525	4855-SNKMY	Yes	Female	0	No	No	1	Yes	No	DSL
7166	9850-OWRHQ	Yes	Female	0	Yes	No	3	Yes	Yes	Fiber optic

```
# Analizamos los tipos de datos #
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7267 entries, 0 to 7266
Data columns (total 40 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   customerid                7267 non-null   object
 1   churn                     7267 non-null   object
 2   gender                    7267 non-null   object
 3   seniorcitizen             7267 non-null   int64
 4   partner                   7267 non-null   object
 5   dependents                7267 non-null   object
 6   tenure                    7267 non-null   int64
 7   phoneservice              7267 non-null   object
 8   multiplelines             7267 non-null   object
 9   internetservice           7267 non-null   object
 10  onlinesecurity            7267 non-null   object
 11  onlinebackup              7267 non-null   object
 12  deviceprotection          7267 non-null   object
 13  techsupport               7267 non-null   object
 14  streamingtv               7267 non-null   object
 15  streamingmovies           7267 non-null   object
 16  contract                  7267 non-null   object
 17  paperlessbilling          7267 non-null   object
 18  paymentmethod             7267 non-null   object
 19  charges_monthly           7267 non-null   float64
 20  charges_total             7267 non-null   object
 21  customer_gender           7267 non-null   object
 22  customer_seniorcitizen    7267 non-null   int64
 23  customer_partner          7267 non-null   object
 24  customer_dependents       7267 non-null   object
 25  customer_tenure           7267 non-null   int64
 26  phone_phoneservice        7267 non-null   object
 27  phone_multiplelines       7267 non-null   object
 28  internet_internetservice  7267 non-null   object
 29  internet_onlinesecurity   7267 non-null   object
 30  internet_onlinebackup     7267 non-null   object
 31  internet_deviceprotection 7267 non-null   object
 32  internet_techsupport      7267 non-null   object
 33  internet_streamingtv      7267 non-null   object
 34  internet_streamingmovies  7267 non-null   object
 35  account_contract          7267 non-null   object
 36  account_paperlessbilling  7267 non-null   object
 37  account_paymentmethod     7267 non-null   object
 38  account_charges_monthly   7267 non-null   float64
 39  account_charges_total     7267 non-null   object
dtypes: float64(2), int64(4), object(34)
memory usage: 2.2+ MB
```

```
# ¿El customerid es unico ?#
df['customerid'].nunique()==len(df)
# Si no es igual → hay duplicados.
```

```
True
```

```
# Tenemos Valores Nulos? #
# contamos lo Nulos #
df.isnull().sum().sort_values(ascending=False)
```

	0
customerid	0
churn	0
gender	0
seniorcitizen	0
partner	0
dependents	0
tenure	0
phoneservice	0
multiplelines	0
internetservice	0
onlinesecurity	0
onlinebackup	0
deviceprotection	0
techsupport	0
streamingtv	0
streamingmovies	0
contract	0
paperlessbilling	0
paymentmethod	0
charges_monthly	0
charges_total	0
customer_gender	0
customer_seniorcitizen	0
customer_partner	0
customer_dependents	0
customer_tenure	0
phone_phoneservice	0
phone_multiplelines	0
internet_internetservice	0
internet_onlinesecurity	0
internet_onlinebackup	0
internet_deviceprotection	0
internet_techsupport	0
internet_streamingtv	0
internet_streamingmovies	0
account_contract	0
account_paperlessbilling	0
account_paymentmethod	0
account_charges_monthly	0
account_charges_total	0

dtype: int64

```
# Hay columnas duplicadas ?#
df.columns
```

```
Index(['customerid', 'churn', 'gender', 'seniorcitizen', 'partner',
       'dependents', 'tenure', 'phoneservice', 'multiplelines',
       'internetservice', 'onlinesecurity', 'onlinebackup', 'deviceprotection',
       'techsupport', 'streamingtv', 'streamingmovies', 'contract',
       'paperlessbilling', 'paymentmethod', 'charges_monthly', 'charges_total',
       'customer_gender', 'customer_seniorcitizen', 'customer_partner',
       'customer_dependents', 'customer_tenure', 'phone_phoneservice',
       'phone_multiplelines', 'internet_internetservice',
       'internet_onlinesecurity', 'internet_onlinebackup',
       'internet_deviceprotection', 'internet_techsupport',
       'internet_streamingtv', 'internet_streamingmovies', 'account_contract',
       'account_paperlessbilling', 'account_paymentmethod',
       'account_charges_monthly', 'account_charges_total'],
      dtype='object')
```

Puliendo el DataSet: Elimina Columnas

```
# Como tenemos colunmnas duplicadas al expandir el json #
# ahora eliminamos columnas duplicadas #
cols_to_drop = [
    'customer_gender', 'customer_seniorcitizen', 'customer_partner',
    'customer_dependents', 'customer_tenure',
    'phone_phoneservice', 'phone_multiplelines',
    'internet_internetservice', 'internet_onlinesecurity',
    'internet_onlinebackup', 'internet_deviceprotection',
    'internet_techsupport', 'internet_streamingtv',
    'internet_streamingmovies',
    'account_contract', 'account_paperlessbilling',
    'account_paymentmethod', 'account_charges_monthly',
    'account_charges_total'
]
# Eliminando Columnas #
df = df.drop(columns=cols_to_drop)
```

```
# Vemos el Resultado #
df.shape
```

```
(7267, 21)
```

```
# Vemos la info del dataset #
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7267 entries, 0 to 7266
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerid        7267 non-null   object 
 1   churn             7267 non-null   object 
 2   gender            7267 non-null   object 
 3   seniorcitizen     7267 non-null   int64  
 4   partner           7267 non-null   object 
 5   dependents        7267 non-null   object 
 6   tenure            7267 non-null   int64  
 7   phoneservice      7267 non-null   object 
 8   multiplelines     7267 non-null   object 
 9   internetservice   7267 non-null   object 
 10  onlinesecurity    7267 non-null   object 
 11  onlinebackup      7267 non-null   object 
 12  deviceprotection  7267 non-null   object 
 13  techsupport       7267 non-null   object 
 14  streamingtv       7267 non-null   object 
 15  streamingmovies   7267 non-null   object 
 16  contract          7267 non-null   object 
 17  paperlessbilling  7267 non-null   object 
 18  paymentmethod     7267 non-null   object 
 19  charges_monthly   7267 non-null   float64
 20  charges_total     7267 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.2+ MB
```

¿Qué hace cada parte?

- `df.select_dtypes(include='object')`: Filtra el DataFrame para seleccionar únicamente las columnas que contienen texto (strings) o categorías.
- `print(f"\n{col}")`: Imprime el nombre de la columna actual para que sepas qué variable estás analizando.
- `df[col].value_counts()`: Cuenta cuántas veces aparece cada valor único en esa columna.

```
# Contabilizar Los Yes/No y Otros Valores#
# Vemos Rápidamente la composición de tus datos no numéricos= object#
for col in df.select_dtypes(include='object').columns:
    print(f"\n{col}")
    print(df[col].value_counts())
```

```
onlinebackup
onlinebackup
No                     3182
Yes                    2504
No internet service    1581
Name: count, dtype: int64

deviceprotection
deviceprotection
No                     3195
Yes                    2491
No internet service    1581
Name: count, dtype: int64

techsupport
techsupport
No                     3582
Yes                    2104
No internet service    1581
Name: count, dtype: int64

streamingtv
streamingtv
No                     2896
Yes                    2790
No internet service    1581
Name: count, dtype: int64

streamingmovies
streamingmovies
No                     2870
Yes                    2816
No internet service    1581
Name: count, dtype: int64

contract
contract
Month-to-month    4005
Two year          1743
One year          1519
Name: count, dtype: int64

paperlessbilling
paperlessbilling
Yes    4311
No     2956
Name: count, dtype: int64

paymentmethod
paymentmethod
Electronic check             2445
Mailed check                 1665
Bank transfer (automatic)    1589
Credit card (automatic)      1568
Name: count, dtype: int64

charges_total
```

Problemas Detectados de la Celda Anterior:

- 👉 Tienes 224 registros con churn vacío ("")
- charges_total tiene **11** strings vacíos

```
# la variable charges_total deberia ser numerica #
# Pero al hacer el info vemos que es object#
df['charges_total'].sample(5)
```

	charges_total
283	6526.65
6895	1191.2
3619	764.95
4830	19.7
7180	2070.75

dtype: object

Antes de cambiar charges_total a tipo Número
Debemos saber sí:

- ¿Son strings numéricos?
- ¿Hay espacios vacíos (" ")?
- ¿Hay valores raros?

Si conviertes charges_total a float sin revisar, puedes:

- Introducir NaN silenciosos
- Perder filas sin darte cuenta
- Generar sesgos

```
# Verificando charges_total #
df["charges_total"].apply(type).value_counts()
```

	count
charges_total	
<class 'str'>	7267

dtype: int64

Tenemos Valores vacios que deben ser Convertidos:

- Convertir strings vacíos a NaN

```
# Converitr sstring vacios a NaN #
# Nan = not A Number #
import numpy as np
# Remplazo #
df = df.replace("", np.nan)
```

```
# Verificamos Los cambios #
df.isnull().sum().sort_values(ascending=False)
```

	0
churn	224
customerid	0
gender	0
seniorcitizen	0
partner	0
dependents	0
tenure	0
phoneservice	0
multiplelines	0
internetservice	0
onlinesecurity	0
onlinebackup	0
deviceprotection	0
techsupport	0
streamingtv	0
streamingmovies	0
contract	0
paperlessbilling	0
paymentmethod	0
charges_monthly	0
charges_total	0

dtype: int64

Eliminando Valores Nan de Churn

Se eliminaron 224 registros debido a ausencia de variable objetivo (churn), ya que no es posible realizar imputación válida en variable dependiente para modelo supervisado.

```
# Ver renglones y Columnas #
df.shape
```

```
(7267, 21)
```

```
# Eliminamos Nan de churn #
df = df.dropna(subset=["churn"])
# Ver Renglones y Columnas #
df.shape
```

```
(7043, 21)
```

Convertir charges_total (object) a charges_total (float)

```
#Eliminamos los Espacios en blanco al inicio y final de charges_total #
df["charges_total"] = df["charges_total"].str.strip()
```

```
# Al quitar los espacios en blanco quedaran strings vacios #
# Hay que cambiar los string vacios por nan #
df["charges_total"] = df["charges_total"].replace("", np.nan)
```

```
# Verificamos los NAN que aparecen Ahora #
df["charges_total"].isnull().sum()
```

```
np.int64(11)
```

```
# Eliminamos Nan de charges_total #
df = df.dropna(subset=["charges_total"])
# Verificamos #
df.shape
```

```
(7032, 21)
```

Se eliminaron 11 registros con charges_total inválido (strings vacíos/espacios).

Representaban <0.2 % del dataset, por lo que no afectan la representatividad estadística.

```
# El cambio de Object a Float
df["charges_total"] = df["charges_total"].astype(float)
# Verificamos #
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7032 entries, 0 to 7266
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerid        7032 non-null   object 
 1   churn             7032 non-null   object 
 2   gender            7032 non-null   object 
 3   seniorcitizen     7032 non-null   int64  
 4   partner           7032 non-null   object 
 5   dependents        7032 non-null   object 
 6   tenure            7032 non-null   int64  
 7   phoneservice      7032 non-null   object 
 8   multiplelines     7032 non-null   object 
 9   internetservice   7032 non-null   object 
 10  onlinesecurity    7032 non-null   object 
 11  onlinebackup      7032 non-null   object 
 12  deviceprotection  7032 non-null   object 
 13  techsupport       7032 non-null   object 
 14  streamingtv       7032 non-null   object 
 15  streamingmovies   7032 non-null   object 
 16  contract          7032 non-null   object 
 17  paperlessbilling  7032 non-null   object 
 18  paymentmethod     7032 non-null   object 
 19  charges_monthly   7032 non-null   float64
 20  charges_total     7032 non-null   float64
dtypes: float64(2), int64(2), object(17)
memory usage: 1.2+ MB
```

Lo que Tenemos Ahora:

- ✔ Dataset estructuralmente limpio
- ✔ Tipos correctos
- ✔ Target válido
- ✔ Sin nulos
- ✔ Sin duplicados
- ✔ Listo para análisis real

Normalizar texto (minúsculas)

1. Selección de Columnas: select_dtypes(include="object")

Identifica y selecciona únicamente las columnas que contienen cadenas de texto (strings).

Ignora las columnas numéricas para evitar errores de ejecución.

2. Eliminación de Espacios: .str.strip()

Elimina los espacios en blanco invisibles al inicio y al final de cada palabra (ej. transforma " Aeromexico " en "Aeromexico").

Esto es vital **porque, para Python, "México" y "México "** son categorías diferentes.

3. Estandarización de Caja: .str.lower()

Convierte todo el texto a minúsculas (ej. transforma "VUELO" en "vuelo").

Asegura que el modelo no se confunda con variaciones de escritura (ej. "Iberia", "IBERIA" e "iberia" ahora serán la misma etiqueta).

```
#Todas las Columnas a minusculas#
for col in df.select_dtypes(include="object").columns:
    df[col] = df[col].str.strip().str.lower()
```

```
# Verificamos para contrac#
df["contract"].value_counts()
```

contract	count
month-to-month	3875
two year	1685
one year	1472

dtype: int64

```
# Verificamos para multiplelines#
df["multiplelines"].value_counts()
```

multiplelines	count
no	3385
yes	2967
no phone service	680

dtype: int64

```
# Verificamos para onlinesecurity#
df["onlinesecurity"].value_counts()
```

onlinesecurity	count
no	3497
yes	2015
no internet service	1520

dtype: int64

Simplificación inteligente de categorías

tus variables tienes patrones como:

multiplelines:

- yes
- no
- no phone service

onlinesecurity:

- yes
- no
- no internet service

Esto es redundante porque:

- Si internetservice == no entonces automáticamente:
- onlinesecurity = no internet service
- onlinebackup = no internet service, etc.

👉 Eso introduce ruido y complejidad innecesaria.

🎯 Estrategia profesional

Convertiremos todas las columnas tipo servicio en binarias:

- yes → 1
- no → 0
- no internet service → 0
- no phone service → 0

Porque conceptualmente:

Si no tiene el servicio base, no puede tener el servicio adicional. **Esto simplifica muchísimo el modelo.**

```
# Ejemplo del dataSet Actual #
df.sample(3)
```

	customerid	churn	gender	seniorcitizen	partner	dependents	tenure	phoneservice	multiplelines	internetservi
5642	7706-ylmqa	no	female	0	no	no	70	yes	no	
1315	1897-rcfum	no	female	0	yes	yes	39	yes	yes	
918	1299-aurja	no	female	0	yes	yes	70	yes	yes	

```
#  Creamos un arreglo con la columnas dicotomicas #
service_cols = [
    "phoneservice", "multiplelines",
    "onlinesecurity", "onlinebackup",
    "deviceprotection", "techsupport",
    "streamingtv", "streamingmovies"
]
# usamos ese Arreglo para asignar los valores de 0 y 1 #
df[service_cols] = df[service_cols].replace({
    "yes": 1,
    "no": 0,
    "no internet service": 0,
    "no phone service": 0
})
```

```
/tmp/ipython-input-674330640.py:9: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed
  df[service_cols] = df[service_cols].replace({
```

```
# Verificamos 4 renglones random de service_cols #
df[service_cols].sample(4)
```

	phoneservice	multiplelines	onlinesecurity	onlinebackup	deviceprotection	techsupport	streamingtv	streamingr
729	1	1	0	0	0	0	0	
4611	1	0	0	1	0	0	1	

```
# Verificacion inmediata de los cambios En tipo #
df[service_cols].dtypes
```

	0
phoneservice	int64
multiplelines	int64
onlinesecurity	int64
onlinebackup	int64
deviceprotection	int64
techsupport	int64
streamingtv	int64
streamingmovies	int64

dtype: object

La instrucción df.loc[df["internetservice"] == "no", service_cols]

Le pide a Python que busque todas las filas donde el cliente no tiene contratado el servicio de internet (df["**internetservice"] == "no**")
y que, para esas filas en particular, te muestre únicamente las columnas listadas en service_cols.

```
# Verificacion conceptual #
df.loc[df["internetservice"] == "no", service_cols]
```

	phoneservice	multiplelines	onlinesecurity	onlinebackup	deviceprotection	techsupport	streamingtv	streamingr
20	1	1	0	0	0	0	0	
23	1	0	0	0	0	0	0	
24	1	0	0	0	0	0	0	
27	1	0	0	0	0	0	0	
28	1	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	
7250	1	0	0	0	0	0	0	
7252	1	0	0	0	0	0	0	
7256	1	0	0	0	0	0	0	
7257	1	0	0	0	0	0	0	
7261	1	0	0	0	0	0	0	

1520 rows × 8 columns

Binarización de variables dicotómicas

Variables a transformar:

- churn → target

- partner
- dependents
- paperlessbilling
- gender (opcional, pero recomendable)

Decisión:

- yes → 1
- no → 0
- female → 1, male → 0 (**convención simple**)

```
# Generamos un diccionario y un Arreglo #
binary_map = {
    "yes": 1,
    "no": 0,
    "female": 1,
    "male": 0
}

binary_cols = [
    "churn", "partner", "dependents",
    "paperlessbilling", "gender"
]
```

```
# Hacemos el Reemplazo de los valores string -> int #
# Reemplazo de yes por 1 y no por 0 #
df[binary_cols] = df[binary_cols].replace(binary_map)
# Definimos el tipo = entero #
df[binary_cols] = df[binary_cols].astype(int)
```

```
/tmp/ipython-input-1689631311.py:3: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed
  df[binary_cols] = df[binary_cols].replace(binary_map)
```

```
# Verificacion de tipo de Dato #
df[binary_cols].dtypes
```

	0
churn	int64
partner	int64
dependents	int64
paperlessbilling	int64
gender	int64

dtype: object

```
# Conteo de los valores en las binary_cols #
df[binary_cols].value_counts()
```

churn	partner	dependents	paperlessbilling	gender	count
0	0	0	1	1	591
				0	568
			0	0	521
				1	477
1	0	0	1	1	438
				0	404
0	1	1	0	0	387
		0	1	0	382
		1	0	1	381
		0	1	1	375
		1	1	1	365
				0	358
		0	0	1	243
				0	233
1	1	0	1	0	192
	0	0	0	1	149
	1	0	1	1	148
	0	0	0	0	132
0	0	1	0	0	96
1	1	1	1	1	90
				0	77
0	0	1	1	0	74
			0	1	57
			1	1	55
1	1	1	0	1	42
		0	0	0	41
		1	0	0	40
		0	0	1	39
	0	1	1	0	28
				1	23
			0	0	16
				1	10

dtype: int64

Tratamiento de Variables Politómicas

Las variables que quedan categóricas son:

- internetservice (sin orden especifico)
- contract (Ordinal)
- paymentmethod

Estas **NO** deben convertirse en 0/1 directamente.

```
# Verificamos Valores unicos para Variables Politomicas  #
for col in ["internetservice", "contract", "paymentmethod"]:
    print("\n", col)
    print(df[col].value_counts())
```

```
 internetservice
internetservice
fiber optic    3096
dsl            2416
no             1520
Name: count, dtype: int64

 contract
contract
month-to-month    3875
two year          1685
one year          1472
Name: count, dtype: int64

 paymentmethod
paymentmethod
electronic check             2365
mailed check                 1604
bank transfer (automatic)    1542
credit card (automatic)      1521
Name: count, dtype: int64
```

Transformación con pd.get_dummies()

La función **get_dummies se encarga de realizar el One-Hot Encoding.**

- Su objetivo es convertir variables categóricas (texto) en variables numéricas (0 y 1)
- ya que los modelos matemáticos como el Random Forest no pueden realizar cálculos directamente sobre palabras como "Fiber optic" o "Electronic check".
- columns=["internetservice", ...]: Le indicas específicamente qué columnas quieres transformar. Las columnas que no menciones (como las numéricas) se quedarán tal cual están.
- drop_first=True: Esta es una técnica para evitar la multicolinealidad (trampa de la variable ficticia). Elimina la primera columna generada de cada categoría.

```
# Hacemos la transformacion #
df = pd.get_dummies(
    df,
    columns=["internetservice", "contract", "paymentmethod"],
    drop_first=True
)
```

```
# Verificamos #
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7032 entries, 0 to 7266
Data columns (total 25 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   customerid                            7032 non-null   object
 1   churn                                 7032 non-null   int64
 2   gender                                7032 non-null   int64
 3   seniorcitizen                         7032 non-null   int64
 4   partner                               7032 non-null   int64
 5   dependents                            7032 non-null   int64
 6   tenure                                7032 non-null   int64
 7   phoneservice                          7032 non-null   int64
 8   multiplelines                         7032 non-null   int64
 9   onlinesecurity                        7032 non-null   int64
 10  onlinebackup                          7032 non-null   int64
 11  deviceprotection                      7032 non-null   int64
 12  techsupport                           7032 non-null   int64
 13  streamingtv                           7032 non-null   int64
```

```
 14  streamingmovies                          7032 non-null   int64
 15  paperlessbilling                         7032 non-null   int64
 16  charges_monthly                          7032 non-null   float64
 17  charges_total                            7032 non-null   float64
 18  internetservice_fiber optic              7032 non-null   bool
 19  internetservice_no                       7032 non-null   bool
 20  contract_one year                        7032 non-null   bool
 21  contract_two year                        7032 non-null   bool
 22  paymentmethod_credit card (automatic)    7032 non-null   bool
 23  paymentmethod_electronic check           7032 non-null   bool
 24  paymentmethod_mailed check               7032 non-null   bool
dtypes: bool(7), float64(2), int64(15), object(1)
memory usage: 1.1+ MB
```

```
# Churn Promedio #
df['churn'].mean()
```

```
np.float64(0.26578498293515357)
```

👉 **26.6%** de los clientes abandonan.

Eso significa:

De cada 100 clientes → 27 se van

Es un churn relativamente alto

Problema de negocio serio:

💡 **En telecom, >20% ya es preocupante.**

```
# Como se distribuye el Tenure "tenure: meses de contrato del cliente #
df['tenure'].describe()
```

	tenure
count	7032.000000
mean	32.421786
std	24.545260
min	1.000000
25%	9.000000
50%	29.000000
75%	55.000000
max	72.000000

dtype: float64

📊 2️⃣ Distribución de Tenure

Resumen:

- Media: 32 meses
- Mediana: 29 meses
- Q1: 9 meses
- Q3: 55 meses
- Máximo: 72 meses

🔎 **Observaciones importantes:**

- El 25% de clientes tiene menos de 9 meses
- Hay mucha dispersión (std = 24)

```
#¿El churn se concentra en clientes nuevos ?
# Churn: si el cliente dejó o no la empresa
df.groupby("churn")['tenure'].mean()
```

churn	tenure
0	37.650010
1	17.979133

dtype: float64

Los clientes que se van: → Tienen en promedio la mitad de antigüedad.

Clientes nuevos tienen muchísimo mayor riesgo.

Esto sugiere:

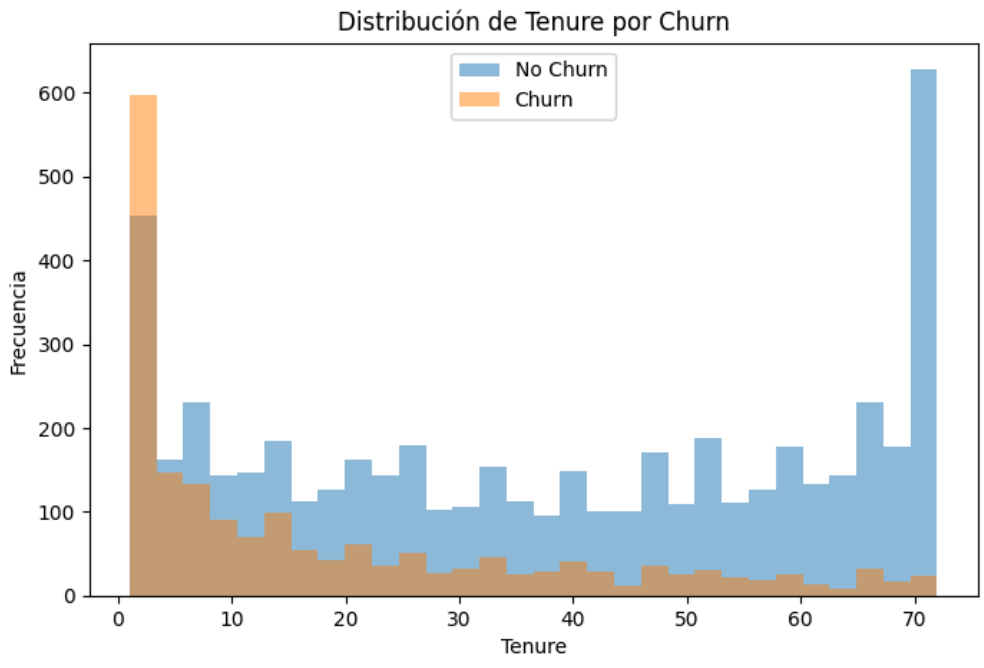
- Problema en onboarding
- Expectativas no cumplidas
- Mala experiencia inicial
- Ofertas introductorias que luego decepcionan.

A menor tenure, mayor probabilidad de churn.

3. Creación de Visualizaciones

```
# Vamos a crear visualizaciones #
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
plt.hist(df[df["churn"]==0]["tenure"], bins=30, alpha=0.5, label="No Churn")
plt.hist(df[df["churn"]==1]["tenure"], bins=30, alpha=0.5, label="Churn")
plt.legend()
plt.xlabel("Tenure")
plt.ylabel("Frecuencia")
plt.title("Distribución de Tenure por Churn")
plt.show()
```



📊 Lo que muestra el gráfico 🔴 Churn (naranja)

- **Altísima concentración en los primeros meses (0–10)**
- Luego cae drásticamente
- Después de **~30 meses casi desaparece.**

🔵 No Churn (azul)

- Mucho más distribuido
- Fuerte acumulación en clientes de **60–72 meses**

Análisis de Correlación: Tenure vs. Churn

En tu análisis, el coeficiente de -0.354 nos da una pista vital sobre el comportamiento de tus clientes.

¿Qué significa el valor -0.354?

- **Relación Inversa:** Significa que a medida que aumenta el Tenure(**la antigüedad del cliente**), la probabilidad de Churn (**abandono**) disminuye.
- **Fuerza de la relación:** Un valor de **0.35** se considera una correlación moderada. No es una regla absoluta, pero sí una tendencia clara en los datos.

```
# Dado lo Anterior podemos calcular la correlacion de ambas variables #
df[["tenure", "churn"]].corr()
```

	tenure	churn
tenure	1.000000	-0.354049
churn	-0.354049	1.000000

4. Realizar un Análisis Exploratorio de Datos (EDA)

👉 Tasa de churn por tramo de antigüedad.

```
# Veremos la tabla de churn por antiguedad #
df["tenure_group"] = pd.cut(
    df["tenure"],
    bins=[0,12,24,36,48,60,72],
    labels=["0-12","12-24","24-36","36-48","48-60","60-72"]
)

df.groupby("tenure_group")["churn"].mean()
```

```
/tmp/ipython-input-2611057453.py:8: FutureWarning: The default of observed=False is deprecated and will be changed to
  df.groupby("tenure_group")["churn"].mean()
```

tenure_group	churn
0-12	0.476782
12-24	0.287109
24-36	0.216346
36-48	0.190289
48-60	0.144231
60-72	0.066098

dtype: float64

"El churn no es generalizado.
Está hiperconcentrado en clientes con menos de 12 meses,
donde alcanza casi el **48%**. Después del primer año, el riesgo cae drásticamente."

Contrato y Churn

- ¿El tipo de contrato reduce o aumenta el churn?

```
# Contrato por 1 año #
df.groupby("contract_one year")["churn"].mean()
```

contract_one year	churn
False	0.306295
True	0.112772

dtype: float64

📊 1️⃣ Contrato de 1 año vs Churn

Contract 1 year	Churn
False	30.6%
True	11.3%

👉 **Tener contrato anual reduce el churn casi 3 veces.**

```
# Contrato por 2 años #
df.groupby("contract_two year")["churn"].mean()
```

contract_two year	churn
False	0.340565
True	0.028487

dtype: float64

📊 2️⃣ Contrato de 2 años vs Churn

Contract 2 years	Churn
False	34.0%
True	2.8% 🚀

Esto es brutal.

👉 Clientes con contrato de 2 años casi no se van.

```
# Contrato de un año  vs meses de contratacion del servicio #
df.groupby("contract_one year")["tenure"].mean()
```

contract_one year	tenure
False	29.866547
True	42.073370

dtype: float64

```
# Tipo de Contrato por 2 años vs meses de contratacion #
df.groupby("contract_two year")["tenure"].mean()
```

contract_two year	tenure
False	24.653825
True	57.071810

dtype: float64

🎯 Interpretación estratégica fuerte

Tenemos dos fuerzas actuando:

- 📈 A mayor tenure ⟶ menor churn
- 📜 A mayor duración de contrato ⟶ muchísimo menor churn

Pero el contrato reduce churn de manera más agresiva.

IMPORTANTE:

"El churn cae de 34% a solo 2.8% en clientes con contratos de 2 años.
Esto sugiere que el tipo de contrato es uno de los principales mecanismos de retención."

👉 Tasa de churn en month-to-month

Recuerda que es la categoría base (cuando one_year = 0 y two_year = 0).

```
#Tasa de churn en mes en mes#
df[
    (df["contract_one year"]==0) &
    (df["contract_two year"]==0)
]["churn"].mean()
```

```
np.float64(0.4270967741935484)
```

🔥 Month-to-Month Churn = 42.7%

Eso confirma todo:

Tipo contrato	Churn
Month-to-month	42.7% 🚨
1 año	11.3%
2 años	2.8%

📌 El verdadero problema está en contratos mensuales.

```
#Charges Monthly VS Churn #
df.groupby("churn")["charges_monthly"].mean()
```

churn	charges_monthly
0	61.307408
1	74.441332

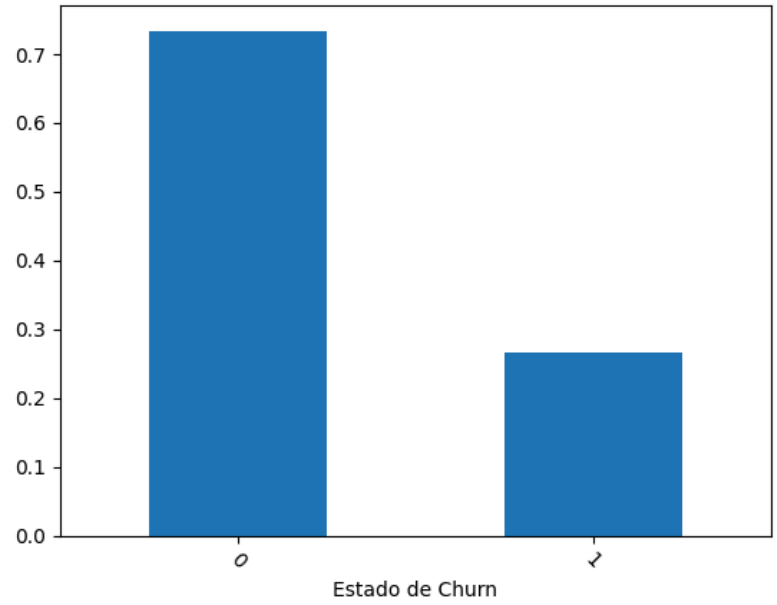
dtype: float64

```
# Descripcion Estadistica #
df.describe()
```

	churn	gender	seniorcitizen	partner	dependents	tenure	phoneservice	multiplelines	onlir
count	7032.000000	7032.000000	7032.000000	7032.000000	7032.000000	7032.000000	7032.000000	7032.000000	7(
mean	0.265785	0.495307	0.162400	0.482509	0.298493	32.421786	0.903299	0.421928	
std	0.441782	0.500014	0.368844	0.499729	0.457629	24.545260	0.295571	0.493902	
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	0.000000	9.000000	1.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	0.000000	29.000000	1.000000	0.000000	
75%	1.000000	1.000000	0.000000	1.000000	1.000000	55.000000	1.000000	1.000000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	72.000000	1.000000	1.000000	

```
# Grafico de Distribucion de churn #
df["churn"].value_counts(normalize=True).plot(kind="bar",rot=-45)
plt.xlabel("Estado de Churn")
```

```
Text(0.5, 0, 'Estado de Churn')
```



📌 1️⃣ Churn por Género

```
# Promedio de genero con churn#
df.groupby("gender")["churn"].mean()
```

gender	churn
0	0.262046
1	0.269595

dtype: float64

🧠 Interpretación

- Diferencia mínima
- No hay sesgo claro por género

📌 **Conclusión:**

El género no es un factor determinante en la evasión.

📌 2️⃣ Churn por Senior Citizen

```
# Promedio de seniorcitizen vs churn#
df.groupby("seniorcitizen")["churn"].mean()
```

seniorcitizen	churn
0	0.236503
1	0.416813

dtype: float64

🧠 Interpretación

- Los adultos mayores casi duplican el churn
- Es una variable muy relevante

📌 Insight fuerte:

Los clientes senior son un grupo de alto riesgo.

📌 3️⃣ Churn por Partner

```
# partner vs churn #
df.groupby("partner")["churn"].mean()
```

partner	churn
0	0.329761
1	0.197171

dtype: float64

🧠 Interpretación

- Tener pareja reduce fuertemente el churn
- Estabilidad familiar → mayor permanencia

📌 Insight:

Clientes con pareja son significativamente más estables.

📌 5️⃣ Churn por Método de Pago

```
# Tipo de pago  electronico y churn #
df.groupby("paymentmethod_electronic check")["churn"].mean()
```

paymentmethod_electronic check	churn
False	0.170988
True	0.452854

dtype: float64

```
# Pago por correo vs desercion #
df.groupby("paymentmethod_mailed check")["churn"].mean()
```

paymentmethod_mailed check	churn
False	0.287583
True	0.192020

dtype: float64

```
# PAgo con tarjeta vs desercion #
df.groupby("paymentmethod_credit card (automatic)")["churn"].mean()
```

paymentmethod_credit card (automatic)	churn
False	0.297042
True	0.152531

dtype: float64

🧠 Interpretación global

- Electronic check es el método más riesgoso
- Pagos automáticos reducen churn
- Fricción en pago = abandono

📌 Insight estratégico:

Automatizar pagos reduce significativamente la evasión.

📌 6️⃣ Churn por Internet Service

```
# fibra Optica vs desercion #
df.groupby("internetservice_fiber optic")["churn"].mean()
```

internetservice_fiber optic	churn
False	0.145325
True	0.418928

dtype: float64

```
# no servicio de internet vs desercicon #
df.groupby("internetservice_no")["churn"].mean()
```

internetservice_no	churn
False	0.318578
True	0.074342

dtype: float64

🧠 Interpretación

- Fiber tiene churn altísimo
- Clientes sin internet casi no se van
- Probable problema de:

- Precio
- Calidad
- Expectativas

5. Creación de un Informe con insights

🧩 PERFIL COMPLETO DE ALTO RIESGO (**RESUMEN**)

Con todo el análisis, el perfil más riesgoso es:

🔴 Cliente nuevo (0–12 meses)

🔴 Contrato month-to-month

🔴 Cargo mensual alto

🔴 Internet Fiber

🔴 Pago con electronic check

🔴 Senior citizen

🔴 Sin partner

5.1 Introducción

El presente análisis tiene como objetivo estudiar el fenómeno de evasión de clientes (Churn) en una empresa de telecomunicaciones.

El churn representa la cancelación del servicio por parte de los clientes, lo que impacta directamente en los ingresos y en la estabilidad financiera de la empresa.

Comprender qué factores están asociados a la evasión permite diseñar estrategias de retención más efectivas, optimizar recursos y mejorar la experiencia del cliente.

El objetivo principal de este estudio es:

Identificar patrones asociados al churn.

Detectar perfiles de alto riesgo.

Proponer recomendaciones estratégicas basadas en datos.

5.2 Limpieza y Tratamiento de Datos

Durante esta etapa se realizaron los siguientes procesos:

✓ Eliminación de valores nulos

- Se eliminaron registros con valores faltantes en la variable objetivo churn.

✓ Conversión de tipos de datos

- La variable charges_total fue convertida de texto a tipo numérico.
- Variables categóricas binarias fueron transformadas a valores 0 y 1.

✓ Codificación de variables categóricas

Se aplicó One-Hot Encoding a:

- Tipo de contrato
- Método de pago
- Servicio de internet

Se utilizó drop_first=True para evitar multicolinealidad.

✓ Estandarización de nombres:

Se normalizaron los nombres de columnas a minúsculas para mantener consistencia.

El dataset final quedó completamente limpio, estructurado y listo para análisis.

5.3 Análisis Exploratorio de Datos (EDA)

📌 1 Distribución General del Churn

La tasa general de evasión es:

- 26.6%

Es decir, aproximadamente 1 de cada 4 clientes cancela el servicio.

📌 2 Antigüedad (Tenure) y Churn

Se identificó una relación negativa entre antigüedad y churn:

- Correlación: -0.35
- Clientes que se quedan: 37.6 meses promedio
- Clientes que se van: 17.9 meses promedio

Tasa de churn por tramo:

Tenure	Churn
0–12 meses	47.7%
60–72 meses	6.6%

🔎 Conclusión: El riesgo es crítico durante el primer año.

📌 3 Tipo de Contrato

Tipo	Churn
Month-to-month	42.7%
1 año	11.3%
2 años	2.8%

El contrato es uno de los factores más determinantes.

📌 4 Cargo Mensual

- Clientes que abandonan pagan en promedio: **$74.44**
- Clientes que permanecen pagan: **$61.30**

El churn se concentra en clientes con mayor facturación mensual.

📌 5 Método de Pago

Método	Churn
Electronic check	45.3%
Credit card automático	15.3%
Bank transfer automático	Bajo

Los pagos automáticos reducen significativamente la evasión.

📌 6 Servicio de Internet

Servicio	Churn
Fiber optic	41.9%
Sin internet	7.4%

El servicio premium presenta mayor evasión.

📌 7 Factores Demográficos

- Senior citizens: 41.7% churn
- Sin partner: 33%
- Con partner: 19.7%

Género: no presenta diferencias relevantes

5.4 Perfil de Alto Riesgo (Desglose Analítico)

Se identificó el siguiente perfil con alta probabilidad de churn:

Cliente nuevo + contrato mensual + cargo alto + fibra óptica + pago electronic check + senior citizen + sin partner.

Ahora lo desglosamos:

🔴 **Cliente nuevo**

Mayor incertidumbre y menor vínculo con la empresa.

🔴 **Contrato mensual**

No existe compromiso de permanencia.

🔴 **Cargo mensual alto**

Mayor sensibilidad al precio y percepción de bajo valor.

🔴 **Fiber optic**

Servicio premium con expectativas altas. Posible insatisfacción.

🔴 **Electronic check**

Fricción en pago y menor automatización.

🔴 **Senior citizen**

Posible dificultad tecnológica o menor tolerancia a problemas.

🔴 **Sin partner**

Menor estabilidad del hogar asociada al servicio.

Este perfil no surge al azar; es la combinación de múltiples factores que estadísticamente aumentan el riesgo.

5.5 Conclusiones e Insights

1. El churn está altamente concentrado en los primeros 12 meses.
2. El tipo de contrato es el factor más protector.
3. Los servicios premium presentan mayor evasión.
4. Los métodos de pago automáticos reducen churn.
5. Clientes senior representan un grupo vulnerable.
6. No existe diferencia significativa por género.

El churn no es aleatorio; sigue patrones claros y accionables.

5.6 Recomendaciones Estratégicas