

✓ INFORMACIÓN GENERAL

UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO
INGENIERÍA INFORMÁTICA Y DE SISTEMAS
APRENDIZAJE NO SUPERVISADO

✓ Preprocesamiento de Datos en Machine Learning

✓ 1. Contexto

Una empresa quiere predecir el abandono de clientes para priorizar campañas de retención.
Se cuenta con un dataset sintético que emula datos reales y contiene imperfecciones típicas (faltantes, outliers, duplicados y desbalance de clases).

- Datos disponibles**
- Conjunto de registros de clientes con las siguientes variables:
- edad (numérico): edad del cliente (años).
 - ingreso (numérico): ingreso anual estimado (sesgado, con posibles outliers).
 - canal (categórico): canal de adquisición (web, store, phone).
 - ciudad (categórico): ciudad principal (Cusco, Lima, Arequipa).
 - fecha_alta (fecha): fecha de alta del cliente.
 - abandono (binario; target): 1 si abandonó, 0 si se retuvo.

Objetivo

Desarrollar un pipeline de preprocesamiento que deje los datos listos para modelar y entrenar un modelo base de clasificación del abandono, reportando métricas y hallazgos clave.

✓ Marco conceptual

- 1.- EDA (Exploratory Data Analysis)**
- Diagnosticar: entender la estructura, calidad y distribución de los datos (incluyendo la variable target).
 - Identificar problemas y potenciales mejoras.
 - Es exploratorio: no cambia los datos, solo los describe y revela.
- 2.- Limpieza y preparación de datos**
- Corregir errores, valores faltantes, duplicados, outliers, codificación de variables, escalado, etc.
 - Incluye garantizar que los datos tengan un formato y calidad adecuados para el modelado.
 - Aquí también entra la preparación de conjuntos (train/valid/test) y aspectos como balanceo de clases en train.
- 3.- Ingeniería de características (Feature Engineering)**
- Crear nuevas variables, transformar o seleccionar las más útiles.
 - Aumentar la capacidad del modelo de aprender patrones significativos.
 - Implica creatividad y conocimiento del dominio.

- En resumen:**
- El EDA es un análisis exploratorio (pregunta: ¿cómo están mis datos?).
 - La limpieza/preparación corrige y estandariza (pregunta: ¿mis datos son aptos para modelar?).
 - La ingeniería de características crea valor extra (pregunta: ¿puedo representar mis datos de una forma que mejore el aprendizaje?).

Conceptos considerados en cada fase

- 1. EDA – Exploratory Data Analysis**
- Estructura y tipos
 - Valores faltantes
 - Distribuciones y estadísticas básicas
 - Outliers
 - Relaciones entre variables
 - Análisis del target
 - Patrones temporales
- 2. Limpieza / Preparación**
- Duplicados e inconsistencias
 - Conversión de tipos
 - Imputación de valores faltantes
 - Tratamiento de outliers
 - Escalado / normalización
 - Codificación de categóricas
 - División Train/Valid/Test (sin fuga de datos)
- 3. Ingeniería de características**
- Derivadas de fecha/tiempo
 - Transformaciones y binning
 - Interacciones y cruces
 - Codificación avanzada (target encoding – precauciones)
 - Reducción de dimensionalidad (PCA)
 - Pipeline integrado con ColumnTransformer + evaluación
 - Balanceo - SMOTE (sobremuestreo de la clase minoritaria)

✓ Configuración

✓ Instalación de librerías

```
1 # Importar librerías
2 import numpy as np # Operaciones numéricas eficientes y arrays multidimensionales
3 import pandas as pd # Manipulación y análisis de datos estructurados (DataFrames, Series)
4 import matplotlib.pyplot as plt # Visualización de datos y creación de gráficos
5
6 from datetime import datetime, timedelta # Manipulación de fechas y tiempos
7
8 from sklearn.model_selection import train_test_split # División de datos en conjuntos de entrenamiento y prueba
9 from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxScaler, RobustScaler, FunctionTransformer # Preprocesamiento y transformación de características
```


```
10 from sklearn.compose import ColumnTransformer # Transformación de columnas específicas en pipelines
11 from sklearn.pipeline import Pipeline # Creación de flujos de procesamiento de datos secuenciales
12 from sklearn.linear_model import LogisticRegression # Modelo de regresión logística para clasificación
13 from sklearn.metrics import classification_report, roc_auc_score, roc_curve # Métricas de evaluación de modelos de clasificación
14 from sklearn.impute import SimpleImputer # Imputación de valores faltantes en los datos
15
16
```



```
1 # semilla para garantizar reproductibilidad de resultados
2 np.random.seed(42)
```

Generación de data set

Se generará un dataset **sintético** con problemas intencionales (faltantes, outliers y duplicados) para ilustrar el preprocesamiento.

```
1 # Generación de datos sintéticos
2 n = 1500 # cantidad de datos
3 edad = np.random.normal(38, 10, n).round(0) # Distribución normal (μ=38, σ=10)
4 ingreso = np.random.lognormal(mean=9.5, sigma=0.55, size=n) # Distribución log-normal (sesgada positivamente)
5 canal = np.random.choice(['web','store','phone'], size=n, p=[0.6, 0.3, 0.1]) # Distribución categórica multinomial con probabilidades específicas
6 ciudad = np.random.choice(['Cusco','Lima','Arequipa'], size=n, p=[0.2, 0.6, 0.2]) # Distribución categórica multinomial con sesgo hacia Lima
7
8 start = datetime(2019,1,1)
9 fecha_altas = [start + timedelta(days=int(x)) for x in np.random.randint(0, 2400, n)] # Distribución uniforme discreta para fechas (≈6.5 años)
10
11 # Target binario desbalanceado: ~15% positivos
12 abandono = (np.random.rand(n) < 0.15).astype(int) # Distribución Bernoulli (p=0.15)
13
14 # creación del dataframe
15 df = pd.DataFrame({
16     'edad': edad,
17     'ingreso': ingreso,
18     'canal': canal,
19     'ciudad': ciudad,
20     'fecha_alta': fecha_altas,
21     'abandono': abandono
22 })
23
24 # Problemas intencionales
25 df.loc[np.random.choice(df.index, 60, replace=False), 'edad'] = np.nan # Introduce 60 valores faltantes aleatorios
26 df.loc[np.random.choice(df.index, 40, replace=False), 'canal'] = None # Introduce 40 valores faltantes aleatorios
27 df.loc[np.random.choice(df.index, 30, replace=False), 'ingreso'] *= 8 # Crea 30 outliers extremos multiplicando ingresos por 8
28 df = pd.concat([df, df.iloc[[5]]], ignore_index=True) # Duplica intencionalmente la sexta fila del DataFrame
29
30 # visualizar primeros datos
31 df.head()
```



	edad	ingreso	canal	ciudad	fecha_alta	abandono	
0	43.0	20498.327817	store	Lima	2024-01-08	0	
1	37.0	9865.981000	phone	Arequipa	2021-09-18	0	
2	44.0	8518.457971	web	Lima	2024-11-16	0	
3	53.0	13334.954819	phone	Lima	2025-06-22	0	
4	36.0	12165.975701	web	Lima	2024-12-10	0	


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



2. EDA – Exploratory Data Analysis

Objetivo: diagnosticar estructura, calidad y patrones. Aún **no modificamos** los datos; solo observamos.

2.1 Estructura y tipos

```
1 display(df.head())
2 print("\nDimensiones:", df.shape)
3 print("\nInformación general:")
4 print(df.info())
```



	edad	ingreso	canal	ciudad	fecha_alta	abandono	
0	43.0	20498.327817	store	Lima	2024-01-08	0	
1	37.0	9865.981000	phone	Arequipa	2021-09-18	0	
2	44.0	8518.457971	web	Lima	2024-11-16	0	
3	53.0	13334.954819	phone	Lima	2025-06-22	0	
4	36.0	12165.975701	web	Lima	2024-12-10	0	

Dimensiones: (1501, 6)

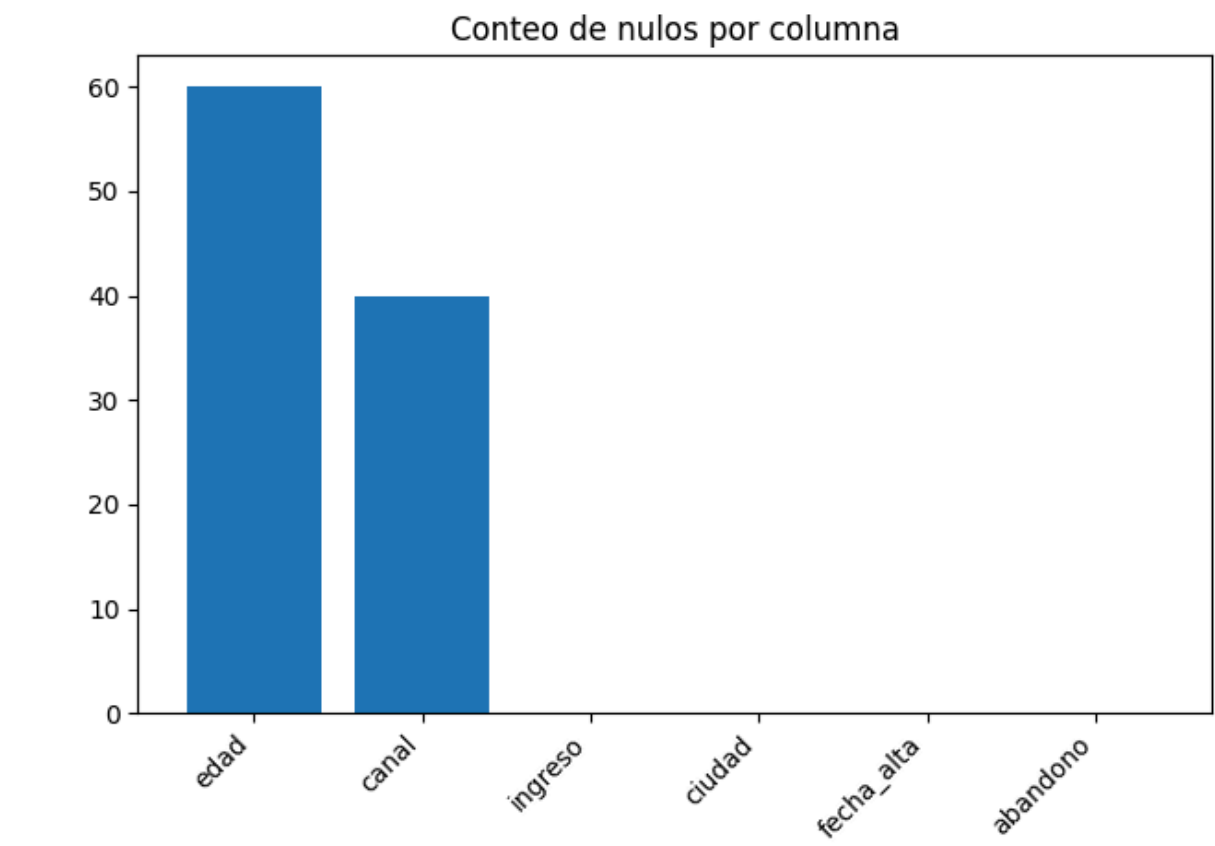
Información general:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1501 entries, 0 to 1500
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  -
0   edad        1441 non-null   float64
1   ingreso     1501 non-null   float64
2   canal       1461 non-null   object
3   ciudad      1501 non-null   object
4   fecha_alta  1501 non-null   datetime64[ns]
5   abandono    1501 non-null   int64
dtypes: datetime64[ns](1), float64(2), int64(1), object(2)
memory usage: 70.5+ KB
None
```

2.2 Valores faltantes

```
1 # Determinar valores faltantes y contar valores nulos por columna
2 missing = df.isna().sum().sort_values(ascending=False)
3 display(missing.to_frame('n_missing').T)
4
5 # Gráfica del conteo de nulos por columna
6 plt.figure()
7 plt.bar(missing.index, missing.values)
8 plt.xticks(rotation=45, ha='right')
9 plt.title('Conteo de nulos por columna')
10 plt.tight_layout()
11 plt.show()
```

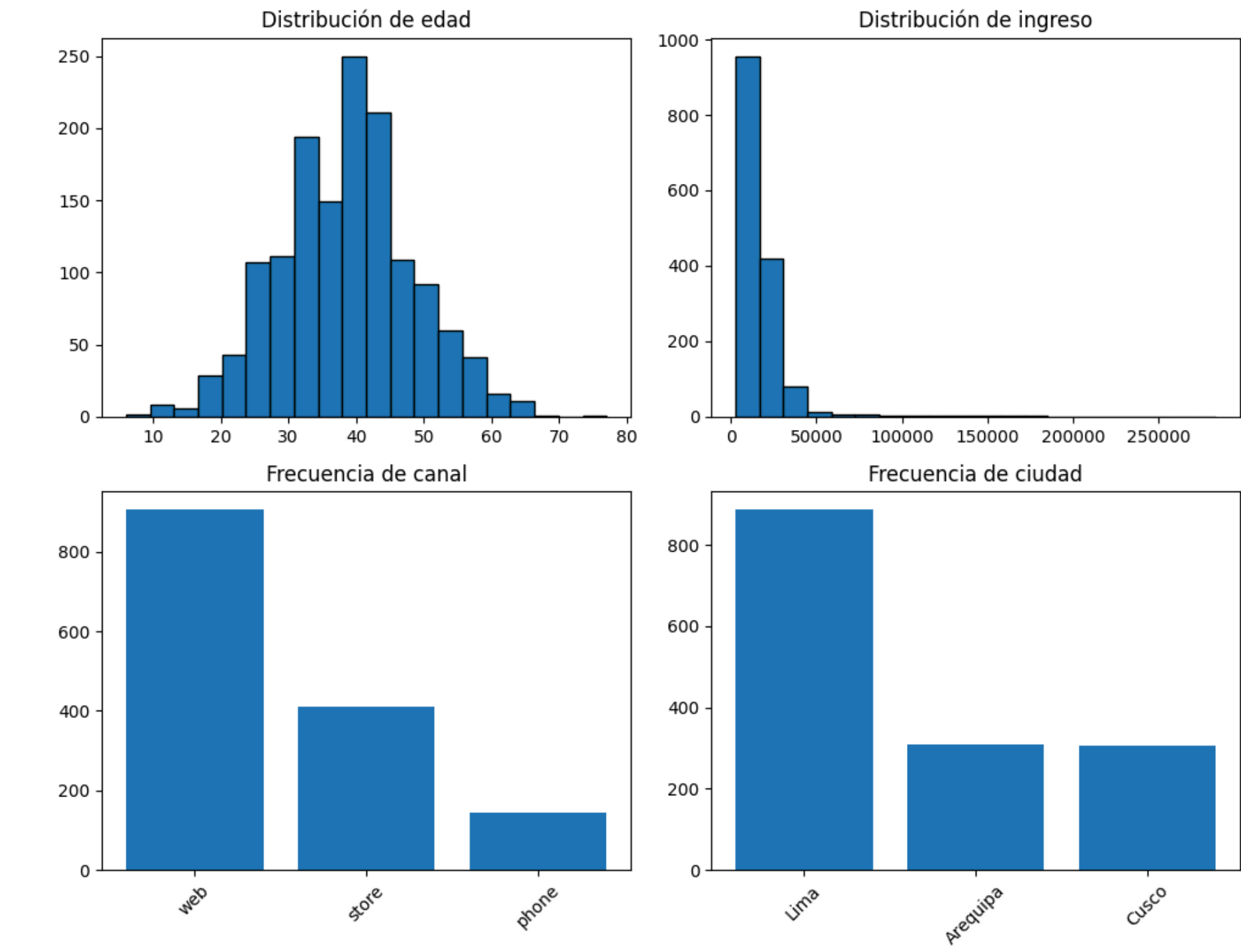
	edad	canal	ingreso	ciudad	fecha_alta	abandono
n_missing	60	40	0	0	0	0



2.3 Distribuciones y estadísticas básicas

```
1 # Distribuciones y estadísticas básicas
2 display(df[['edad','ingreso']].describe().T)
3
4 # Histogramas
5 fig, axes = plt.subplots(2, 2, figsize=(10, 8))
6
7 # Histograma de Edad
8 axes[0, 0].hist(df['edad'].dropna(), bins=20, edgecolor='black')
9 axes[0, 0].set_title('Distribución de edad')
10
11 # Histograma de Ingreso
12 axes[0, 1].hist(df['ingreso'].dropna(), bins=20, edgecolor='black')
13 axes[0, 1].set_title('Distribución de ingreso')
14
15 # Frecuencia de Canal
16 axes[1, 0].bar(df['canal'].value_counts().index, df['canal'].value_counts().values)
17 axes[1, 0].set_title('Frecuencia de canal')
18 axes[1, 0].tick_params(axis='x', rotation=45)
19
20 # Frecuencia de Ciudad
21 axes[1, 1].bar(df['ciudad'].value_counts().index, df['ciudad'].value_counts().values)
22 axes[1, 1].set_title('Frecuencia de ciudad')
23 axes[1, 1].tick_params(axis='x', rotation=45)
24
25 plt.tight_layout()
26 plt.show()
27
```

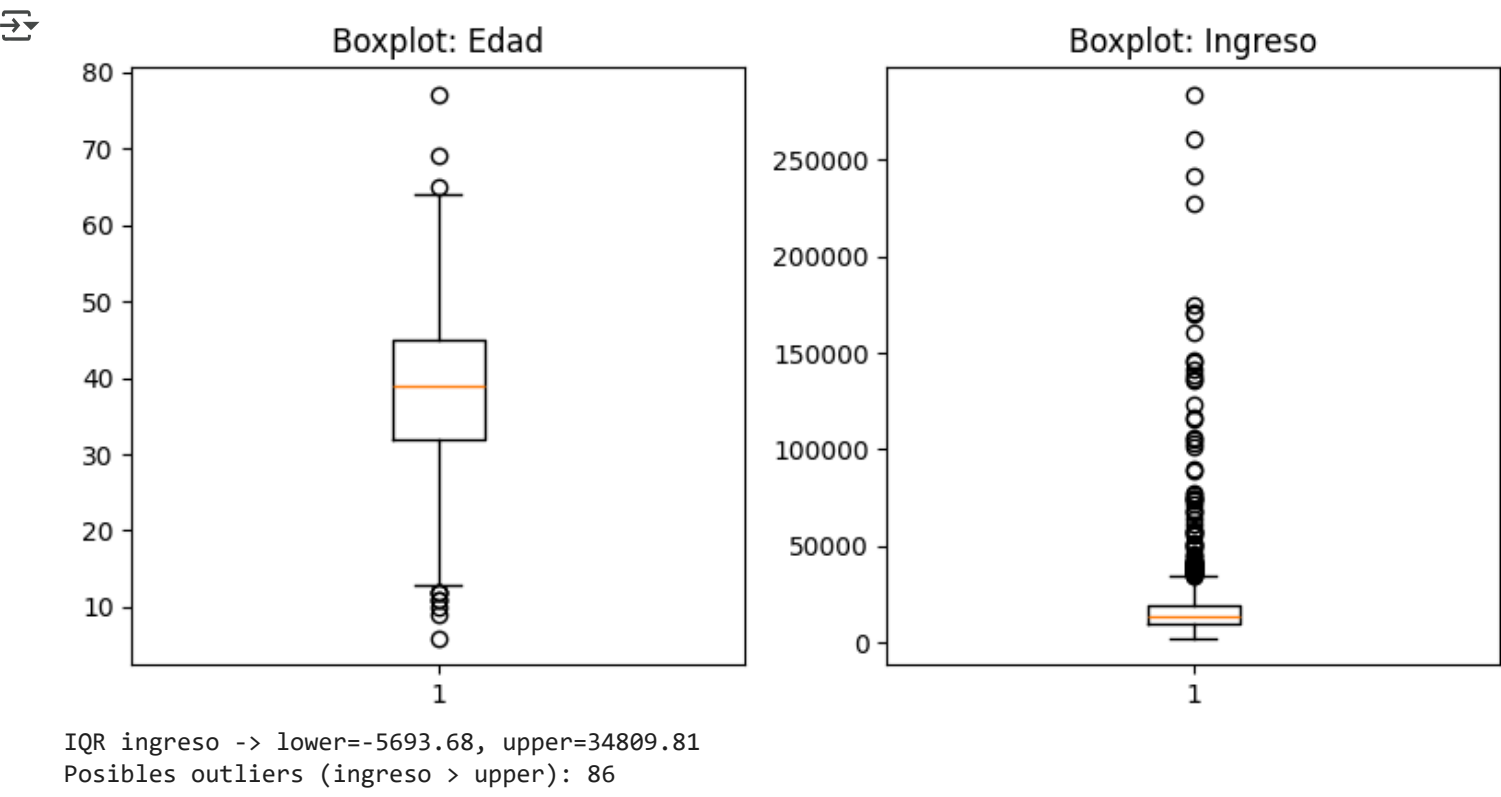
	count	mean	std	min	25%	50%	75%	max
edad	1441.0	38.514226	9.969187	6.000000	32.000000	39.000000	45.000000	77.000000
ingreso	1501.0	17886.646177	20426.808097	2538.346882	9495.125313	13504.647463	19620.998644	283753.883099



2.4 Outliers

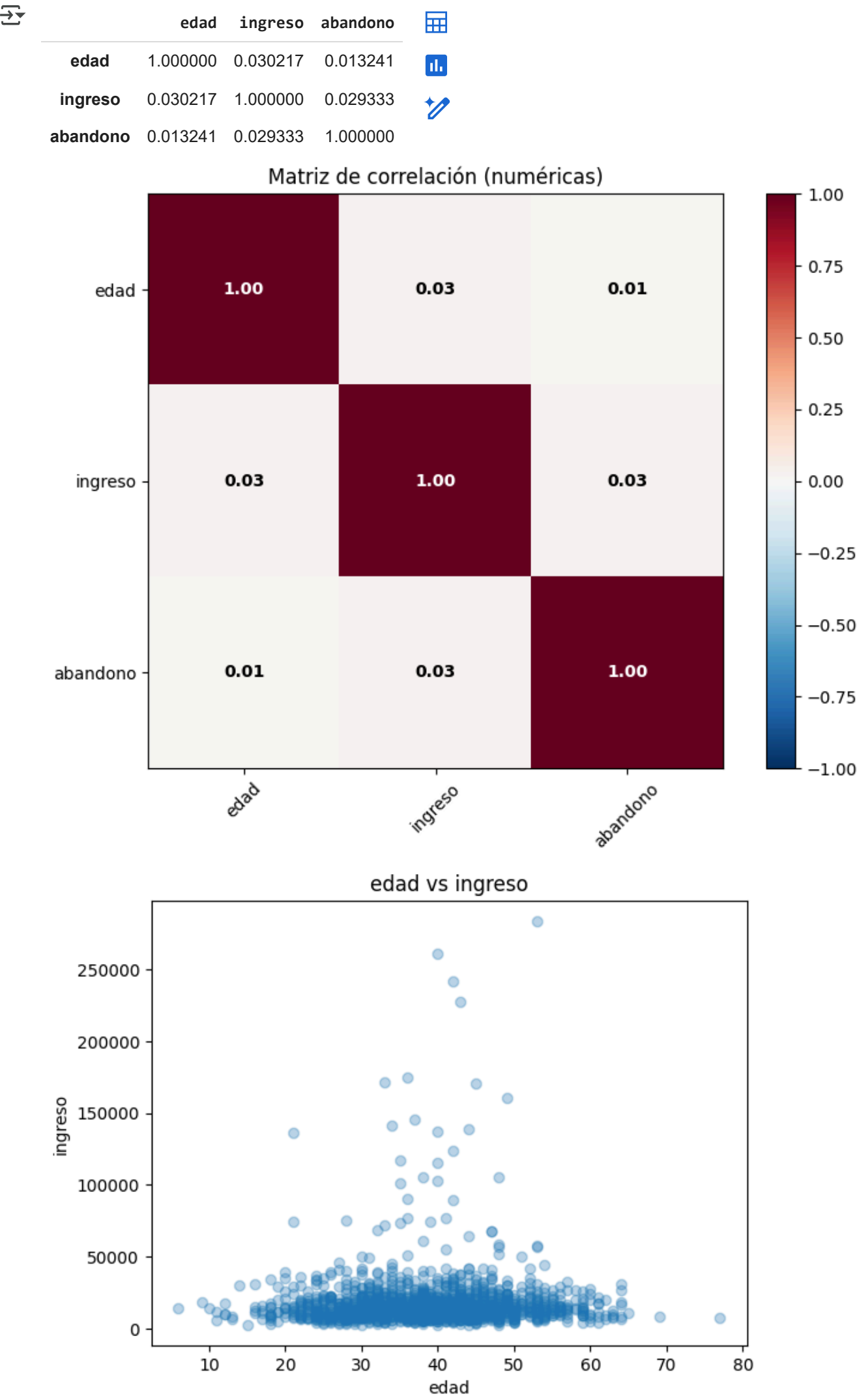
```
1 # Boxplots de edad e ingreso
2 fig, axes = plt.subplots(1, 2, figsize=(8, 4))
3
4 # Diagrama de caja de Edad
5 axes[0].boxplot(df['edad'].dropna(), vert=True)
6 axes[0].set_title('Boxplot: Edad')
7
8 # Diagrama de caja de Ingreso
9 axes[1].boxplot(df['ingreso'].dropna(), vert=True)
10 axes[1].set_title('Boxplot: Ingreso')
11
12 plt.tight_layout()
13 plt.show()
14
15 # Cálculo de IQR y outliers en ingreso
```

```
16 Q1, Q3 = df['ingreso'].quantile([0.25, 0.75])
17 IQR = Q3 - Q1
18 upper = Q3 + 1.5 * IQR
19 lower = Q1 - 1.5 * IQR
20
21 print(f'IQR ingreso -> lower={lower:.2f}, upper={upper:.2f}')
22 print('Posibles outliers (ingreso > upper):', int((df['ingreso'] > upper).sum()))
23
```



2.5 Relaciones entre variables

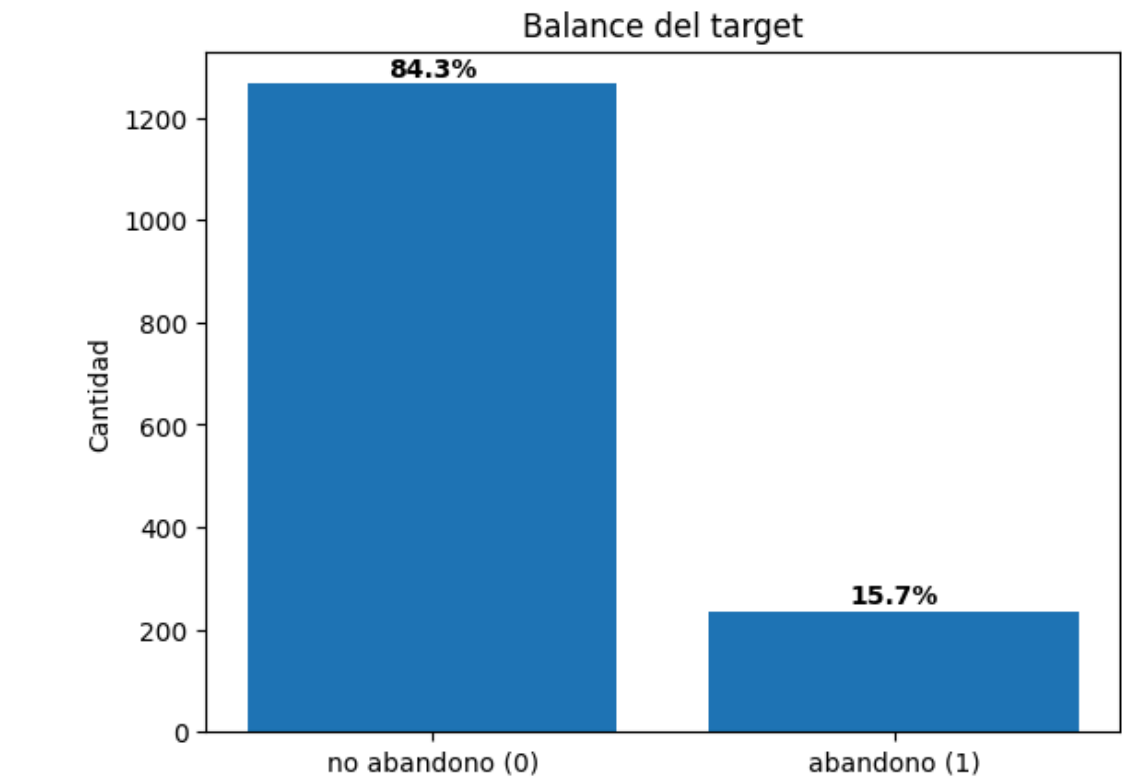
```
1 # Determinar correlación entre variables numéricas
2 corr = df[['edad', 'ingreso', 'abandono']].corr(method='pearson')
3 display(corr)
4
5 # Matriz de correlación entre las variables numéricas edad, ingreso y abandono
6 # Coeficiente de correlación de Pearson
7 plt.figure(figsize=(8, 6))
8 im = plt.imshow(corr, vmin=-1, vmax=1, cmap='RdBu_r')
9
10 # Añadir los valores de correlación como texto en cada celda
11 for i in range(corr.shape[0]):
12     for j in range(corr.shape[1]):
13         plt.text(j, i, f'{corr.iloc[i, j]:.2f}', # Formato a 2 decimales
14                 ha='center', va='center',
15                 color='white' if abs(corr.iloc[i, j]) > 0.5 else 'black', # Color contraste
16                 fontweight='bold')
17
18 plt.xticks(range(corr.shape[1]), corr.columns, rotation=45)
19 plt.yticks(range(corr.shape[0]), corr.index)
20 plt.colorbar(im)
21 plt.title('Matriz de correlación (numéricas)')
22 plt.tight_layout()
23 plt.show()
24
25 # Diagrama de dispersión entre edad (eje X) e ingreso (eje Y).
26 plt.figure()
27 plt.scatter(df['edad'], df['ingreso'], alpha=0.3)
28 plt.xlabel('edad'); plt.ylabel('ingreso'); plt.title('edad vs ingreso')
29 plt.show()
```



2.6 Análisis del target

```
1 # Explorar el balance de la variable objetivo (abandono)
2 tc = df['abandono'].value_counts().sort_index()
3 print(tc)
4
5 plt.figure()
6 bars = plt.bar(['no abandono (0)', 'abandono (1)'], tc.values)
7 plt.title('Balance del target')
8 plt.ylabel('Cantidad')
9
10 # Calcular porcentajes y añadirlos sobre las barras
11 total = sum(tc.values)
12 for i, bar in enumerate(bars):
13     height = bar.get_height()
14     percentage = (height / total) * 100
15     plt.text(bar.get_x() + bar.get_width()/2., height + 5, # Posición del texto
16             f'{percentage:.1f}%', # Formato a 1 decimal
17             ha='center', va='bottom', # Alineación centrada
18             fontweight='bold')
19
20 plt.show()
```

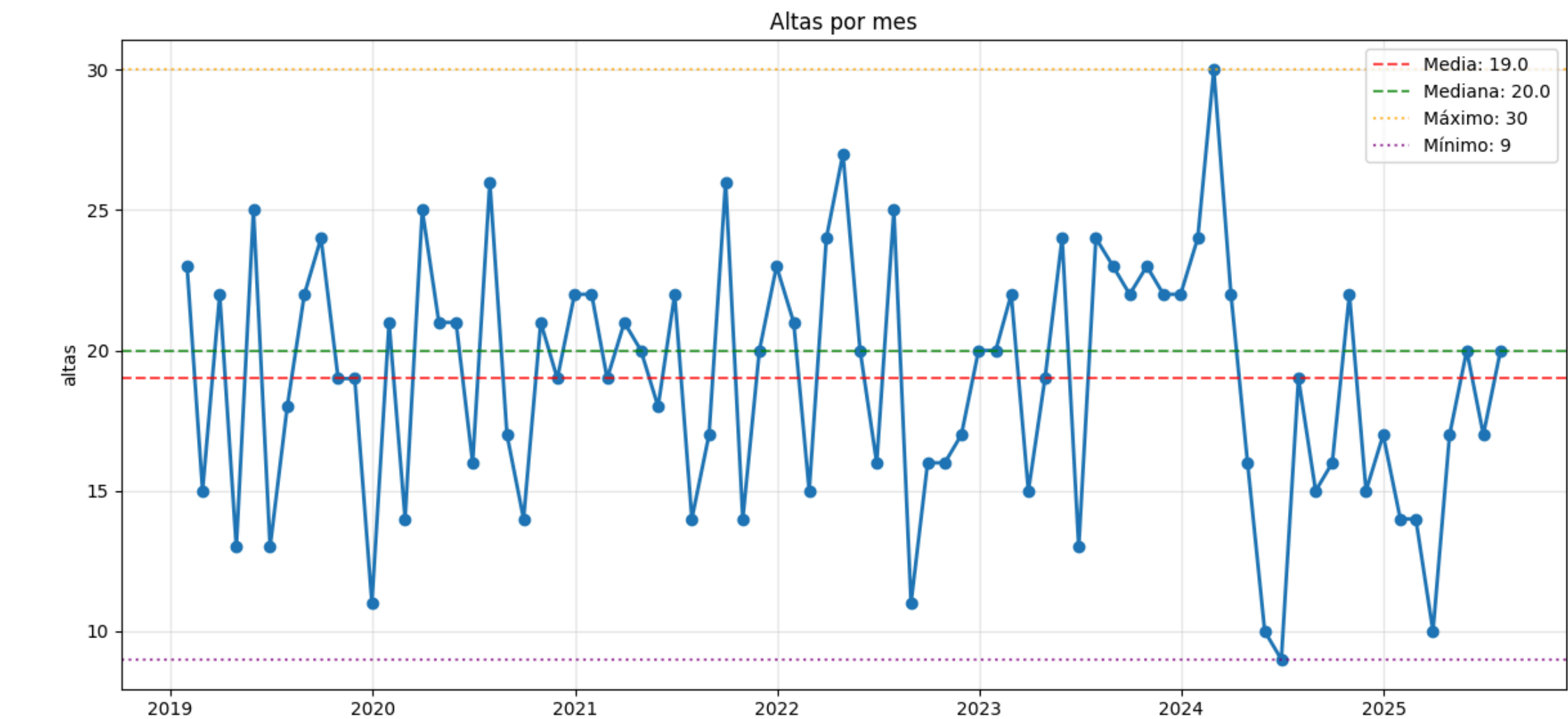
```
abandono
0    1266
1     235
Name: count, dtype: int64
```



2.7 Patrones temporales

```
1 # crear serie de pandas
2 series = pd.Series(1, index=pd.to_datetime(df['fecha_alta'])).resample('M').sum()
3 plt.figure(figsize=(12, 6))
4 plt.plot(series.index, series.values, marker='o', linestyle='-', linewidth=2)
5 plt.title('Altas por mes'); plt.xlabel('mes'); plt.ylabel('altas')
6
7 # Agregar líneas horizontales de referencia
8 plt.axhline(y=series.mean(), color='r', linestyle='--', alpha=0.7, label=f'Media: {series.mean():.1f}')
9 plt.axhline(y=series.median(), color='g', linestyle='--', alpha=0.7, label=f'Mediana: {series.median():.1f}')
10 plt.axhline(y=series.max(), color='orange', linestyle=':', alpha=0.7, label=f'Máximo: {series.max():.0f}')
11 plt.axhline(y=series.min(), color='purple', linestyle=':', alpha=0.7, label=f'Mínimo: {series.min():.0f}')
12
13 # Añadir cuadrícula
14 plt.grid(True, alpha=0.3)
15
16 # Mostrar leyenda
17 plt.legend()
18
19 plt.tight_layout()
20 plt.show()
```

```
/tmp/ipython-input-3046938592.py:2: FutureWarning: 'M' is deprecated and will be removed in a future version, please use 'ME' instead.
series = pd.Series(1, index=pd.to_datetime(df['fecha_alta'])).resample('M').sum()
```




3. Limpieza / Preparación

Objetivo: corregir problemas (nulos, outliers, tipos), normalizar y **dejar los datos listos** para el modelado, evitando *data leakedad*.

3.1 Duplicados e inconsistencias

```
1 # eliminación de duplicados
2 before = len(df)
3 df = df.drop_duplicates()
4 print('Duplicados eliminados:', before - len(df))
```

 Duplicados eliminados: 1

3.2 Conversión de tipos

```
1 # información antes de conversión (poner atención a atributos categóricos)
2 print("\nInformación general antes de la conversión:")
3 print(df.info())
```



```
Información general antes de la conversión:
<class 'pandas.core.frame.DataFrame'>
Index: 1500 entries, 0 to 1499
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   edad        1440 non-null   float64
1   ingreso     1500 non-null   float64
2   canal       1460 non-null   object
3   ciudad      1500 non-null   object
4   fecha_alta  1500 non-null   datetime64[ns]
5   abandono   1500 non-null   int64
dtypes: datetime64[ns](1), float64(2), int64(1), object(2)
memory usage: 82.0+ KB
None
```

```
1 # Convertir la columna a tipo datetime, forzando valores inválidos a NaT (Not a Time)
2 df['fecha_alta'] = pd.to_datetime(df['fecha_alta'], errors='coerce')
3
4 # Convertir a tipo category (para eficiencia de memoria y procesamiento)
5 df['canal'] = df['canal'].astype('category')
6 df['ciudad'] = df['ciudad'].astype('category')
7
8 # Mostrar información resumida del DataFrame: tipos de datos, memoria usada y valores no nulos
9 df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
Index: 1500 entries, 0 to 1499
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   edad        1440 non-null   float64
1   ingreso     1500 non-null   float64
2   canal       1460 non-null   category
3   ciudad      1500 non-null   category
4   fecha_alta  1500 non-null   datetime64[ns]
5   abandono   1500 non-null   int64
dtypes: category(2), datetime64[ns](1), float64(2), int64(1)
memory usage: 61.8 KB
```

3.3 Imputación de valores faltantes

```
1 print("=== ESTADÍSTICOS ANTES Y DESPUÉS DE LA IMPUTACIÓN ===\n")
2
3 # Estadísticos originales de edad
4 print("IMPACTO EN EDAD:")
5 print("Original - antes de imputación:")
6 print(f"  Valores nulos: {df['edad'].isnull().sum()}")
7 print(f"  Media: {df['edad'].mean():.2f}")
8 print(f"  Mediana: {df['edad'].median():.2f}")
9 print(f"  Desviación estándar: {df['edad'].std():.2f}")
10 print(f"  Mínimo: {df['edad'].min():.2f}")
11 print(f"  Máximo: {df['edad'].max():.2f}")
12
13 # Reemplazar los valores nulos con la mediana de la columna
14 imp_edad = SimpleImputer(strategy='median')
15 df[['edad_imp']] = imp_edad.fit_transform(df[['edad']])
16
17 print("\nDespués de imputación (mediana):")
18 print(f"  Valores nulos: {df['edad_imp'].isnull().sum()}")
19 print(f"  Media: {df['edad_imp'].mean():.2f}")
20 print(f"  Mediana: {df['edad_imp'].median():.2f}")
21 print(f"  Desviación estándar: {df['edad_imp'].std():.2f}")
22 print(f"  Mínimo: {df['edad_imp'].min():.2f}")
23 print(f"  Máximo: {df['edad_imp'].max():.2f}")
24
25 print("\n" + "="*50 + "\n")
26
27 # Estadísticos originales de canal
28 print("IMPACTO EN CANAL:")
29 print("Original - antes de imputación:")
30 print(f"  Valores nulos: {df['canal'].isnull().sum()}")
31 print(f"  Valores únicos: {df['canal'].nunique()}")
32 print("  Distribución de categorías:")
33 print(df['canal'].value_counts(dropna=False))
34
35 # Reemplazar los nulos con la categoría más frecuente (moda)
36 imp_canal = SimpleImputer(strategy='most_frequent')
37 df[['canal_imp']] = imp_canal.fit_transform(df[['canal']])
38 df['canal_imp'] = df['canal_imp'].astype('category')
39
40 print("\nDespués de imputación (moda):")
41 print(f"  Valores nulos: {df['canal_imp'].isnull().sum()}")
42 print(f"  Valores únicos: {df['canal_imp'].nunique()}")
43 print("  Distribución de categorías:")
44 print(df['canal_imp'].value_counts())
45
46 print("\n" + "="*50 + "\n")
47
48 # Mostrar las primeras filas comparativas
49 print("  PRIMERAS FILAS COMPARATIVAS:")
50 display(df[['edad', 'edad_imp', 'canal', 'canal_imp']].head(10))
51
52 # Mostrar filas donde hubo imputación
53 print("\n  FILAS CON IMPUTACIÓN (donde original era nulo):")
54 imputed_rows = df[df['edad'].isnull() | df['canal'].isnull()]
55 display(imputed_rows[['edad', 'edad_imp', 'canal', 'canal_imp']].head())
```

```

===== ESTADÍSTICOS ANTES Y DESPUÉS DE LA IMPUTACIÓN =====

IMPACTO EN EDAD:
Original - antes de imputación:
  Valores nulos: 60
  Media: 38.52
  Mediana: 39.00
  Desviación estándar: 9.97
  Mínimo: 6.00
  Máximo: 77.00




Después de imputación (mediana):
  Valores nulos: 0
  Media: 38.54
  Mediana: 39.00
  Desviación estándar: 9.77
  Mínimo: 6.00
  Máximo: 77.00

=====

IMPACTO EN CANAL:
Original - antes de imputación:
  Valores nulos: 40
  Valores únicos: 3
  Distribución de categorías:
canal
web      906
store    411
phone    143
NaN       40
Name: count, dtype: int64

Después de imputación (moda):
  Valores nulos: 0
  Valores únicos: 3
  Distribución de categorías:
canal_imp
web      946
store    411
phone    143
Name: count, dtype: int64

=====
```

PRIMERAS FILAS COMPARATIVAS:					
	edad	edad_imp	canal	canal_imp	
0	43.0	43.0	store	store	
1	37.0	37.0	phone	phone	
2	44.0	44.0	web	web	
3	53.0	53.0	phone	phone	
4	36.0	36.0	web	web	
5	36.0	36.0	web	web	
6	54.0	54.0	NaN	web	
7	46.0	46.0	web	web	
8	33.0	33.0	store	store	
9	43.0	43.0	web	web	
FILAS CON IMPUTACIÓN (donde original era nulo):					
	edad	edad_imp	canal	canal_imp	
6	54.0	54.0	NaN	web	
10	NaN	39.0	web	web	
32	38.0	38.0	NaN	web	
40	45.0	45.0	NaN	web	
47	49.0	49.0	NaN	web	

Importante: En un flujo productivo, nunca se debe imputar valores sobre todo el dataset simultáneamente debido al riesgo de data leakage (fuga de datos). Este problema ocurre cuando información del conjunto de prueba (TEST) o validación (VALID) se utiliza durante la fase de preprocesamiento, contaminando el proceso de entrenamiento.

La metodología correcta consiste en ajustar los imputadores utilizando exclusivamente los datos de entrenamiento (TRAIN) para calcular estadísticas como medianas, modas o medias. Luego, se debe aplicar ese mismo imputador ya ajustado para transformar tanto los datos de validación como de prueba. De esta manera, se garantiza que el modelo no tenga acceso anticipado a información futura y su evaluación sea realista y confiable.

Esto se resuelve de forma elegante con un **ColumnTransformer + Pipeline** (Se ilustra en la sección 4.6).

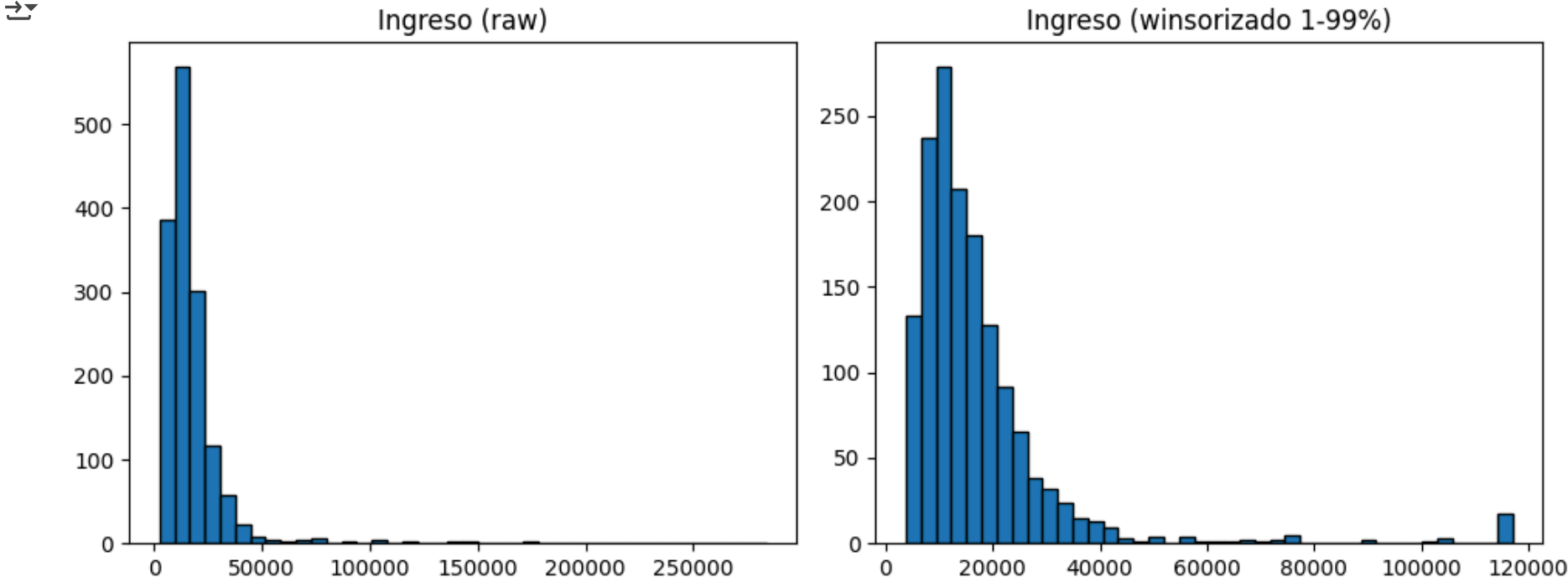
3.4 Tratamiento de outliers

Winsorización es el proceso de reducir el impacto de outliers extremos sin eliminarlos del dataset.
En que consiste:

- Se calculan los percentiles 1 y 99 de los datos (ignorando NaN)
- Estos valores se usan como límites inferior y superior.
- Con ***np.clip(x, low, high)*** se “recorta” todos los valores que estén por debajo del percentil 1 (se reemplazan por el valor del percentil 1) y los que estén por encima del percentil 99 (se reemplazan por el valor del percentil 99).

```

1 def winsorize_1_99(x):
2     # Aplica winsorización al 1%: reemplaza valores extremos inferior y superior (1% y 99%)
3     # con los percentiles correspondientes para reducir el impacto de outliers
4     low, high = np.nanpercentile(x, 1), np.nanpercentile(x, 99)
5     return np.clip(x, low, high)
6
7 df['ingreso_win'] = winsorize_1_99(df['ingreso'].values)
8
9 # Mostrar histogramas lado a lado
10 fig, axes = plt.subplots(1, 2, figsize=(10, 4))
11
12 # Histograma original
13 axes[0].hist(df['ingreso'], bins=40, edgecolor='black')
14 axes[0].set_title('Ingreso (raw)')
15
16 # Histograma winsorizado
17 axes[1].hist(df['ingreso_win'], bins=40, edgecolor='black')
18 axes[1].set_title('Ingreso (winsorizado 1-99%)')
19
20 plt.tight_layout()
21 plt.show()
22
```



3.5 Escalado / normalización

Escalar (normalizar/estandarizar) variables numéricas

El escalado de datos numéricos consiste en transformar las variables a un rango o distribución comparable, evitando que diferencias en magnitud distorsionen el análisis o el aprendizaje automático; esto es necesario porque muchos algoritmos —como los basados en distancias (kNN, k-Means, SVM) o en gradientes (regresión logística, redes neuronales)— son sensibles a la escala de las variables, de modo que, sin escalado, aquellas con valores más grandes dominarían los cálculos y generarían modelos sesgados, menos estables o con convergencia más lenta.

Algunas técnicas de escalado

- *StandardScaler*: transforma cada variable para que tenga media 0 y desviación estándar 1.
- *RobustScaler*: usa la mediana y el IQR (Q3–Q1) en lugar de media y desviación estándar, por lo que es más robusto frente a outliers.
- *MinMaxScaler*: escala cada variable a un rango fijo, normalmente [0, 1] (puedes cambiarlo).

```
1 # Definir técnicas de escalado
2 std = StandardScaler()
3 rob = RobustScaler()
4 mm = MinMaxScaler()
5
6 # Escalar variables con valores muy dispares
7 X_num = df[['edad_imp', 'ingreso_win']].values
8 std_vals = std.fit_transform(X_num)
9 rob_vals = rob.fit_transform(X_num)
10 mm_vals = mm.fit_transform(X_num)
11
12 # Mostrar valores originales
13 print('Valores originales (3 primeras filas)')
14 print(X_num[:3])
15 print()
16
17 # Mostrar resultados de valores escalados
18 print('StandardScaler (3 primeras filas):\n', std_vals[:3])
19 print('RobustScaler (3 primeras filas):\n', rob_vals[:3])
20 print('MinMaxScaler (3 primeras filas):\n', mm_vals[:3])
```

Valores originales (3 primeras filas)

[43.	20498.32781689]
[37.	9865.98100033]
[44.	8518.45797095]

```
StandardScaler (3 primeras filas):
[[ 0.45707008  0.20819368]
 [-0.15717969 -0.4830578 ]
 [ 0.55944504 -0.57066568]]
RobustScaler (3 primeras filas):
[[ 0.32653061  0.68903842]
 [-0.16326531 -0.35914773]
 [ 0.40816327 -0.49199281]]
MinMaxScaler (3 primeras filas):
[[0.52112676 0.14757911]
 [0.43661972 0.05373898]
 [0.53521127 0.04184586]]
```

3.6 Codificación de variables categóricas

La **codificación de variables categóricas** es el proceso de transformar variables que contienen categorías o etiquetas (texto o clases) en representaciones numéricas que los algoritmos de Machine Learning puedan procesar. Se hace porque la mayoría de los modelos matemáticos solo trabajan con números.

Se codifican variables categóricas para que los algoritmos:

- Puedan utilizarlas en cálculos matemáticos, ya que los modelos no interpretan texto.
- Eviten sesgos artificiales: la elección del método de codificación (one-hot, ordinal, target encoding, etc.) depende del tipo de variable y del modelo; por ejemplo, no conviene asignar números arbitrarios (1,2,3) a categorías sin orden, porque el modelo podría “suponer” que hay una relación ordinal que no existe.

```
1 from sklearn.preprocessing import OneHotEncoder
2 import numpy as np
3 import pandas as pd
4
5 # Convertir variables categóricas (canal_imp y ciudad) a variables numéricas binarias (0/1) usando One-Hot Encoding
6 ohe = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
7 X_cat = ohe.fit_transform(df[['canal_imp', 'ciudad']])
8
9 print('Resultados de one hot encoding \n')
10
11 # Mostrar dimensiones
12 print(' Tamaño de la matriz transformada:', X_cat.shape)
13 print(f' - Filas (muestras): {X_cat.shape[0]}')
14 print(f' - Columnas (categorías binarias): {X_cat.shape[1]}')
15
16 # Mostrar nombres de las nuevas columnas
17 print('\n Nombres de las nuevas columnas:')
18 categorias = ohe.get_feature_names_out(['canal_imp', 'ciudad'])
19 for i, categoria in enumerate(categorias):
20     print(f' Columna {i}: {categoria}')
21
22 # Mostrar valores únicos originales y su transformación
23 print('\n Valores únicos originales:')
24 print(' - canal_imp:', df['canal_imp'].unique())
25 print(' - ciudad:', df['ciudad'].unique())
26
27 # Mostrar ejemplos de la transformación (primeras 5 filas)
28 print('\n Ejemplo de transformación (primeras 5 filas):')
29 # Separar las columnas por variable original
30 canal_columns = [col for col in categorias if col.startswith('canal_imp')]
```



```
1 ciudad_columns = [col for col in categorias if col.startswith('ciudad')]
32 # Mostrar codificación de CANAL
33 print('\n Codificación One-Hot para CANAL_IMP:')
34 df_canal = pd.DataFrame(X_cat[:5, :len(canal_columns)], columns=canal_columns)
35 df_canal.insert(0, 'canal_imp_original', df['canal_imp'].head().values)
36 display(df_canal)
37 # Mostrar codificación de CIUDAD
38 print('\n Codificación One-Hot para CIUDAD:')
39 df_ciudad = pd.DataFrame(X_cat[:5, len(canal_columns):], columns=ciudad_columns)
40 df_ciudad.insert(0, 'ciudad_original', df['ciudad'].head().values)
41 display(df_ciudad)
42
43 # Información adicional
44 print('\n Información adicional:')
45 print(f' - Se crearon {len(categorias)} columnas binarias')
46 print(' - handle_unknown="ignore": categorías nuevas en test se codifican como ceros')
47 print(' - Cada fila tendrá exactamente un 1 por variable original (suma=1 por grupo)')
```

Resultados de one hot encoding

Tamaño de la matriz transformada: (1500, 6)
- Filas (muestras): 1500
- Columnas (categorías binarias): 6

Nombres de las nuevas columnas:
Columna 0: canal_imp_phone
Columna 1: canal_imp_store
Columna 2: canal_imp_web
Columna 3: ciudad_Arequipa
Columna 4: ciudad_Cusco
Columna 5: ciudad_Lima

Valores únicos originales:
- canal_imp: ['store', 'phone', 'web']
Categories (3, object): ['phone', 'store', 'web']
- ciudad: ['Lima', 'Arequipa', 'Cusco']
Categories (3, object): ['Arequipa', 'Cusco', 'Lima']

Ejemplo de transformación (primeras 5 filas):

Codificación One-Hot para CANAL_IMP:				
	canal_imp_original	canal_imp_phone	canal_imp_store	canal_imp_web
0	store	0.0	1.0	0.0
1	phone	1.0	0.0	0.0
2	web	0.0	0.0	1.0
3	phone	1.0	0.0	0.0
4	web	0.0	0.0	1.0

Codificación One-Hot para CIUDAD:				
	ciudad_original	ciudad_Arequipa	ciudad_Cusco	ciudad_Lima
0	Lima	0.0	0.0	1.0
1	Arequipa	1.0	0.0	0.0
2	Lima	0.0	0.0	1.0
3	Lima	0.0	0.0	1.0
4	Lima	0.0	0.0	1.0

Información adicional:
- Se crearon 6 columnas binarias
- handle_unknown="ignore": categorías nuevas en test se codifican como ceros
- Cada fila tendrá exactamente un 1 por variable original (suma=1 por grupo)

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

Ejercicio:
Investigar otras técnicas de codificar variables categóricas, y escribirlas en esta celda.

1 Empieza a programar o a [crear código](#) con IA.

3.7 División Train/Valid/Test

Separar los datos en conjuntos de entrenamiento y prueba (train/test split), preparando el dataset para entrenar y evaluar un modelo de Machine Learning.

```
1 # Separar características base (sin transformar) y variable objetivo
2 X_base = df[['edad','ingreso','canal','ciudad','fecha_alta']].copy() # usaremos originales para el ColumnTransformer
3 y = df['abandono']
4
5 # Dividir en conjuntos de entrenamiento y prueba manteniendo la proporción de clases (stratify preserva balance de clases)
6 X_train, X_test, y_train, y_test = train_test_split(
7     X_base, y, test_size=0.2, stratify=y, random_state=42
8 )
9
10 # mostrar información de la división
11 print('Tamaños -> train:', X_train.shape, ' test:', X_test.shape)
12 print('Balance train:', y_train.value_counts(normalize=True).round(3).to_dict())
```

Tamaños -> train: (1200, 5) test: (300, 5)
Balance train: {0: 0.843, 1: 0.157}

Ejercicio:
¿Cómo separar un dataset en conjuntos de entrenamiento, validación y prueba?
Intente implementar una función que generalice este proceso.

1 Empieza a programar o a [crear código](#) con IA.

4. Ingeniería de características

Objetivo: Crear y transformar variables para aumentar su poder predictivo (mejorar el valor expresivo de los atributos) y facilitar el aprendizaje del modelo

4.1 Derivadas de fecha/tiempo

Transformar fechas, enriquece los datos y permite que los modelos de Machine Learning detecten tendencias temporales y estacionales que influyen en el comportamiento (por ejemplo, abandono de clientes). Es crucial porque las variables de tiempo son periódicas: después de diciembre viene enero, después de domingo viene lunes, etc.

```
1 # Establecer fecha de corte, para calcular el tiempo de antigüedad de cada cliente
2 fecha_corte = np.datetime64('2025-08-23')
```

```
4 # Convierte fecha_alta en fechas tipo datetime, para calcular
5 # dias_antigüedad = número de días desde la fecha de alta hasta el corte,
6 # para medir la antigüedad del cliente en días.
7 def agregar_caracteristicas_temporales(parte):
8     # Convertir la columna de fecha a formato datetime para operaciones temporales
9     valores = pd.to_datetime(parte['fecha_alta']).values.astype('datetime64[ns]')
10
11     # Calcular antigüedad en días desde la fecha de alta hasta la fecha de corte
12     antigüedad = (fecha_corte - valores).astype('timedelta64[D]').astype('float')
13     parte['dias_antigüedad'] = antigüedad # Almacenar antigüedad en días como float
14
15     # Crear representación cíclica de mes usando seno/coseno (captura estacionalidad)
16     fecha_dt = pd.to_datetime(parte['fecha_alta'])
17     parte['mes_seno'] = np.sin(2*np.pi*(fecha_dt.dt.month/12)) # Seno para ciclo mensual
18     parte['mes_coseno'] = np.cos(2*np.pi*(fecha_dt.dt.month/12)) # Coseno para ciclo mensual
19
20     # Crear representación cíclica del día de la semana (captura patrones semanales)
21     parte['dia_semana_seno'] = np.sin(2*np.pi*(fecha_dt.dt.dayofweek/7)) # Seno para día semana
22     parte['dia_semana_coseno'] = np.cos(2*np.pi*(fecha_dt.dt.dayofweek/7)) # Coseno para día semana
23
24 # DEMOSTRACIÓN EDUCATIVA (FUERA DE PIPELINE):
25 # Se crean copias explícitas para visualizar el efecto de la ingeniería de características.
26 # EN PRODUCCIÓN esto debe hacerse DENTRO del pipeline para evitar data leakage.
27 # Propósito didáctico: Mostrar paso a paso cómo se transforman los datos.
28 X_train_ic = X_train.copy()
29 X_test_ic = X_test.copy()
30 agregar_caracteristicas_temporales(X_train_ic); agregar_caracteristicas_temporales(X_test_ic)
31
32 X_train_ic[['fecha_alta', 'dias_antigüedad', 'mes_seno', 'dia_semana_coseno']].head()
```

	fecha_alta	dias_antigüedad	mes_seno	dia_semana_coseno	
1088	2023-12-11	621.0	-2.449294e-16	1.000000	
1380	2023-01-01	965.0	5.000000e-01	0.623490	
132	2021-03-26	1611.0	1.000000e+00	-0.900969	
808	2025-06-17	67.0	1.224647e-16	0.623490	
358	2022-05-30	1181.0	5.000000e-01	1.000000	

4.2 Transformaciones y binning

Binning se refiere a la técnica de agrupar valores numéricos continuos en intervalos o “cajones” (bins) y asignarles una categoría o etiqueta.

Propósito

- Simplificar el modelo: convierte una variable continua en categorías más manejables.
- Capturar relaciones no lineales: quizá el riesgo de abandono no aumenta de forma lineal con la edad, sino por tramos (ejemplo: jóvenes y adultos mayores se comportan distinto).
- Robustez: reduce la sensibilidad a outliers o pequeñas fluctuaciones en los valores.

```
1 # Ingeniería de características: Transformaciones para mejorar el poder predictivo
2 # - Winsorización: Reduce impacto de valores extremos en ingresos
3 # - Transformación logarítmica: Normaliza distribución sesgada de ingresos
4 # - Discretización de edad: Captura patrones no lineales por grupos etarios
5
6 eps = 1e-6 # Épsilon para evitar log(0) en caso de ingresos cero
7
8 for part in (X_train_ic, X_test_ic):
9
10     # Winsorización local (solo para demo; en producción se haría dentro del pipeline)
11     def winsorize_1_99_local(x):
12         # Limita valores extremos: reemplaza por percentiles 1 y 99
13         low, high = np.nanpercentile(x, 1), np.nanpercentile(x, 99)
14         return np.clip(x, low, high)
15
16     # Aplicar winsorización a ingresos para atenuar outliers
17     ingreso_win_local = winsorize_1_99_local(part['ingreso'].values)
18
19     # Transformación logarítmica: convierte distribución exponencial en normal
20     part['ingreso_log'] = np.log(ingreso_win_local + eps)
21
22     # Discretización de edad en grupos etarios para capturar relaciones no lineales
23     part['edad_bin'] = pd.cut(part['edad'], bins=[0,25,35,45,60,120], labels=False)
24
25 # Mostrar efecto de las transformaciones en las primeras filas
26 X_train_ic[['ingreso', 'ingreso_log', 'edad', 'edad_bin']].head()
```

	ingreso	ingreso_log	edad	edad_bin	
1088	8603.968158	9.059979	35.0	1.0	
1380	3900.939973	8.268973	45.0	2.0	
132	9377.304109	9.146048	NaN	NaN	
808	17925.473965	9.793978	51.0	3.0	
358	15658.692860	9.658781	41.0	2.0	

4.3 Interacciones y cruces

Los **feature de interacción** combinan dos características para capturar relaciones más complejas que cada variable por separado.

Ejemplo

- Un cliente con alto ingreso pero poca antigüedad puede comportarse distinto a uno con alto ingreso y mucha antigüedad.
- Esta interacción permite que el modelo “vea” esa diferencia.

```
1 # Crear variable de interacción: ingreso multiplicado por antigüedad en días
2 # Esta feature captura el efecto combinado del poder adquisitivo y la lealtad del cliente
3 for part in (X_train_ic, X_test_ic):
4     part['ingreso_x_antigüedad'] = part['ingreso'] * part['dias_antigüedad']
5
6 # Mostrar las variables originales y la nueva feature de interacción
7 X_train_ic[['ingreso', 'dias_antigüedad', 'ingreso_x_antigüedad']].head()
```

	ingreso	dias_antigüedad	ingreso_x_antigüedad	
1088	8603.968158	621.0	5.343064e+06	
1380	3900.939973	965.0	3.764407e+06	
132	9377.304109	1611.0	1.510684e+07	
808	17925.473965	67.0	1.201007e+06	
358	15658.692860	1181.0	1.849292e+07	

4.4 Codificación avanzada (target encoding)

La técnica de codificación de variables categóricas conocida como Target Encoding (codificación basada en el objetivo) se considera una estrategia avanzada dentro de la Ingeniería de Características, ya que no se limita a asignar valores directos a las categorías, sino que transforma cada una de ellas mediante cálculos estadísticos derivados de la variable objetivo. De esta manera, se incrementa la capacidad expresiva de la representación categórica, permitiendo capturar relaciones más profundas entre las categorías y el comportamiento a predecir.

```
1 # Calcular la media de abandono por canal (Target Encoding)
2 # Esta técnica codifica categorías con la probabilidad promedio de abandono de cada grupo
3 means_by_canal = X_train_ic.join(y_train).groupby('canal')['abandono'].mean()
4
5 # Aplicar Target Encoding a los conjuntos de entrenamiento y prueba
6 X_train_ic['canal_te'] = X_train_ic['canal'].map(means_by_canal) # Train: usando medias calculadas
7 X_test_ic['canal_te'] = X_test_ic['canal'].map(means_by_canal) # Test: mismas medias de train (evita data leakage)
8
9 # Mostrar el mapeo original -> valor codificado
10 X_train_ic[['canal','canal_te']].head()
```

```
🔗 /tmp/ipython-input-1836557017.py:3: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True
  means_by_canal = X_train_ic.join(y_train).groupby('canal')['abandono'].mean()
```

	canal	canal_te	
1088	web	0.15775	🏠
1380	web	0.15775	🏠
132	web	0.15775	
808	web	0.15775	
358	web	0.15775	

4.5 Pipeline integrado con ColumnTransformer + evaluación

Ejercicio:
Investigar sobre "Pipeline". Documentar en esta celda.

Notar que en el pipeline se **imputa, winsoriza, escala y codifica** usando **solo TRAIN** internamente durante `fit`, evitando fugas.

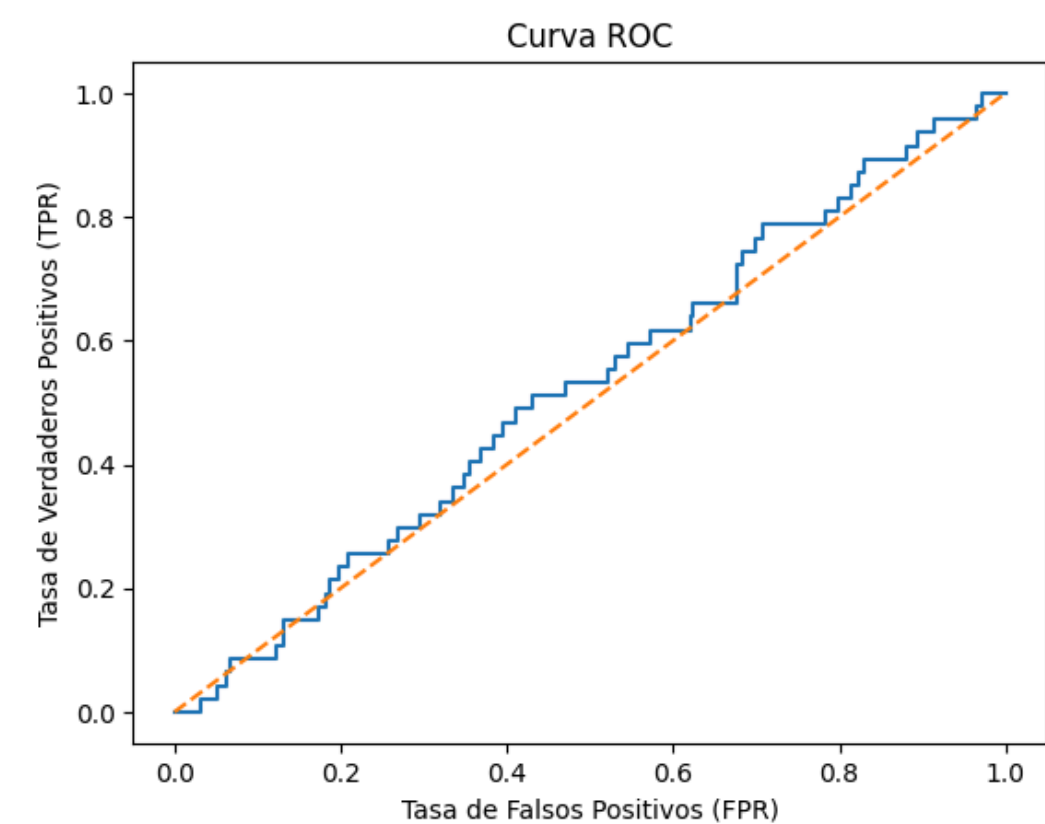
```
1 # TRANSFORMADORES PERSONALIZADOS PARA FEATURES ESPECIALES
2
3 def antigüedad_desde_fecha(X):
4     # Convierte fechas de alta a antigüedad en días (días desde fecha de corte)
5     if hasattr(X, "values"):
6         valores = X.values
7     else:
8         valores = np.asarray(X)
9     if valores.ndim == 1:
10        valores = valores.reshape(-1, 1)
11    import pandas as pd, numpy as np
12    hoy = np.datetime64('2025-08-23')
13    mascara_nat = pd.isna(valores[:, 0])
14    valores_dt = valores[:, 0].astype('datetime64[ns]')
15    antigüedad = (hoy - valores_dt).astype('timedelta64[D]').astype(float)
16    antigüedad[mascara_nat] = np.nan
17    return antigüedad.reshape(-1, 1)
18
19 def winsorizar_array_1_99(X):
20     # Aplica winsorización al 1% para reducir impacto de valores extremos en múltiples columnas
21     X = X.copy().astype(float)
22     inferior = np.nanpercentile(X, 1, axis=0); superior = np.nanpercentile(X, 99, axis=0)
23     for j in range(X.shape[1]):
24         X[:, j] = np.clip(X[:, j], inferior[j], superior[j])
25     return X
26
27 # Transformadores de función para usar en el pipeline
28 winsorizador_tf = FunctionTransformer(winsorizar_array_1_99, validate=False)
29 antigüedad_tf = FunctionTransformer(antigüedad_desde_fecha, validate=False)
30
31 # DEFINICIÓN DE CARACTERÍSTICAS POR TIPO
32 características_numericas = ['edad','ingreso']
33 características_categoricas = ['canal','ciudad']
34 característica_fecha = ['fecha_alta']
35
36 # PIPELINES ESPECÍFICOS POR TIPO DE CARACTERÍSTICA
37 pipeline_numerico = Pipeline([
38     ('imputacion', SimpleImputer(strategy='median')), # Imputa NaN con mediana
39     ('winsorizacion', winsorizador_tf), # Reduce valores extremos via winsorización
40     ('escalado', StandardScaler()) # Estandariza: media=0, desviación=1
41 ])
42 pipeline_categorico = Pipeline([
43     ('imputacion', SimpleImputer(strategy='most_frequent')), # Imputa NaN con moda
44     ('codificacion', OneHotEncoder(handle_unknown='ignore', sparse_output=False)) # One-hot encoding
45 ])
46 pipeline_fecha = Pipeline([
47     ('antigüedad', antigüedad_tf), # Convierte fecha a antigüedad en días
48     ('imputacion', SimpleImputer(strategy='median')), # Imputa NaN en antigüedad
49     ('escalado', StandardScaler()) # Estandariza antigüedad
50 ])
51
52 # COLUMN TRANSFORMER: APLICA TRANSFORMACIONES ESPECÍFICAS POR COLUMNA
53 preprocesador = ColumnTransformer([
54     ('numericas', pipeline_numerico, características_numericas), # Características numéricas
55     ('categoricas', pipeline_categorico, características_categoricas), # Características categóricas
56     ('fecha', pipeline_fecha, característica_fecha) # Característica de fecha
57 ])
58
59 # PIPELINE COMPLETO: PREPROCESAMIENTO + MODELO
60 clasificador = Pipeline([
61     ('preprocesamiento', preprocesador), # Transforma todas las características
62     ('modelo', LogisticRegression(max_iter=200, class_weight='balanced')) # Modelo con balanceo de clases
63 ])
64
65 # ENTRENAMIENTO Y EVALUACIÓN DEL MODELO
66 X_completo_entrenamiento = X_train.copy() # Usar solo entrenamiento para evitar fuga de datos
67 X_completo_prueba = X_test.copy()
68
69 clasificador.fit(X_completo_entrenamiento, y_train) # Entrena pipeline completo
70
71 # PREDICCIONES Y EVALUACIÓN
72 y_probabilidades = clasificador.predict_proba(X_completo_prueba)[:,-1] # Probabilidades de clase positiva
73 y_predicciones = (y_probabilidades >= 0.5).astype(int) # Predicciones binarias (umbral 0.5)
74
75 print(classification_report(y_test, y_predicciones, digits=3)) # Reporte de clasificación
76 print("ROC AUC:", roc_auc_score(y_test, y_probabilidades)) # Métrica principal para datos desbalanceados
77
78 # CURVA ROC PARA EVALUAR DESEMPEÑO EN DIFERENTES UMBRALES
79 fpr, tpr, umbrales = roc_curve(y_test, y_probabilidades)
80 plt.figure()
81 plt.plot(fpr, tpr)
82 plt.plot([0,1],[0,1], linestyle='--') # Línea de referencia (clasificador aleatorio)
```

```
83 plt.xlabel('Tasa de Falsos Positivos (FPR)'); plt.ylabel('Tasa de Verdaderos Positivos (TPR)'); plt.title('Curva ROC')
84 plt.show()
```



	precision	recall	f1-score	support
0	0.849	0.779	0.812	253
1	0.176	0.255	0.209	47
accuracy			0.697	300
macro avg	0.513	0.517	0.511	300
weighted avg	0.744	0.697	0.718	300

ROC AUC: 0.5235892691951896



4.6 (Opcional) SMOTE en el pipeline

Ejercicio:

Investigar sobre la técnica SMOTE. Documentar en esta celda.

```
1 try:
2     # Intentar importar SMOTE para balanceo de clases
3     from imblearn.over_sampling import SMOTE
4     from imblearn.pipeline import Pipeline as ImbPipeline # Pipeline especial para SMOTE
5
6     # Crear pipeline con SMOTE para balancear la clase minoritaria durante el entrenamiento
7     smote_clf = ImbPipeline(steps=[
8         ('preprocess', preprocesador), # Preprocesamiento de características
9         ('smote', SMOTE(random_state=42)), # SMOTE: genera muestras sintéticas de la clase minoritaria
10        ('model', LogisticRegression(max_iter=200)) # Modelo de regresión logística
11    ])
12
13    # Entrenar el pipeline con SMOTE (balancea solo el conjunto de entrenamiento)
14    smote_clf.fit(X_completo_entrenamiento, y_train)
15
16    # Realizar predicciones y calcular probabilidades
17    proba_s = smote_clf.predict_proba(X_completo_prueba)[:,-1] # Probabilidades de clase positiva
18    pred_s = (proba_s >= 0.5).astype(int) # Predicciones binarias con umbral 0.5
19
20    # Evaluar el modelo con SMOTE
21    print("SMOTE - ROC AUC:", roc_auc_score(y_test, proba_s)) # Métrica principal para datos desbalanceados
22    print(classification_report(y_test, pred_s, digits=3)) # Reporte detallado de clasificación
23
24 except Exception as e:
```