Facultad de Ciencias Económicas y Empresariales

ICADE

Predicción de valor y estrategia de pricing para aiXtensa

Un enfoque de inferencia causal sobre la automatización de recordatorios en clínicas de salud privadas

Autor: Sergio Cuervo Arango

Director: José Portela González

MADRID | Junio 2026

Resumen

Este trabajo desarrolla un motor analítico que estima el valor económico de la automatización de recordatorios de cita en clínicas privadas españolas, y fundamenta sobre esa estimación una estrategia de *pricing* basado en valor para el proyecto aiXtensa. La inasistencia a citas, o *no-show*, genera un coste directo y un coste de capacidad ociosa difícil de recuperar. El recordatorio por SMS es la intervención más barata y escalable para reducirla. El análisis se apoya en un registro de unas 107.000 citas, empleado como proxy conductual, con los parámetros económicos calibrados al mercado español. Combina cuatro piezas: un modelo predictivo de riesgo de *no-show* con *gradient boosting* calibrado e interpretado con valores SHAP, un análisis de inferencia causal del efecto del SMS con diagnósticos preespecificados, una segmentación de pacientes por perfil de riesgo y una simulación Monte Carlo del valor recuperado. El análisis exploratorio revela un fuerte sesgo de selección, porque el recordatorio se asignó a las citas de mayor riesgo. El análisis causal muestra que, con los datos observacionales disponibles, el efecto del SMS no es identificable de forma estable, lo que constituye en sí un resultado metodológico. Por ello, la simulación económica se apoya en la evidencia experimental publicada. La segmentación muestra que las inasistencias se concentran en los segmentos de mayor riesgo, mientras que el grupo de bajo riesgo apenas las genera, lo que indica que un envío uniforme de recordatorios es subóptimo. El modelo sitúa el valor neto mensual recuperado por clínica en una mediana en torno a 1.300 euros, y ancla tres tarifas, conservadora, recomendada y premium, a los percentiles de la distribución de valor. El trabajo concluye que la medición experimental debe formar parte del propio servicio.

Palabras clave: no-show; recordatorios SMS; inferencia causal; simulación Monte Carlo; pricing basado en valor; clínicas privadas.

Abstract

This thesis develops an analytical engine that estimates the economic value of automating appointment reminders in Spanish private healthcare clinics, and uses that estimate to ground a value-based pricing strategy for the aiXtensa venture. Missed appointments, or no-shows, generate both a direct revenue loss and an idle-capacity cost that is hard to recover. SMS reminders are the cheapest and most scalable intervention to reduce them. The analysis draws on a record of around 107,000 appointments, used as a behavioural proxy, with the economic parameters calibrated to the Spanish market. It combines four components: a no-show risk model built with calibrated gradient boosting and interpreted with SHAP values, a causal inference analysis of the SMS effect with pre-specified diagnostics, a patient segmentation by risk profile and a Monte Carlo simulation of the recovered value. The exploratory analysis reveals a strong selection bias, since reminders were sent to the highest-risk appointments. The causal analysis shows that, with the available observational data, the effect of SMS is not stably identifiable, which is itself a methodological result. The economic simulation therefore relies on published experimental evidence. The segmentation shows that missed appointments concentrate in the higher-risk segments, while the low-risk group barely produces any, indicating that sending reminders uniformly is suboptimal. The model places the median monthly net value recovered per clinic at around 1,300 euros and anchors three pricing tiers, conservative, recommended and premium, to the percentiles of the value distribution. The work concludes that experimental measurement should be part of the service itself.

Keywords: no-show; SMS reminders; causal inference; Monte Carlo simulation; value-based pricing; private clinics.

Índice

Resumen *2*

Abstract *3*

Índice *4*

1. Introducción y motivación *6*

1.1. La inasistencia a citas como problema operativo y económico **6**

1.2. Contexto empresarial: la cuantificación del valor **7**

1.3. Por qué predecir no basta **7**

1.4. Estructura del documento **8**

2. Objetivos *9*

3. Marco teórico y estado del arte *10*

3.1. Determinantes de la inasistencia **10**

3.2. Evidencia experimental de los recordatorios SMS **10**

3.3. Inferencia causal con datos observacionales **12**

3.4. Modelos de gradient boosting e interpretabilidad **13**

3.5. Nota sobre el pricing basado en valor **14**

4. Datos y metodología *15*

4.1. El conjunto de datos **15**

4.2. Limpieza y construcción de variables **16**

4.3. Modelo predictivo **17**

4.4. Diseño del análisis causal **17**

4.5. Segmentación de pacientes **18**

4.6. Simulación económica y parámetro de efecto **18**

4.7. Benchmarking de mercado **19**

5. Resultados *20*

5.1. Análisis exploratorio **20**

5.2. Modelo predictivo **21**

5.3. Análisis causal **23**

5.4. Segmentación de pacientes **25**

5.5. Simulación económica **27**

5.6. Propuesta de pricing **28**

6. Discusión y limitaciones *30*

7. Conclusiones *32*

Bibliografía *33*

Anexo A. Declaración de uso de herramientas de inteligencia artificial generativa ... *35*

Anexo B. Código fuente del análisis *36*

B.1 Limpieza de datos y creación de variables **36**

B.2 Análisis exploratorio de datos **41**

B.3 Modelo predictivo: XGBoost, calibración e interpretabilidad (SHAP) **50**

B.4 Inferencia causal por ponderación de propensión (IPW/ATO) **62**

B.5 Segmentación de pacientes **72**

B.6 Modelo de pricing por simulación de Monte Carlo **76**

B.7 Benchmarking de parámetros de mercado **78**

1. Introducción y motivación

La inasistencia a citas es un problema conocido en el sector sanitario, y el recordatorio por SMS es la herramienta más extendida para combatirlo. Que el recordatorio funciona está demostrado por ensayos clínicos. Lo que no está resuelto es cuánto vale esa mejora para una clínica concreta, ni cómo convertir ese valor en un precio. De esa pregunta nace este trabajo, que sirve de base analítica al proyecto de aiXtensa. El capítulo sitúa primero el problema de la inasistencia y su coste, expone después el contexto empresarial que motiva el análisis y justifica por qué predecir el riesgo no basta y hace falta medir el efecto de la intervención.

1.1. La inasistencia a citas como problema operativo y económico

La inasistencia del paciente a una cita programada, conocida en la literatura como *no-show*, genera un doble coste para el proveedor sanitario. El primero es directo: la cita no se presta y se pierde el ingreso previsto. El segundo es de capacidad. El hueco en agenda rara vez se cubre con otro paciente, de modo que el coste de personal, instalaciones y equipamiento se incurre igualmente, pero sin producción asociada. En una clínica privada pequeña o mediana este segundo efecto pesa más, porque sus estructuras de coste son poco flexibles y sus agendas están ajustadas. Cada hueco vacío es capacidad ociosa que no se recupera al día siguiente.

El fenómeno está documentado en España. Un estudio en las consultas externas de Medicina Preventiva del Hospital Clínico Universitario Lozano Blesa de Zaragoza midió una tasa de absentismo del 12,5% (Hernández-García et al., 2018). La cifra fue del 13,7% en primeras citas y del 11,7% en citas sucesivas. No existe, en cambio, una estadística oficial de ámbito nacional. Ni el Instituto Nacional de Estadística (INE) ni el Ministerio de Sanidad publican una tasa agregada de inasistencia, por lo que la dimensión del problema debe reconstruirse a partir de estudios de centro como el citado. La prensa sectorial sitúa la inasistencia en clínicas privadas entre el 12% y el 19% de las citas, con un coste estimado de entre 2.500 y 7.500 euros mensuales por clínica (El Independiente, 2025). Son cifras divulgativas, sin metodología publicada, y se utilizan aquí solo para presentar el orden de magnitud.

Frente a este problema, el recordatorio por SMS es la intervención de menor coste y mayor escalabilidad disponible, y su eficacia cuenta con respaldo experimental. La

revisión Cochrane de Gurol-Urganci et al. (2013) reúne ocho ensayos aleatorizados y 6.615 participantes. En la comparación de SMS frente a no enviar recordatorio (siete estudios, 5.841 participantes), estima un riesgo relativo de asistencia de 1,14 (IC95% 1,03-1,26). Las tasas absolutas de asistencia fueron del 67,8% sin recordatorio y del 78,6% con SMS. El meta-análisis de Robotham et al. (2016), con 21 estudios, apunta en la misma dirección y estima una reducción de la inasistencia del 21% al 15% (RR 0,75; IC95% 0,68-0,82). La intervención funciona y su efecto está cuantificado en entornos experimentales. La cuestión abierta no es si recordar la cita ayuda, sino cuánto valor genera esa ayuda en un contexto concreto y cómo trasladar parte de ese valor a un precio.

1.2. Contexto empresarial: la cuantificación del valor

Este trabajo es la base analítica de aiXtensa, un proyecto emprendedor promovido por el autor junto a dos socios, orientado a la automatización de procesos administrativos en clínicas privadas españolas, empezando por los recordatorios de cita. La barrera de adopción de este tipo de servicios no es tecnológica sino comercial. El gerente de una clínica no contrata ante la promesa genérica de ahorrar tiempo, sino ante una estimación creíble, en euros, del valor que la automatización recupera.

De esa necesidad nace la pregunta central del trabajo: cuánto valor económico genera un sistema de recordatorios en una clínica privada española y cómo trasladar ese valor a un precio. La respuesta adopta la lógica del *pricing* basado en valor, en la que el precio del servicio se ancla al beneficio que produce en el cliente y no a su coste de provisión. Ese anclaje exige dos piezas. La primera es una estimación rigurosa del efecto de la intervención sobre la asistencia. La segunda es una traducción de ese efecto a euros bajo parámetros realistas del mercado español. El propósito de esta memoria es construir ambas piezas y cuantificar su incertidumbre.

1.3. Por qué predecir no basta

Un modelo predictivo de *no-show* responde a la pregunta de qué pacientes tienen mayor probabilidad de faltar. Es una pieza necesaria, porque permite priorizar a qué pacientes dirigir la intervención y dimensionar el problema de cada clínica. Pero no responde a la pregunta económica relevante: cuántas asistencias adicionales se producen por enviar el recordatorio. El valor del servicio, y con él su precio, depende del efecto de la intervención, no de la probabilidad de que los pacientes falten.

Responder a esa segunda pregunta con datos observacionales exige un tratamiento metodológico específico. En la operativa real el SMS no se asigna al azar, sino que se envía más a unos perfiles de cita que a otros, y esos perfiles difieren también en su probabilidad de faltar. Comparar directamente la asistencia de quienes reciben y no reciben el recordatorio no aísla el efecto del SMS, porque arrastra esas diferencias previas entre los dos grupos. El sesgo puede ser tan fuerte que el recordatorio llegue a parecer perjudicial cuando no lo es. Este trabajo aborda el problema con métodos de inferencia causal para datos observacionales. El estimando, los diagnósticos y las reglas de decisión se definen antes de ejecutar el análisis, y los resultados propios se contrastan con la evidencia experimental publicada. Este diseño de diagnóstico preespecificado es el que sostiene la honestidad del resultado final.

El análisis empírico se apoya en un conjunto de datos público de citas médicas de Vitória (Espírito Santo, Brasil) de 2016, con 106.987 registros tras la limpieza. Se utiliza como proxy conductual del fenómeno de la inasistencia, mientras que los parámetros económicos de la simulación se calibran al mercado español. Esta decisión, sus implicaciones y sus límites se justifican en el capítulo 4 y se discuten en el capítulo 6.

1.4. Estructura del documento

El resto del documento se organiza como sigue. El capítulo 2 formula los objetivos del trabajo. El capítulo 3 revisa el marco teórico en cuatro bloques: los determinantes del *no-show*, la evidencia experimental de los recordatorios SMS, la inferencia causal con datos observacionales y los modelos de *gradient boosting* con técnicas de interpretabilidad. Incluye además una nota sobre *pricing* basado en valor. El capítulo 4 describe La base de datos y la metodología, con las decisiones de control de fugas de información y el diseño del análisis causal. El capítulo 5 presenta los resultados: análisis exploratorio, modelo predictivo, análisis causal, segmentación de pacientes y simulación económica con la propuesta de *pricing.* El capítulo 6 discute los hallazgos y las limitaciones, y el capítulo 7 recoge las conclusiones.

2. Objetivos

Este capítulo formula el objetivo general del trabajo y los objetivos específicos que lo desarrollan, y delimita el alcance de la memoria.

El objetivo general es desarrollar y validar un motor analítico que estime el valor económico de la automatización de recordatorios en clínicas privadas españolas. Sobre esa estimación, el motor debe fundamentar una estrategia de *pricing* basada en valor para aiXtensa.

Este objetivo general se concreta en cuatro objetivos específicos. El primero es predictivo. Consiste en construir un modelo calibrado que estime la probabilidad de *no-show* de cada cita, a partir de las variables disponibles en el momento de la reserva, e identificar sus determinantes con técnicas de interpretabilidad. El segundo es causal. Busca estimar el efecto de los recordatorios SMS sobre la asistencia mediante inferencia causal con datos observacionales. El tercero es descriptivo. Segmenta a los pacientes en perfiles de riesgo diferenciados, caracteriza el potencial económico de cada segmento y explora el efecto del SMS a nivel de clúster, donde los diagnósticos lo permitan. El cuarto es económico. Formula un modelo de *pricing* con componente fija y variable mediante simulación Monte Carlo. De esa simulación se derivan bandas de precios, ancladas a los percentiles P10, P50 y P90 de la distribución de valor.

El proyecto aiXtensa tiene además componentes de naturaleza estratégica: análisis de mercado, diseño del modelo de negocio y prototipos comerciales. En esta memoria se tratan de forma instrumental, no como capítulos propios. Los parámetros del mercado español alimentan la simulación económica del cuarto objetivo. El trabajo se centra en el núcleo analítico del proyecto; el desarrollo de un plan de negocio completo queda fuera de su alcance.

3. Marco teórico y estado del arte

Este capítulo revisa la literatura en la que se apoya el trabajo. Se organiza en cinco bloques. Los dos primeros tratan el problema aplicado: los determinantes de la inasistencia y la evidencia experimental de los recordatorios. Los dos siguientes cubren los métodos: la inferencia causal con datos observacionales y los modelos de *gradient boosting* con técnicas de interpretabilidad. El capítulo cierra con una nota sobre el *pricing* basado en valor, que conecta el análisis con la decisión comercial.

3.1. Determinantes de la inasistencia

La inasistencia a citas médicas cuenta con una literatura amplia, centrada sobre todo en sistemas públicos y hospitalarios. La literatura coincide en varios factores que aparecen de forma recurrente. El primero es la antelación de la cita (lead time): cuanto más tiempo transcurre entre la reserva y la cita, mayor suele ser la probabilidad de falta. El segundo es el historial del paciente, ya que quien ha faltado antes tiende a faltar de nuevo. A estos se añaden la edad, el tipo de cita (primera visita frente a sucesiva) y diversas variables socioeconómicas. La revisión sistemática de Dantas et al. (2018), sobre 105 estudios, confirma este patrón. Los determinantes más reportados son la antelación alta y el historial previo de inasistencia, y la tasa media de absentismo se sitúa en torno al 23%.

Conviene una cautela desde el principio. La mayoría de estos hallazgos son asociaciones, no efectos causales, y dependen del sistema sanitario estudiado. El análisis exploratorio del capítulo 5 confirma estos mismos patrones en La base de datos utilizado, en particular el papel dominante de la antelación. Esa confirmación no convierte las asociaciones en efectos, una distinción que recorre el resto del trabajo.

Identificar estos factores no es un fin en sí mismo. Para una clínica, saber qué citas tienen más riesgo permite dirigir el recordatorio donde más falta hace, en lugar de enviarlo a toda la agenda. Esta idea conecta el modelo predictivo con la segmentación de pacientes del capítulo 5, y con la pregunta de cuánto valor aporta concentrar el esfuerzo en los perfiles de mayor riesgo.

3.2. Evidencia experimental de los recordatorios SMS

A diferencia de los determinantes, el efecto del recordatorio sí cuenta con evidencia experimental. Los ensayos aleatorizados asignan el recordatorio al azar, de modo que el

grupo que lo recibe y el que no son comparables, y la diferencia de asistencia entre ambos puede leerse como efecto causal. Varias revisiones sistemáticas resumen esta evidencia.

Estas revisiones expresan el efecto como un riesgo relativo (RR), el cociente entre la probabilidad del evento en el grupo con recordatorio y en el grupo sin él. Según el evento medido, la dirección cambia: en el riesgo relativo de asistencia un valor mayor que 1 indica mejora, mientras que en el riesgo relativo de no-show un valor menor que 1 indica que las ausencias se reducen. Cada estimación se acompaña de su intervalo de confianza al 95% (IC95%), el rango en el que se sitúa el valor verdadero con esa confianza; cuando el intervalo no incluye el 1, el efecto es estadísticamente significativo.

La revisión Cochrane de Gurol-Urganci et al. (2013) reúne ocho ensayos aleatorizados y 6.615 participantes. En la comparación de SMS frente a no enviar recordatorio, con siete estudios y 5.841 participantes, estima un riesgo relativo de asistencia de 1,14 (IC95% 1,03-1,26), con evidencia de calidad moderada. Las tasas absolutas de asistencia fueron del 67,8% sin recordatorio y del 78,6% con SMS. El mismo trabajo encuentra que el SMS y la llamada telefónica son equivalentes (RR 0,99; IC95% 0,95-1,02), un dato relevante porque el SMS es mucho más barato.

El meta-análisis de Robotham et al. (2016), con 21 estudios, mide la misma magnitud que interesa a este trabajo. Estima una reducción de la inasistencia del 21% al 15%, con un riesgo relativo de *no-show* de 0,75 (IC95% 0,68-0,82). En términos relativos, eso supone reducir las ausencias en torno a una cuarta parte. El estudio observa además que las notificaciones múltiples superan a la notificación única. Otros trabajos apuntan en la misma dirección: Guy et al. (2012) estiman, agrupando los ensayos aleatorizados, una *odds ratio* de asistencia de 1,48 (IC95% 1,23-1,72), si bien no reportan tasas absolutas por brazo. La *odds ratio* (OR) compara las probabilidades en forma de cociente de odds y, por encima de 1, también apunta a mayor asistencia.. Por su parte, Hallsworth et al. (2015) muestran que el contenido del mensaje importa, dado que indicar el coste de la cita en el SMS reduce las ausencias de forma adicional. Este último resultado mide el efecto del redactado, con SMS en ambos brazos, y no el efecto del SMS frente a no enviar nada.

La lectura conjunta es clara. El recordatorio por SMS reduce la inasistencia de forma consistente, con un efecto moderado pero estable a través de contextos. Esta evidencia se

utiliza más adelante como base para el parámetro de efecto de la simulación económica (capítulo 4).

Dos matices acompañan a esta evidencia. El primero es el mecanismo. El recordatorio actúa sobre todo frente al olvido y los cambios de planes, de modo que su efecto tiende a ser mayor cuando la cita se reservó con mucha antelación. El segundo es el contexto. La mayoría de estos estudios proceden de sistemas públicos y hospitalarios de otros países, así que trasladar su efecto a una clínica privada española exige cautela. Esta es una de las razones por las que el trabajo calibra los parámetros económicos al mercado español y trata el efecto como un rango de escenarios.

3.3. Inferencia causal con datos observacionales

Cuando no hay aleatorización, estimar un efecto causal exige un marco formal. El más extendido es el de los resultados potenciales. Para cada cita se definen dos resultados posibles: la asistencia si se envía el SMS y la asistencia si no se envía. Solo se observa uno de los dos, según el tratamiento que de hecho recibió. El efecto medio del tratamiento (ATE) es la diferencia esperada entre ambos resultados en la población. La identificación de este efecto descansa en tres supuestos. El primero es la ausencia de confusión no observada. El segundo es la positividad, es decir, que todo perfil tenga probabilidad no nula de recibir y de no recibir el tratamiento. El tercero es la condición SUTVA, que excluye la interferencia entre unidades (Imbens y Rubin, 2015; Hernán y Robins, 2020).

En términos llanos, el problema es que no se puede observar a la vez lo que ocurre con recordatorio y sin él en la misma cita. La inferencia causal trata de reconstruir esa comparación ausente a partir de citas parecidas en todo salvo en el recordatorio. Si esa reconstrucción es buena, la diferencia de asistencia se acerca a un efecto. Si no lo es, solo refleja las diferencias previas entre los dos grupos.

La herramienta central para ajustar por las diferencias observadas es el propensity score, definido por Rosenbaum y Rubin (1983) como la probabilidad de recibir el tratamiento dadas las covariables. Su resultado clave es que condicionar por este score escalar equilibra la distribución de las covariables entre tratados y controles. El efecto es el mismo que se lograría condicionando por todas ellas a la vez. A partir de él, la ponderación por el inverso de la probabilidad (IPW) construye una pseudopoblación. En

ella el tratamiento es independiente de las covariables observadas. La versión estabilizada del peso reduce la varianza cuando algunas probabilidades son extremas.

Un análisis IPW honesto se sostiene en sus diagnósticos, no en el estimador. El primero es el solapamiento (overlap): las distribuciones del propensity score en tratados y controles deben cubrirse mutuamente. El segundo es el balance, que se mide con la diferencia de medias estandarizada (SMD) de cada covariable tras la ponderación; un umbral habitual es $|SMD| < 0{,}10$. El tercero es el tamaño muestral efectivo (ESS), que indica cuánta información conserva la muestra ponderada. Un ESS muy bajo señala un problema: unos pocos pesos extremos dominan la estimación. Estos diagnósticos pueden revelar que la positividad está violada y que el efecto no es identificable de forma estable, una posibilidad que se materializa en este trabajo (capítulo 5).

Cuando el solapamiento es pobre, una alternativa son los pesos de solapamiento (overlap weights) propuestos por Li, Morgan y Zaslavsky (2018). En lugar de ponderar por el inverso de la probabilidad, ponderan a cada unidad por la probabilidad de pertenecer al grupo contrario. Así dan más peso a las citas con propensity cercano a 0,5. El estimando resultante es el ATO, el efecto medio en la subpoblación de solapamiento. Esta subpoblación se interpreta como las citas para las que el envío del SMS era genuinamente discrecional. El ATO tiene una propiedad atractiva: con propensity score logístico, el balance de covariables es exacto. No es, sin embargo, el ATE, y no debe presentarse como tal.

Por último, la sensibilidad a la confusión no observada se cuantifica con el E-value de VanderWeele y Ding (2017). El E-value indica cuán fuerte tendría que ser un confusor no medido para explicar por completo el efecto estimado. Esa fuerza se mide en su asociación tanto con el tratamiento como con el resultado. Un E-value bajo señala que un confusor moderado bastaría, lo que invita a la prudencia.

3.4. Modelos de gradient boosting e interpretabilidad

El componente predictivo del trabajo emplea *gradient boosting*, en concreto la implementación *XGBoost* de Chen y Guestrin (2016). El método combina muchos árboles de decisión poco profundos, añadidos de forma secuencial, donde cada árbol corrige los errores del conjunto anterior. Ofrece buen rendimiento sobre datos tabulares y gestiona de forma nativa los valores ausentes. Además, admite regularización para controlar el

sobreajuste. Para clasificación de eventos poco frecuentes, como el *no-show*, suele utilizarse una ponderación de la clase positiva que mejora la discriminación.

Un modelo con buena discriminación no produce necesariamente probabilidades fiables. La calibración corrige esa escala. Así, entre las citas a las que el modelo asigna un 30% de riesgo, debe faltar aproximadamente el 30%. Dos técnicas habituales son la regresión logística sobre las puntuaciones (Platt) y la regresión isotónica. La calibración es imprescindible cuando las probabilidades alimentan decisiones económicas posteriores, como ocurre en este trabajo.

La interpretabilidad se aborda con los valores SHAP de Lundberg y Lee (2017). Este método se basa en la teoría de juegos cooperativos. Cada predicción se descompone en la contribución de cada variable, con una atribución que reparte de forma justa el resultado entre los predictores. Los valores SHAP permiten leer tanto la importancia global de cada variable como el efecto local en una cita concreta.

3.5. Nota sobre el pricing basado en valor

El *pricing* basado en valor fija el precio en función del beneficio que el producto genera en el cliente, y no en función de su coste o del precio de los competidores (Hinterhuber, 2008). Aplicado a un servicio de automatización, implica anclar la tarifa al valor económico que el servicio recupera para la clínica. Para que ese anclaje sea defendible se necesita una estimación creíble de ese valor, con su incertidumbre, que es justamente lo que produce la simulación del capítulo 5. Pese a su respaldo teórico, la mayoría de las empresas sigue fijando precios por coste o por competencia, lo que abre una oportunidad para quien sabe cuantificar el valor que entrega (Hinterhuber, 2008).

En la práctica, este enfoque admite combinar una cuota fija con un componente variable ligado al valor recuperado. La parte fija cubre el coste de prestar el servicio y aporta previsibilidad. La parte variable alinea el precio con el resultado, de modo que la clínica paga más solo si recupera más. El capítulo 5 concreta esta estructura en tres tarifas.

4. Datos y metodología

Este capítulo describe el conjunto de datos y el diseño metodológico. Sigue el orden del flujo analítico: limpieza y construcción de variables, modelo predictivo, análisis causal, segmentación, simulación económica y benchmarking de mercado. Se detallan las decisiones para evitar fugas de información y las reglas de decisión fijadas antes de ver los resultados. La Figura 1 resume este flujo.

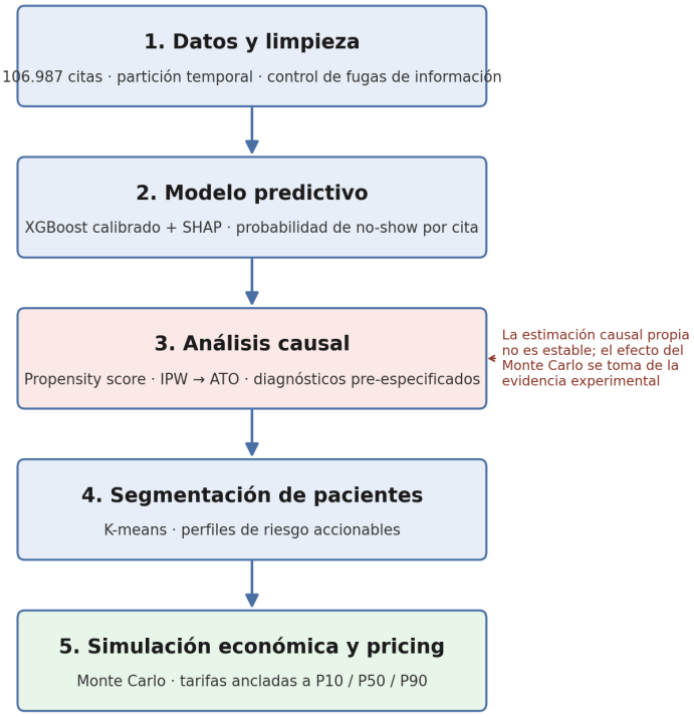

Figura 1. Flujo metodológico del trabajo. Fuente: elaboración propia.

4.1. El conjunto de datos

El análisis empírico utiliza un conjunto de datos público de citas médicas de Vitória (Espírito Santo, Brasil), recogido en 2016 y difundido a través de Kaggle. Tras la limpieza contiene 106.987 citas. Cada registro incluye la fecha de programación y la fecha de la cita, características del paciente (edad, sexo, comorbilidades, beca social), el barrio, el envío o no de recordatorio por SMS y el resultado de asistencia. La variable de tratamiento es la recepción del SMS, y la variable de resultado es la asistencia a la cita.

La base de datos se utiliza como proxy conductual de la inasistencia, no como muestra del mercado español. Las regularidades de comportamiento que captura (el papel de la antelación, del historial o del tipo de cita) son razonablemente transversales entre sistemas

sanitarios. Los parámetros económicos, en cambio, se calibran a España en el capítulo de simulación. Esta elección y sus límites se discuten en el capítulo 6.

La elección de este conjunto responde a tres razones prácticas. Es público y reproducible, tiene un tamaño suficiente para entrenar y validar un modelo, y registra de forma explícita si se envió el recordatorio, que es la variable de tratamiento del estudio. Su principal límite es que procede de un sistema sanitario distinto. Por eso el análisis separa el comportamiento, que se modela sobre estos datos, de los parámetros económicos, que se calibran a España.

4.2. Limpieza y construcción de variables

La preparación de los datos persigue dos objetivos: corregir anomalías y construir variables informativas sin filtrar información del futuro. Se documentaron cinco citas con fecha de cita anterior a la de programación, que se excluyeron del cálculo de las variables de historial. Se detectaron también cinco identificadores de paciente con parte fraccionaria, que se trataron de forma controlada tras verificar que no colisionaban con identificadores enteros. La variable de resultado se codifica como *no-show* (1 si el paciente no asiste), y su correcta orientación se verifica como primer paso de cada notebook de modelado.

La partición entre entrenamiento y prueba es temporal, no aleatoria. Como los pacientes se repiten, una partición aleatoria filtraría información de un paciente entre ambos conjuntos. El corte se fija en la fecha de cita del 1 de junio de 2016, lo que deja un 80,1% de las citas para entrenamiento y un 19,9% para prueba (unas 21.330 citas). Las variables de historial del paciente se calculan con una regla de tiempo de decisión: solo se usan citas previas cuya fecha de cita es anterior a la fecha de programación de la cita actual. Así, ninguna variable incorpora información no disponible en el momento de la reserva. El barrio se codifica por frecuencia, nunca por su tasa de *no-show*, para no introducir el resultado en los predictores.

Una consecuencia de la ventana temporal merece mención. La base de datos cubre unas seis semanas, de modo que el 72% de las citas figuran como primera visita observada. Esa etiqueta significa primera visita dentro del periodo, no primera visita en términos absolutos. Es un caso de censura por la izquierda del historial, que se tiene en cuenta al interpretar las variables de historial y su atribución SHAP.

4.3. Modelo predictivo

El modelo de riesgo de *no-show* es un *XGBoost* con 18 variables. La variable de tratamiento (recepción del SMS) se excluye de forma deliberada, ya que su relación con el resultado se estima por separado en el análisis causal; incluirla contaminaría el modelo predictivo. Para compensar el desequilibrio de clases se utiliza una ponderación de la clase positiva. Esta ponderación mejora la discriminación, pero distorsiona la escala de probabilidad. Por eso las probabilidades que se usan después en segmentación y simulación se obtienen tras calibrar el modelo en un subconjunto separado.

Las variables se agrupan en tres bloques. El primero recoge el calendario de la cita, como la antelación y el día y el mes. El segundo describe al paciente, con la edad, el sexo y las comorbilidades. El tercero resume su historial, como el número de citas previas y su tasa de inasistencia anterior. Ninguna de estas variables usa información posterior al momento de la reserva.

Los hiperparámetros se ajustan mediante una rejilla de nueve combinaciones (profundidad máxima por tasa de aprendizaje), con validación cruzada *walk-forward* de tres particiones (*folds*) de ventana expansiva dentro del periodo de entrenamiento y parada temprana. La calibración compara la regresión de Platt y la isotónica sobre el subconjunto de calibración, y se elige por el Brier Score. La combinación ganadora y los resultados se reportan en el capítulo 5.

4.4. Diseño del análisis causal

El estimando de interés es el efecto medio del SMS sobre la asistencia, expresado en puntos porcentuales absolutos. La convención de signo es explícita: el resultado es la asistencia, y un signo positivo indica que el SMS se asocia con más asistencia tras el ajuste. El *propensity* score se modela con regresión logística, con términos cuadráticos en la antelación, la edad y la tasa de *no-show* previa, y con codificación por frecuencia del barrio. A partir de él se calculan pesos IPW estabilizados.

Las reglas de decisión se fijan antes de ejecutar el análisis. Se evalúan tres diagnósticos: el solapamiento de las distribuciones del propensity score, el balance de las 21 covariables mediante SMD y el tamaño muestral efectivo por brazo. La regla de recorte está pre-especificada: si el ESS sobre el tamaño muestral cae por debajo de 0,5 en algún brazo, se recortan los pesos extremos. Se contempla también una contingencia pre-especificada: si

el IPW no logra balance, se pasa al estimando ATO con pesos de solapamiento (Li, Morgan y Zaslavsky, 2018). La incertidumbre se cuantifica con un bootstrap por conglomerados de paciente (1.000 iteraciones), apropiado porque los pacientes se repiten. La robustez se evalúa con un análisis de sensibilidad a la especificación de la antelación y con el E-value.

Fijar estas reglas antes de ver los resultados cumple una función concreta. Evita elegir, a posteriori, la especificación que da el número más conveniente. En un análisis causal con datos observacionales esa tentación es real, porque pequeñas decisiones técnicas pueden mover el resultado. Dejar las reglas por escrito de antemano hace que el análisis sea más transparente y fácil de defender.

4.5. Segmentación de pacientes

La segmentación agrupa las citas del conjunto de prueba mediante K-means. Utiliza cuatro variables. La probabilidad calibrada de *no-show* mide el riesgo. La antelación discretizada, el barrio y el indicador de primera visita aportan accionabilidad. El número de grupos se elige combinando el método del codo, la silueta y la interpretabilidad de los perfiles. No se estima un efecto causal por clúster, ya que la violación de positividad observada a nivel global no se resuelve dentro de los subgrupos. La comparación de asistencia entre quienes reciben y no reciben SMS dentro de cada clúster se reporta como descriptiva y confundida, no como efecto.

4.6. Simulación económica y parámetro de efecto

El valor económico se estima con una simulación Monte Carlo de 10.000 iteraciones. El parámetro de efecto se modela como una reducción relativa de la tasa de *no-show*, mediante una distribución triangular de escenarios informada por la evidencia experimental publicada. Los vértices son una reducción del 18%, del 25% y del 33,5%. El mínimo recoge el extremo conservador del intervalo de Robotham et al. (2016); el central, su estimación puntual (RR 0,75); y el máximo, la diferencia entre brazos de la revisión Cochrane (Gurol-Urganci et al., 2013). La triangular es una elección simple y transparente para representar la incertidumbre comercial del parámetro, no una distribución estimada estadísticamente.

La ganancia de cada iteración se obtiene multiplicando la tasa basal de *no-show* de la clínica por la reducción relativa. La tasa basal es un parámetro de entrada del modelo, con

un rango de sensibilidad del 10% al 20%. El valor de referencia es del 12,5% (Hernández-García et al., 2018), que no debe leerse como tasa nacional. Esta multiplicación no contradice la regla de no multiplicar el efecto por la tasa basal. Aquella regla prohíbe multiplicar un efecto ya expresado en puntos porcentuales absolutos. Aquí el parámetro es relativo por definición, de modo que la multiplicación es la conversión correcta a puntos porcentuales. El resto de parámetros (valor de la consulta, coste del SMS, volumen mensual y margen de contribución) procede del benchmarking de mercado y se presenta en el capítulo 5. El modelo de pricing combina una cuota fija y un componente variable sobre el valor recuperado, con bandas ancladas a los percentiles P10, P50 y P90 de la distribución de valor.

La simulación Monte Carlo repite el cálculo del valor muchas veces. En cada repetición toma un valor distinto de cada parámetro dentro de su rango. El resultado no es una cifra única, sino una distribución de valores posibles. Los percentiles resumen esa distribución, ya que el P10 marca un escenario prudente, el P50 el central y el P90 uno favorable. Anclar las tarifas a esos tres puntos traslada la incertidumbre del análisis a la propuesta comercial.

La razón de no usar la estimación causal propia como parámetro es metodológica. Esa estimación no es estable (capítulo 5), de modo que elegir una especificación concreta equivaldría a elegir el resultado conveniente. La evidencia experimental, en cambio, no sufre la confusión por indicación de estos datos. El precio de esta decisión es el transporte de evidencia de otros sistemas sanitarios a España, una limitación que se declara de forma explícita

4.7. Benchmarking de mercado

Los parámetros económicos españoles se obtienen mediante una recogida documental de tarifas públicas, sin extracción programática, para preservar la reproducibilidad. Cada precio se acompaña de su fuente, su URL y la fecha de consulta. Se recopilan precios de consulta privada y costes unitarios de SMS de proveedores españoles, que se triangulan en las distribuciones de la simulación.

Cuando una misma magnitud aparece en varias fuentes, se toma un rango en lugar de un único dato. Así, la simulación no depende de un proveedor ni de una tarifa concretos.

5. Resultados

Este capítulo presenta los resultados en el orden del flujo analítico. Empieza por el análisis exploratorio, sigue con el modelo predictivo y el análisis causal, y termina con la segmentación, la simulación económica y la propuesta de pricing. El hilo conductor es la distinción entre asociación y efecto, que aparece ya en los datos en crudo y condiciona todo lo demás.

5.1. Análisis exploratorio

La tasa global de *no-show* en La base de datos es del 20,26%. El hallazgo principal del análisis exploratorio es contraintuitivo. Los pacientes que reciben SMS faltan más, no menos. La tasa de *no-show* es del 16,73% entre quienes no reciben recordatorio y del 27,67% entre quienes sí lo reciben, una diferencia de 10,94 puntos porcentuales en la dirección opuesta a la esperada (Figura 2).

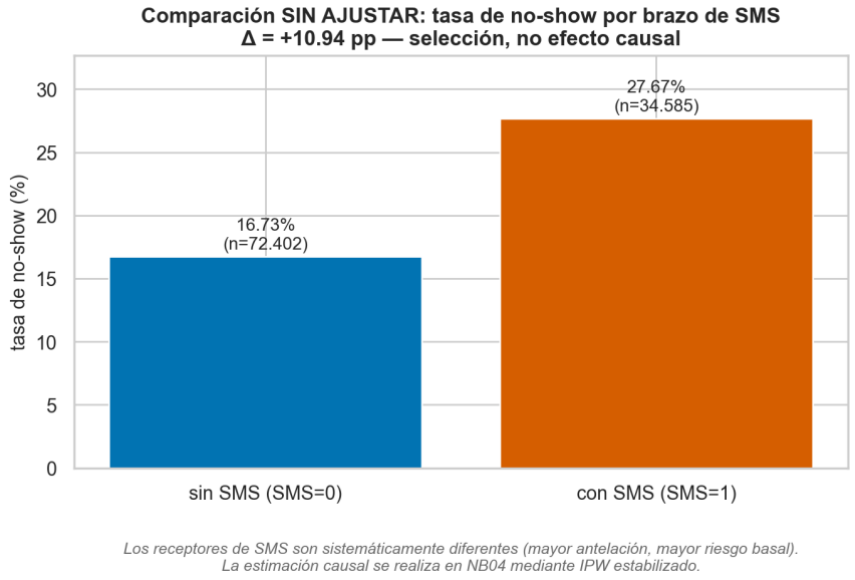

Figura 2. Tasa de no-show por brazo de SMS, sin ajustar. Fuente: elaboración propia.

La explicación es el sesgo de selección. El SMS no se envía al azar, sino que se concentra en citas de mayor antelación y mayor riesgo basal. La correlación entre la recepción del SMS y la antelación es de 0,40, la más alta de todas las variables. El recordatorio se asignó, por tanto, a las citas con más probabilidad de fallar (Figura 3). La comparación directa entre brazos mezcla la asociación del SMS con esas diferencias previas, lo que explica el signo invertido. Este patrón es la motivación cualitativa del análisis causal: sin ajustar por la antelación, cualquier estimación del efecto está sesgada.

Este patrón obliga a separar dos preguntas que es fácil confundir. Una es quién falta más, que se responde con los datos en crudo. Otra es cuánto cambia la asistencia por enviar el recordatorio, que es la que importa para el valor y que exige ajustar por las diferencias previas. El resto del capítulo aborda las dos por separado.

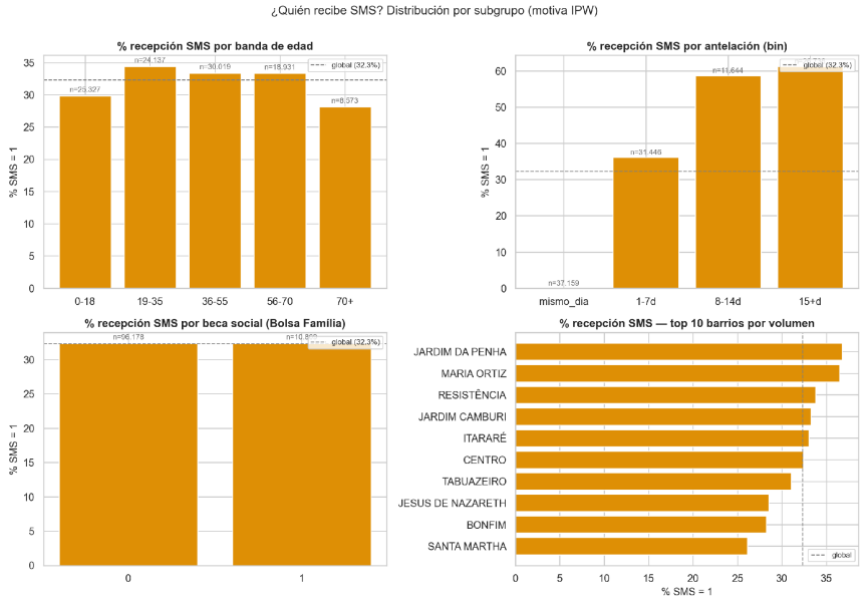

Figura 3. Asignación del SMS según características de la cita. Fuente: elaboración propia.

5.2. Modelo predictivo

El AUC-ROC mide la capacidad del modelo de ordenar los casos por riesgo: es la probabilidad de asignar mayor riesgo a una cita que acaba en falta que a una que no, donde 0,5 equivale al azar y 1 al acierto perfecto. El PR-AUC, área bajo la curva de precisión-exhaustividad, es más informativo cuando la clase positiva es minoritaria, como aquí; su referencia de no-información es la propia prevalencia, 0,186, de modo que el valor obtenido casi la duplica. El Brier Score es el error cuadrático medio entre la probabilidad predicha y el resultado observado, por lo que un valor más bajo indica probabilidades más fiables.

El modelo XGBoost calibrado alcanza, en el conjunto de prueba, un AUC-ROC de 0,730 y un PR-AUC de 0,329, sobre una prevalencia de *no-show* del 18,6%. La capacidad de discriminación es moderada, coherente con la dificultad intrínseca de predecir un comportamiento individual. La calibración mejora de forma sustancial la fiabilidad de las probabilidades: el Brier Score pasa de 0,217 sin calibrar a 0,137 tras la calibración

isotónica, una reducción del 37% (Figura 4). La combinación de hiperparámetros ganadora es una profundidad máxima de 4 y una tasa de aprendizaje de 0,1.

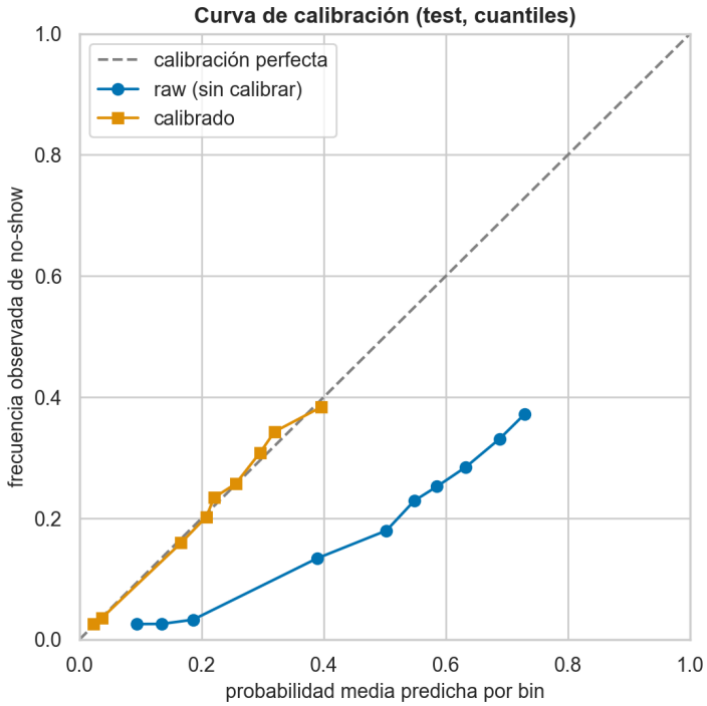

Figura 4. Curva de calibración del modelo antes y después del ajuste. Fuente: elaboración propia.

El análisis SHAP muestra un predictor dominante. La antelación concentra la mayor importancia media (0,84), muy por encima de la edad (0,22) y de la tasa de no-show previa (0,08). El resto de variables aporta una señal menor. Este resultado es consistente con la literatura y con el análisis exploratorio, y anticipa el problema causal: la variable que mejor predice el *no-show* es también la que gobierna la asignación del SMS (Figura 5).

Conviene leer estas cifras con sentido práctico. Predecir si una persona concreta acudirá a su cita es difícil, y por eso la capacidad de discriminación es moderada y no alta. Aun así, el modelo ordena bien el riesgo, que es lo que necesita una clínica para priorizar a quién recordar. La calibración añade un segundo uso. Como las probabilidades son fiables, pueden alimentar después la simulación económica sin distorsionar el resultado.

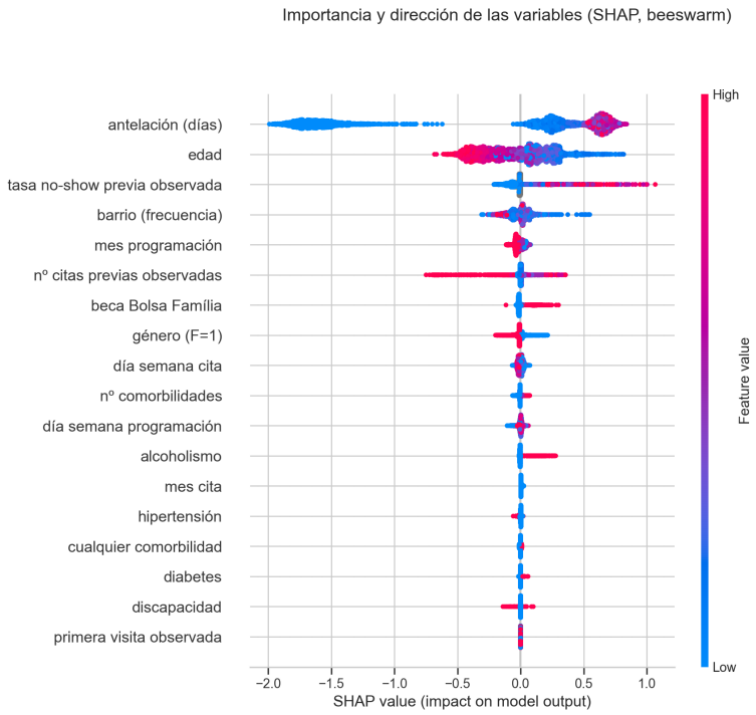

Figura 5. Importancia de variables según valores SHAP. Fuente: elaboración propia.

5.3. Análisis causal

El modelo de propensity alcanza un AUC de 0,849. Una capacidad tan alta para predecir quién recibe el SMS es, en este contexto, problemática, porque indica que la asignación es casi determinista en función de la antelación. El examen del solapamiento lo confirma, ya que las distribuciones del propensity score en tratados y controles apenas se cubren (Figura 6). La positividad, supuesto necesario para identificar el efecto, está estructuralmente comprometida.

El IPW estabilizado fracasa en sus diagnósticos, tal y como anticipaba la regla pre-especificada. El tamaño muestral efectivo del brazo tratado cae al 0,1% de su tamaño, muy por debajo del umbral de 0,5 que dispara el recorte. Y el ajuste no consigue equilibrar los grupos, porque la antelación sigue casi tan desbalanceada después como antes. El efecto estimado es además muy inestable según el recorte aplicado, y oscila entre -10,4 y -5,8 puntos porcentuales (Tabla 1). Estos números no se interpretan como efecto, sino como señal de que el IPW no es viable con estos datos.

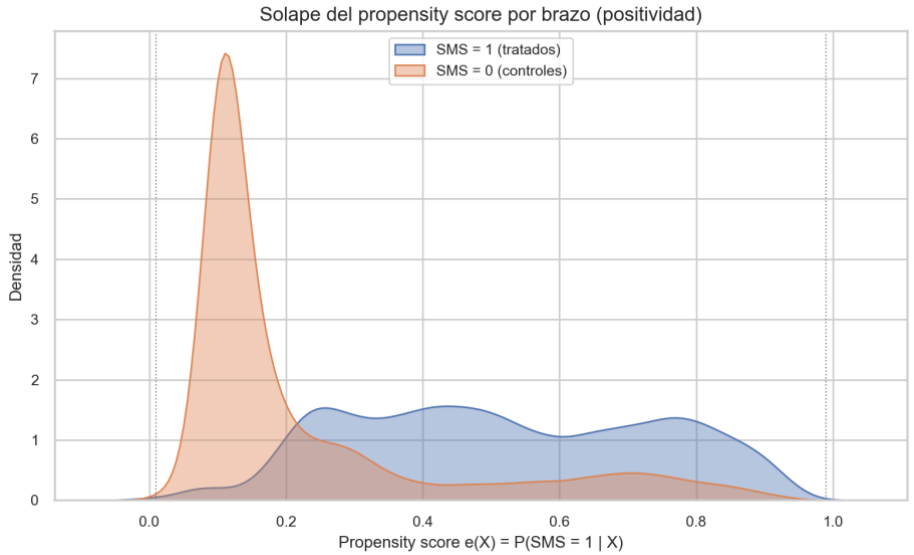

Figura 6. Solapamiento de las distribuciones del propensity score. Fuente: elaboración propia.

Especificación	Citas usadas	Efecto estimado (pp)
Sin recorte	106.987	-10,43
Recorte 1-99 (preferida)	104.847	-5,77
Recorte 5-95	96.287	-8,72

Tabla 1. Estimación IPW del efecto según especificación de recorte. Fuente: elaboración propia.

La contingencia pre-especificada conduce al estimando ATO, con pesos de solapamiento. El cambio es notable, porque ahora el balance entre grupos sí se logra y el tamaño muestral efectivo recupera valores sanos. El ATO estimado es de -1,24 puntos porcentuales, con un intervalo de confianza del 95% de [-1,92, -0,51]. La estimación ATO sugiere, por tanto, que el SMS se asocia con una asistencia ligeramente menor en la subpoblación de solapamiento, tras ajustar por las características observadas. Esa estimación es además frágil, ya que bastaría un factor de confusión moderado no observado para explicarla por completo (un E-value bajo, de 1,15).

El análisis de sensibilidad es determinante. Al cambiar la forma de medir la antelación, de continua a categórica por tramos, el ATO cambia de signo. Pasa de -1,24 a +5,40 puntos porcentuales, con un intervalo de [+4,64, +6,21]. Las dos estimaciones tienen signos opuestos y sus intervalos no se solapan (Figura 7). La regla de robustez pre-especificada exige la misma dirección y el solape de los intervalos. El resultado no cumple ninguna de las dos condiciones, de modo que no es robusto. Se concluye que la evidencia observacional disponible no permite una estimación estable del efecto del SMS. Por ese

motivo la simulación económica se apoya en la evidencia experimental y no en esta estimación.

En lenguaje claro, el análisis dice algo incómodo pero útil. Los datos no permiten saber, por sí solos, cuánto ayuda el recordatorio en esta población, porque casi siempre se envió a las mismas citas. Forzar una cifra sería elegir la que más conviene. Reconocer el límite es más honesto y, además, orienta la decisión de negocio hacia medir el efecto de forma controlada en cada clínica.

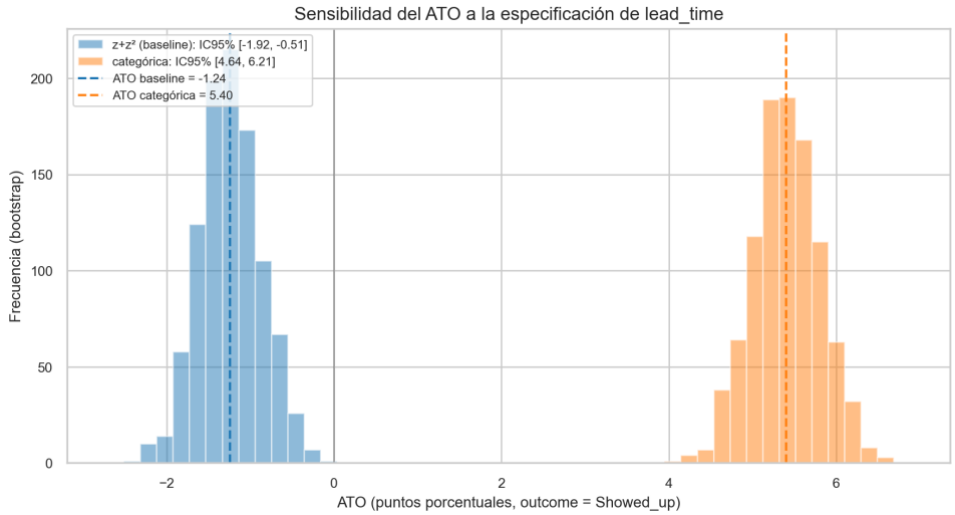

Figura 7. Sensibilidad del ATO a la especificación de la antelación. Fuente: elaboración propia.

5.4. Segmentación de pacientes

La segmentación retiene tres grupos, elegidos por el método del codo y la interpretabilidad, con una silueta de 0,38 (Figura 8). Los perfiles resultantes son diferenciados e interpretables (Tabla 2). El primer grupo reúne citas de bajo riesgo, casi todas del mismo día, con una tasa de *no-show* observada del 3,5% y una presencia de SMS prácticamente nula. Los otros dos grupos son de alto riesgo. Uno está formado por primeras visitas con alta antelación, con un 27,2% de *no-show* y un 72,0% de cobertura de SMS. El otro reúne pacientes recurrentes con antelación media, con un 25,3% de *no-show* y un 59,5% de SMS.

Clúster	Citas	Antelación media	No-show obs.	% SMS	Perfil
C0	7.228	0,0 días	3,5%	0,4%	Bajo riesgo, mismo día
C1	8.029	21,3 días	27,2%	72,0%	Alto riesgo, primera visita
C2	6.073	8,5 días	25,3%	59,5%	Alto riesgo, recurrente

Tabla 2. Perfil de los tres clústeres de citas. Fuente: elaboración propia.

La lectura operativa es directa. Los dos grupos de alto riesgo concentran en torno al 66% de la agenda y el 93,7% de las inasistencias esperadas. Es ahí donde una intervención de recordatorio tiene sentido económico. Un envío de SMS uniforme a toda la agenda resulta, por tanto, subóptimo. La proyección de los clústeres sobre las dos primeras componentes principales muestra una separación coherente con los perfiles (Figura 9). La comparación de asistencia entre quienes reciben y no reciben SMS dentro de cada clúster no se interpreta como efecto, por la misma falta de solapamiento del análisis global.

El grupo de bajo riesgo apenas necesita intervención, ya que casi todas sus citas se cumplen y son del mismo día. La consecuencia para el servicio es clara. El recordatorio rinde más si se concentra en los dos perfiles de alto riesgo, que es donde se acumulan las ausencias y, con ellas, el valor recuperable.

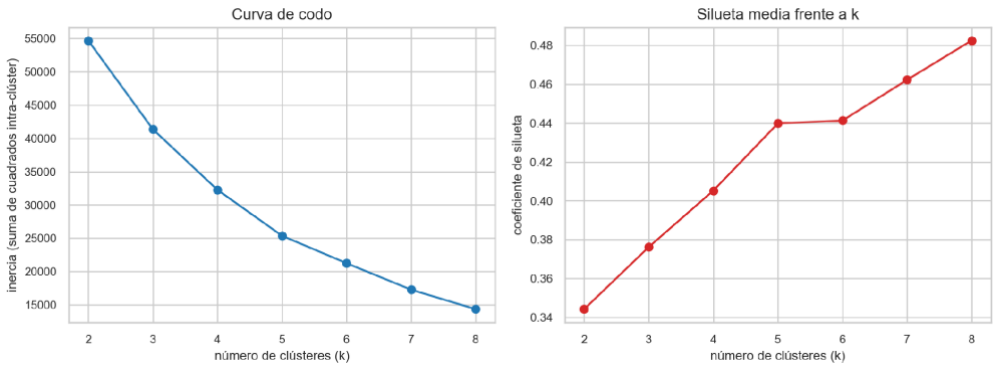

Figura 8. Selección del número de clústeres. Fuente: elaboración propia.

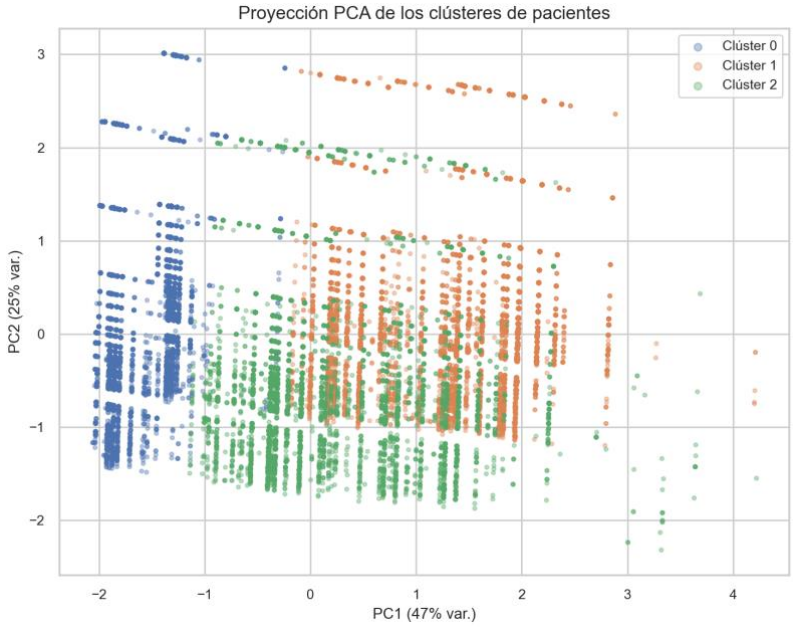

Figura 9. Proyección de los clústeres sobre componentes principales. Fuente: elaboración propia.

5.5. Simulación económica

La simulación Monte Carlo, con 10.000 iteraciones y el parámetro de efecto tomado de la evidencia experimental, produce una distribución de valor neto mensual recuperado por clínica. La mediana es de 1.304 euros, con un percentil P10 de 659 euros y un P90 de 2.318 euros (Figura 10). Ninguna de las 10.000 iteraciones resulta en valor negativo, lo que sugiere que, bajo los parámetros considerados, el servicio recuperaría más de lo que cuesta en todo el rango simulado. La dispersión refleja la incertidumbre real sobre el tamaño de la clínica, su tasa basal y la magnitud del efecto.

La forma de la distribución importa tanto como su centro. El resultado no es un número único, sino un rango, porque cada clínica tiene un tamaño, una tasa de inasistencia y un margen distintos. Una clínica pequeña se situará cerca del extremo bajo, y una grande cerca del alto. Presentar el valor como un rango, y no como una cifra fija, encaja con la incertidumbre real del caso.

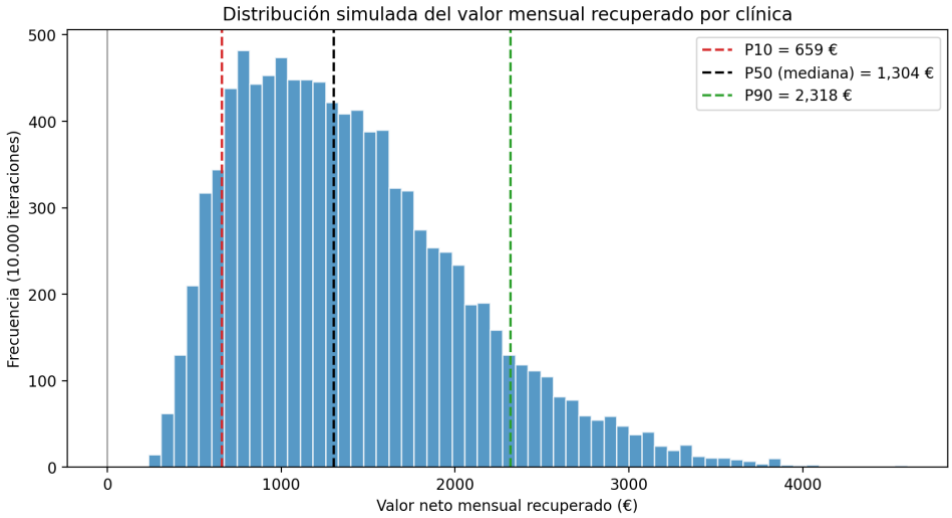

Figura 10. Distribución simulada del valor neto mensual recuperado por clínica. Fuente: elaboración propia.

5.6. Propuesta de pricing

La propuesta de pricing traslada la distribución de valor a tres tarifas, cada una con una cuota fija y un componente variable sobre el valor recuperado, ancladas a los tres percentiles (Tabla 3). El plan conservador se ancla al P10, con una cuota de 132 euros y un 10% variable. El plan recomendado se ancla a la mediana, con 261 euros y un 15%. El plan premium se ancla al P90, con 464 euros y un 20%. Esta estructura captura una parte del valor generado y mantiene la tarifa por debajo del valor recuperado en todos los escenarios.

Cada tarifa encaja con un perfil de clínica. El plan conservador se ajusta a centros pequeños o con poca inasistencia, donde el valor recuperable es menor. El plan recomendado sirve de referencia para una clínica media. El plan premium se reserva para centros grandes, con mucho volumen, donde el valor recuperado es mayor. En los tres casos la cuota se mantiene por debajo del valor esperado, de modo que la propuesta es sostenible para el cliente.

Plan	Ancla (valor neto)	Cuota fija	Variable
Conservador	P10 (659 €)	132 €/mes	10%
Recomendado	P50 (1.304 €)	261 €/mes	15%
Premium	P90 (2.318 €)	464 €/mes	20%

Tabla 3. Tarifas propuestas ancladas a los percentiles de la distribución de valor. Fuente: elaboración propia.

Los parámetros económicos proceden del benchmarking de mercado (Tabla 4). El valor de la consulta privada se sitúa entre 60 y 160 euros según especialidad y proveedor. El coste unitario del SMS oscila entre 0,034 y 0,096 euros según el volumen contratado (Figura 11). El volumen mensual de citas y el margen de contribución se tratan como supuestos declarados. Sobre ellos se realiza el análisis de sensibilidad de la simulación.

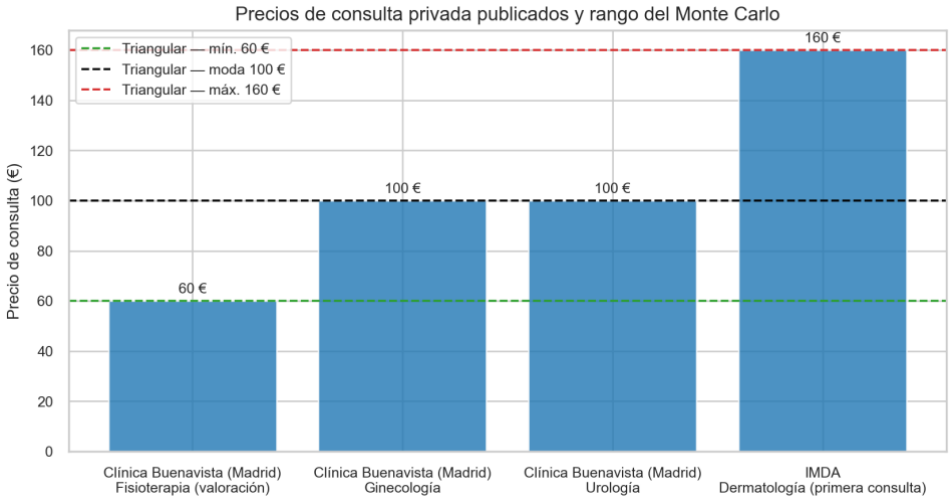

Figura 11. Precios de consulta privada observados. Fuente: elaboración propia a partir de tarifas públicas.

Parámetro	Mínimo	Moda	Máximo	Tipo
Reducción relativa del no-show	18%	25%	33,5%	Verificado
Tasa basal de no-show	10%	12,5%	20%	Verificado
Valor de la consulta	60 €	100 €	160 €	Verificado
Coste del SMS	0,034 €	0,045 €	0,096 €	Verificado
Volumen mensual	200	—	800	Supuesto
Margen de contribución	60%	75%	90%	Supuesto

Tabla 4. Parámetros del modelo de simulación y su origen. Fuente: elaboración propia.

6. Discusión y limitaciones

Los resultados dibujan un arco coherente. En los datos en crudo, el SMS parece empeorar la asistencia, un artefacto del sesgo de selección. El análisis predictivo confirma que la antelación domina tanto el riesgo de *no-show* como, implícitamente, la asignación del recordatorio. El análisis causal intenta corregir ese sesgo, pero sus diagnósticos revelan que la positividad está estructuralmente rota: el SMS se asignó de forma casi determinista, y no existe el solapamiento necesario para comparar. El estimando de contingencia logra balance, pero no resiste un cambio razonable de especificación. La lectura final no es un número, sino una conclusión metodológica: con estos datos observacionales, el efecto del SMS no es identificable de forma estable.

Lejos de ser un fracaso, este resultado es lo que el diseño pretendía proteger. Las reglas de diagnóstico, recorte, contingencia y sensibilidad se fijaron antes de ver los datos y se siguieron. El sistema funcionó: evitó presentar como causal una cifra que no lo es. Lejos de ser un fracaso, este resultado es lo que el diseño pretendía proteger. Las reglas de diagnóstico, recorte, contingencia y sensibilidad se fijaron antes de ver los datos y se siguieron. El sistema funcionó: evitó presentar como causal una cifra que no lo es. De ahí se deriva una implicación práctica para aiXtensa. La medición experimental, mediante pruebas A/B controladas, debería formar parte del propio servicio, porque es la única vía para estimar el efecto real en cada clínica.

Este resultado tiene un valor que va más allá del caso concreto. Muchos análisis con datos observacionales en gestión sanitaria comparan grupos que no son comparables y presentan la diferencia como si fuera un efecto. El trabajo muestra, con un ejemplo real, por qué esa práctica es arriesgada y cómo unos diagnósticos sencillos la detectan a tiempo.

El trabajo tiene varias limitaciones que conviene declarar. La primera es el uso de un dataset brasileño como proxy conductual de un caso de negocio español, lo que obliga a calibrar los parámetros económicos por separado y a asumir que las regularidades de comportamiento son transportables. La segunda es la posible confusión no observada, subrayada por un E-value bajo en la estimación ATO. La tercera es la propia violación de positividad, que impide una estimación causal estable y que ningún estimador resuelve por completo. La cuarta es la censura por la izquierda del historial, derivada de la ventana

temporal de seis semanas. La quinta es el supuesto SUTVA, que podría tensionarse si hubiera interferencia entre citas de un mismo paciente.

Algunas de estas limitaciones se podrían reducir con datos propios. Un registro de citas de clínicas españolas, con la misma información, permitiría sustituir el proxy por datos del mercado objetivo. Una asignación aleatoria del recordatorio, aunque fuera parcial, resolvería el problema de fondo y haría identificable el efecto.

Por último, el parámetro de efecto de la simulación se apoya en evidencia experimental de otros sistemas sanitarios, con el consiguiente riesgo de transporte a España. Una línea de trabajo futura es el uso de métodos de doble aprendizaje automático (double machine learning), que ofrecen robustez frente a la mala especificación (Chernozhukov et al., 2018). No obstante, esos métodos no resuelven una violación estructural de positividad como la observada aquí, de modo que su aportación sería limitada en este caso concreto.

7. Conclusiones

Este capítulo sintetiza las conclusiones del trabajo y cierra con su implicación principal.

El trabajo se propuso estimar el valor económico de la automatización de recordatorios en clínicas privadas españolas y fundamentar sobre él una estrategia de pricing basado en valor. Sus cuatro objetivos específicos se cumplen en distinto grado. El predictivo se alcanza: el modelo XGBoost calibrado ordena el riesgo de *no-show* con una capacidad moderada (AUC de 0,730) e identifica la antelación de la cita como predictor dominante. El descriptivo se cumple de forma exploratoria, con tres perfiles de cita en los que el riesgo y las ausencias se concentran en dos grupos de alto riesgo. El económico se alcanza con la simulación y la propuesta de tarifas. El causal, en cambio, topa con un límite metodológico que marca el resto del trabajo.

Ese límite es el hallazgo central. El recordatorio se envió de forma casi determinista a las citas de mayor antelación y riesgo, de modo que los grupos con y sin SMS no son comparables y la positividad está estructuralmente rota. Ni el IPW ni el estimando de contingencia producen una estimación estable, porque el efecto ajustado cambia de signo ante un cambio razonable de especificación. La conclusión no es una cifra, sino un límite: con los datos observacionales disponibles, el efecto del SMS no es identificable de forma fiable.

Ante esa limitación, la simulación económica se apoyó en la evidencia experimental publicada. El modelo Monte Carlo sitúa el valor neto mensual recuperado por clínica en una mediana de 1.304 euros, sin ninguna iteración negativa, y sobre esa distribución se anclan tres tarifas a los percentiles P10, P50 y P90. De aquí se desprende la principal implicación para aiXtensa: la medición del efecto no puede heredarse de datos ajenos, sino que debe incorporarse al propio servicio mediante la asignación aleatoria del recordatorio en cada clínica.

Las limitaciones del trabajo y sus líneas de continuación se han desarrollado en el capítulo anterior. Todas convergen en la misma idea con la que cierra el trabajo: cuantificar el valor exige primero poder medir el efecto, y eso solo se consigue diseñando la medición desde el principio.

Bibliografía

Chen, T., y Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785-794.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W., y Robins, J. (2018). Double/debiased machine learning for treatment and structural parameters. *The Econometrics Journal*, 21(1), C1-C68.

Dantas, L. F., Fleck, J. L., Cyrino Oliveira, F. L., y Hamacher, S. (2018). No-shows in appointment scheduling: a systematic literature review. *Health Policy*, 122(4), 412-421.

El Independiente (2025, 29 de septiembre). [Reportaje sobre inasistencia y coste en clínicas privadas]. *El Independiente*.

Gurol-Urganci, I., de Jongh, T., Vodopivec-Jamsek, V., Atun, R., y Car, J. (2013). Mobile phone messaging reminders for attendance at healthcare appointments. *Cochrane Database of Systematic Reviews*, 2013(12), CD007458.

Guy, R., Hocking, J., Wand, H., Stott, S., Ali, H., y Kaldor, J. (2012). How effective are short message service reminders at increasing clinic attendance? A meta-analysis and systematic review. *Health Services Research*, 47(2), 614-632.

Hallsworth, M., Berry, D., Sanders, M., Sallis, A., King, D., Vlaev, I., y Darzi, A. (2015). Stating appointment costs in SMS reminders reduces missed hospital appointments: Findings from two randomised controlled trials. *PLoS ONE*, 10(9), e0137306.

Hernán, M. A., y Robins, J. M. (2020). *Causal Inference: What If*. Boca Raton: Chapman & Hall/CRC.

Hinterhuber, A. (2008). Customer value-based pricing strategies: why companies resist. *Journal of Business Strategy*, 29(4), 41-50.

Hernández-García, I., Chaure-Pardos, A., Moliner-Lahoz, J., et al. (2018). Absentismo, y factores asociados, en las citas programadas de una consulta externa de Medicina Preventiva. *Journal of Healthcare Quality Research*, 33(2), 82-87.

Imbens, G. W., y Rubin, D. B. (2015). *Causal Inference for Statistics, Social, and Biomedical Sciences*. Cambridge: Cambridge University Press.

Li, F., Morgan, K. L., y Zaslavsky, A. M. (2018). Balancing covariates via propensity score weighting. *Journal of the American Statistical Association*, 113(521), 390-400.

Lundberg, S. M., y Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30.

Robotham, D., Satkunanathan, S., Reynolds, J., Stahl, D., y Wykes, T. (2016). Using digital notifications to improve attendance in clinic: Systematic review and meta-analysis. *BMJ Open*, 6(10), e012116.

Rosenbaum, P. R., y Rubin, D. B. (1983). The central role of the propensity score in observational studies for causal effects. *Biometrika*, 70(1), 41-55.

VanderWeele, T. J., y Ding, P. (2017). Sensitivity analysis in observational research: introducing the E-value. *Annals of Internal Medicine*, 167(4), 268-274.

Anexo A. Declaración de uso de herramientas de inteligencia artificial generativa

ADVERTENCIA: Desde la Universidad consideramos que ChatGPT u otras herramientas similares son herramientas muy útiles en la vida académica, aunque su uso queda siempre bajo la responsabilidad del alumno, puesto que las respuestas que proporciona pueden no ser veraces. En este sentido, NO está permitido su uso en la elaboración del Trabajo fin de Grado para generar código porque estas herramientas no son fiables en esa tarea. Aunque el código funcione, no hay garantías de que metodológicamente sea correcto, y es altamente probable que no lo sea.

Por la presente, yo, Sergio Cuervo-Arango Sala, estudiante de ICADE de la Universidad Pontificia Comillas al presentar mi Trabajo Fin de Grado titulado "Predicción de valor y estrategia de pricing para aiXtensa ", declaro que he utilizado la herramienta de Inteligencia Artificial Generativa ChatGPT u otras similares de IAG de código sólo en el contexto de las actividades descritas a continuación:

1. **Brainstorming de ideas de investigación:** Utilizado para idear y esbozar posibles áreas de investigación.
2. **Crítico:** Para encontrar contra-argumentos a una tesis específica que pretendo defender.
3. **Referencias:** Usado conjuntamente con otras herramientas, como Science, para identificar referencias preliminares que luego he contrastado y validado.
4. **Metodólogo:** Para descubrir métodos aplicables a problemas específicos de investigación.
5. **Interpretador de código:** Para realizar análisis de datos preliminares.
6. **Corrector de estilo literario y de lenguaje:** Para mejorar la calidad lingüística y estilística del texto.
7. **Generador previo de diagramas de flujo y contenido:** Para esbozar diagramas iniciales.
8. **Sintetizador y divulgador de libros complicados:** Para resumir y comprender literatura compleja.
9. **Revisor:** Para recibir sugerencias sobre cómo mejorar y perfeccionar el trabajo y código con diferentes niveles de exigencia.

Afirmo que toda la información y contenido presentados en este trabajo son producto de mi investigación y esfuerzo individual, excepto donde se ha indicado lo contrario y se han dado los créditos correspondientes (he incluido las referencias adecuadas en el TFG y he explicitado para que se ha usado ChatGPT u otras herramientas similares). Soy consciente de las implicaciones académicas y éticas de presentar un trabajo no original y acepto las consecuencias de cualquier violación a esta declaración.

Fecha: 17 de junio 2026

Firma: _Sergio Cuervo-Arango Sala

Anexo B. Código fuente del análisis

Este anexo reproduce el código que sustenta el análisis empírico, organizado según el orden de trabajo en los siete cuadernos del proyecto. Cada apartado combina el cuaderno que orquesta el análisis con el módulo de la librería (carpeta src) donde reside su lógica; las funciones aparecen en el punto en que se invocan por primera vez. Para no recargar el documento se ha omitido la documentación interna (docstrings y comentarios), que puede consultarse junto con los cuadernos ejecutados y sus resultados en el repositorio: **https://github.com/scuervo26/noshow-causal-pricing**.

Configuración de rutas del proyecto (src/rutas.py)

```
from __future__ import annotations
from pathlib import Path
PROJECT_ROOT: Path = Path(__file__).resolve().parent.parent
DATOS_BRUTOS = PROJECT_ROOT / "datos" / "brutos"
DATOS_PROCESADOS = PROJECT_ROOT / "datos" / "procesados"
CSV_RAW = DATOS_BRUTOS / "healthcare_noshows.csv"
CSV_FULL_CLEAN = DATOS_PROCESADOS / "full_clean_v1.csv"
CSV_TRAIN = DATOS_PROCESADOS / "train_v1.csv"
CSV_TEST = DATOS_PROCESADOS / "test_v1.csv"
OUTPUTS = PROJECT_ROOT / "outputs"
FIGURAS = OUTPUTS / "figuras"
MODELOS = OUTPUTS / "modelos"
REPORTES = OUTPUTS / "reportes"
BOOTSTRAP = OUTPUTS / "bootstrap"
SEED = 42
```

B.1 Limpieza de datos y creación de variables

src/preparacion.py · notebooks/01_limpieza_y_variables.ipynb

```
from __future__ import annotations
import json
import logging
from datetime import datetime, timezone
from pathlib import Path
from typing import Any, Mapping
import numpy as np
import pandas as pd
logger = logging.getLogger(__name__)
AGE_BAND_BINS = [-0.5, 18, 35, 55, 70, 200]
AGE_BAND_LABELS = ["0-18", "19-35", "36-55", "56-70", "70+"]
LEAD_TIME_BIN_EDGES = [-np.inf, 0.5, 7.5, 14.5, np.inf]
LEAD_TIME_BIN_LABELS = ["mismo_dia", "1-7d", "8-14d", "15+d"]
COLS_COMORBILIDAD = ["Hipertension", "Diabetes", "Alcoholism", "Handcap"]
```

```
from __future__ import annotations
import logging
import sys
from pathlib import Path
import numpy as np
import pandas as pd
PROJECT_ROOT = Path.cwd().parent if Path.cwd().name == "notebooks" else Path.cwd()
if str(PROJECT_ROOT) not in sys.path:
    sys.path.insert(0, str(PROJECT_ROOT))
logging.basicConfig(
    level=logging.INFO,
    format="%(message)s",
    stream=sys.stdout,
    force=True,
)
np.random.seed(SEED)
DATOS_PROCESADOS.mkdir(parents=True, exist_ok=True)
REPORTES.mkdir(parents=True, exist_ok=True)
print(f"Semilla fijada en {SEED}. Ruta de salida: {DATOS_PROCESADOS}")
```

```
def cargar_datos_brutos(ruta_csv: str | Path) -> pd.DataFrame:
    df = pd.read_csv(ruta_csv)
    logger.info("Cargado dataset bruto: %d filas, %d columnas", *df.shape)
    return df
```

```
df = prep.cargar_datos_brutos(CSV_RAW)
df_bruto = df.copy()  # snapshot pre-limpieza para los metadatos defensivos
print(f"Forma: {df.shape}")
print()
print("Dtypes por columna:")
print(df.dtypes)
print()
print("Nulos por columna:")
print(df.isna().sum())
print()
print(f"Duplicados completos: {df.duplicated().sum()}")
print(f"AppointmentID duplicados: {df['AppointmentID'].duplicated().sum()}")
```

```
def verificar_codificacion_showed_up(df: pd.DataFrame) -> pd.DataFrame:
    serie = df["Showed_up"]
    if serie.dtype == bool:
        attended = int(serie.sum())
        no_shows = int((~serie).sum())
    elif pd.api.types.is_integer_dtype(serie):
        attended = int((serie == 1).sum())
        no_shows = int((serie == 0).sum())
    else:
        raise ValueError(
            f"Tipo no soportado para Showed_up: {serie.dtype}. "
            "Esperado bool o entero."
        )
    total = attended + no_shows
    if total != len(df):
        raise ValueError(
            f"Showed_up tiene valores inesperados: total={total}, n={len(df)}"
        )
    pct_attended = attended / total
    logger.info(
        "Showed_up: %d asistieron (%.1f%%), %d no-shows (%.1f%%)",
        attended, pct_attended * 100, no_shows, (1 - pct_attended) * 100,
    )
    if not 0.65 < pct_attended < 0.90:
        raise ValueError(
            f"Tasa de asistencia inesperada ({pct_attended:.1%}). "
            "Verificar codificación de Showed_up antes de continuar."
        )
    df = df.copy()
    df["Showed_up"] = df["Showed_up"].astype(int)
    return df

df = prep.verificar_codificacion_showed_up(df)
print()
print(f"Tipo final de Showed_up: {df['Showed_up'].dtype}")
print(f"Distribución: {df['Showed_up'].value_counts().to_dict()}")
```

```
def limpiar_valores_imposibles(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()
    n_inicial = len(df)
    parte_fraccionaria = df["PatientId"] - df["PatientId"].astype("int64")
    mask_frac = parte_fraccionaria.abs() > 1e-9
    n_patient_id_frac = int(mask_frac.sum())
    if n_patient_id_frac:
        ids_enteros = set(df.loc[~mask_frac, "PatientId"].astype("int64").tolist())
        ids_truncados = df.loc[mask_frac, "PatientId"].astype("int64").tolist()
        colisiones = [pid for pid in ids_truncados if pid in ids_enteros]
        if colisiones:
            raise ValueError(
                f"Truncar PatientId fraccionarios produciría {len(colisiones)} "
                f"colisión(es) con IDs enteros existentes — revisar el CSV. "
                f"Primeros ejemplos: {colisiones[:5]}"
            )
        logger.info(
            "PatientId con parte fraccionaria (truncados a int64, sin colisiones): %d filas",
            n_patient_id_frac,
        )
    df["PatientId"] = df["PatientId"].astype("int64")
    df.attrs["n_patient_id_fraccionarios"] = n_patient_id_frac
    n_edad_negativa = int((df["Age"] < 0).sum())
    if n_edad_negativa:
        df = df.loc[df["Age"] >= 0].reset_index(drop=True)
    logger.info("Eliminadas %d filas con Age < 0", n_edad_negativa)
    anomalia_orden = pd.to_datetime(df["AppointmentDay"]) < pd.to_datetime(df["ScheduledDay"])
    n_anomalias = int(anomalia_orden.sum())
```

```
    logger.info(
        "AppointmentDay < ScheduledDay (anomalía documentada, no eliminada): %d filas (%.2f%%)",
        n_anomalias, 100 * n_anomalias / max(len(df), 1),
    )
    valores_handcap = sorted(pd.unique(df["Handcap"]).tolist())
    logger.info(
        "Valores únicos de Handcap: %s (binario, no requiere recodificación)",
        valores_handcap,
    )
    logger.info(
        "Limpieza: %d → %d filas (%d eliminadas)",
        n_inicial, len(df), n_inicial - len(df),
    )
    return df

df = prep.limpiar_valores_imposibles(df)

def parsear_fechas(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()
    df["ScheduledDay"] = pd.to_datetime(df["ScheduledDay"])
    df["AppointmentDay"] = pd.to_datetime(df["AppointmentDay"])
    diff_calculado = (df["AppointmentDay"] - df["ScheduledDay"]).dt.days
    if "Date.diff" in df.columns:
        n_inconsistencias = int((diff_calculado != df["Date.diff"]).sum())
        logger.info(
            "Date.diff: %d inconsistencias entre la columna existente y el recálculo "
            "(se sobrescribe con el recálculo, autoritativo)",
            n_inconsistencias,
        )
    df["Date.diff"] = diff_calculado
    return df

df = prep.parsear_fechas(df)
print()
print("Rango de ScheduledDay: ", df['ScheduledDay'].min().date(), '→',
df['ScheduledDay'].max().date())
print("Rango de AppointmentDay:", df['AppointmentDay'].min().date(), '→',
df['AppointmentDay'].max().date())
print()
print("Lead time (días) — describe:")
print(df['Date.diff'].describe())

def crear_features_basicas(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()
    df["lead_time"] = df["Date.diff"]
    df["lead_time_bin"] = pd.cut(
        df["lead_time"],
        bins=LEAD_TIME_BIN_EDGES,
        labels=LEAD_TIME_BIN_LABELS,
        include_lowest=True,
    ).astype(str)
    df["age_band"] = pd.cut(
        df["Age"],
        bins=AGE_BAND_BINS,
        labels=AGE_BAND_LABELS,
        include_lowest=True,
    ).astype(str)
    df["comorbidity_count"] = df[COLS_COMORBILIDAD].astype(int).sum(axis=1)
    df["chronic_flag"] = (df["comorbidity_count"] > 0).astype(int)
    df["scheduled_weekday"] = df["ScheduledDay"].dt.dayofweek
    df["scheduled_hour"] = df["ScheduledDay"].dt.hour
    df["scheduled_month"] = df["ScheduledDay"].dt.month
    df["appointment_weekday"] = df["AppointmentDay"].dt.dayofweek
    df["appointment_month"] = df["AppointmentDay"].dt.month
    if df["scheduled_hour"].nunique() <= 1:
        valor_constante = df["scheduled_hour"].iloc[0]
        logger.info(
            "scheduled_hour es constante (todo %s) — eliminada de las salidas",
            valor_constante,
        )
        df = df.drop(columns=["scheduled_hour"])
    return df

df = prep.crear_features_basicas(df)
print()
print("age_band:")
print(df['age_band'].value_counts().sort_index())
```

```
print()
print("lead_time_bin:")
print(df['lead_time_bin'].value_counts())
print()
print("comorbidity_count:")
print(df['comorbidity_count'].value_counts().sort_index())

def crear_features_historial_paciente(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy().reset_index(drop=True)
    n = len(df)
    counts = np.zeros(n, dtype=int)
    rates = np.full(n, np.nan, dtype=float)
    indices_por_paciente = df.groupby("PatientId").indices
    appt_all = df["AppointmentDay"].values
    sched_all = df["ScheduledDay"].values
    showed_all = df["Showed_up"].values.astype(int)
    for idx_grupo in indices_por_paciente.values():
        appt_p = appt_all[idx_grupo]
        sched_p = sched_all[idx_grupo]
        showed_p = showed_all[idx_grupo]
        orden = np.argsort(appt_p)
        appt_ordenados = appt_p[orden]
        showed_ordenados = showed_p[orden]
        cum_noshow = np.cumsum(1 - showed_ordenados)
        for i, idx in enumerate(idx_grupo):
            cutoff = sched_p[i]
            own_appt = appt_p[i]
            own_outcome = showed_p[i]
            k = int(np.searchsorted(appt_ordenados, cutoff, side="left"))
            if own_appt < cutoff:
                n_prior = k - 1
                if n_prior > 0:
                    no_show_sum = cum_noshow[k - 1] - (1 - own_outcome)
                    rates[idx] = no_show_sum / n_prior
            else:
                n_prior = k
                if n_prior > 0:
                    rates[idx] = cum_noshow[k - 1] / n_prior
            counts[idx] = n_prior
    df["prior_appointment_count"] = counts
    df["prior_noshow_rate"] = rates
    df["is_first_visit"] = (counts == 0).astype(int)
    n_first = int(df["is_first_visit"].sum())
    logger.info(
        "Historial paciente: %d primeras visitas (%.1f%%); prior_noshow_rate es NaN para esas
filas (correcto, no son cero)",
        n_first, 100 * n_first / len(df),
    )
    return df

df = prep.crear_features_historial_paciente(df)
print()
print("prior_appointment_count — describe:")
print(df['prior_appointment_count'].describe())
print()
print(f"prior_noshow_rate — describe (excluyendo {df['prior_noshow_rate'].isna().sum()} NaN de
primeras visitas):")
print(df['prior_noshow_rate'].dropna().describe())
print()
print("is_first_visit:", df['is_first_visit'].value_counts().to_dict())

def dividir_temporal(
    df: pd.DataFrame, percentil: float = 0.80
) -> tuple[pd.DataFrame, pd.DataFrame, pd.Timestamp]:
    if not 0 < percentil < 1:
        raise ValueError(f"percentil debe estar en (0, 1); recibido: {percentil}")
    fecha_corte = df["AppointmentDay"].quantile(percentil, interpolation="lower")
    train = df.loc[df["AppointmentDay"] <= fecha_corte].reset_index(drop=True)
    test = df.loc[df["AppointmentDay"] > fecha_corte].reset_index(drop=True)
    logger.info(
        "División temporal en %s (P%d): train=%d (%.1f%%), test=%d (%.1f%%)",
        pd.Timestamp(fecha_corte).date(), int(percentil * 100),
        len(train), 100 * len(train) / len(df),
        len(test), 100 * len(test) / len(df),
    )
    return train, test, pd.Timestamp(fecha_corte)
```

```
train, test, fecha_corte = prep.dividir_temporal(df, percentil=0.80)
print()
print(f"Fecha de corte: {fecha_corte.date()} (incluida en train)")
print(f"Train:    {len(train):>7,}    filas    —    {train['AppointmentDay'].min().date()}    →
{train['AppointmentDay'].max().date()}")
print(f"Test:        {len(test):>7,}    filas    —    {test['AppointmentDay'].min().date()}    →
{test['AppointmentDay'].max().date()}")


def ajustar_codificacion_frecuencia_barrio(train_df: pd.DataFrame) -> dict[str, float]:
    counts = train_df["Neighbourhood"].value_counts(normalize=True)
    logger.info(
        "Codificación por frecuencia ajustada en train: %d barrios únicos",
        len(counts),
    )
    return counts.to_dict()


def aplicar_codificacion_frecuencia_barrio(
    df: pd.DataFrame, mapping: Mapping[str, float]
) -> pd.DataFrame:
    df = df.copy()
    encoded = df["Neighbourhood"].map(mapping)
    n_oov = int(encoded.isna().sum())
    df["neighbourhood_encoded"] = encoded.fillna(0.0)
    if n_oov:
        logger.info(
            "%d filas con Neighbourhood fuera de train (codificadas como 0.0)",
            n_oov,
        )
    return df


mapping_barrio = prep.ajustar_codificacion_frecuencia_barrio(train)
df = prep.aplicar_codificacion_frecuencia_barrio(df, mapping_barrio)
train = prep.aplicar_codificacion_frecuencia_barrio(train, mapping_barrio)
test = prep.aplicar_codificacion_frecuencia_barrio(test, mapping_barrio)
print()
print("neighbourhood_encoded — describe en full_clean:")
print(df['neighbourhood_encoded'].describe())


def escribir_metadatos_nb01(
    ruta_json: str | Path,
    *,
    seed: int,
    df_bruto: pd.DataFrame,
    df_full: pd.DataFrame,
    df_train: pd.DataFrame,
    df_test: pd.DataFrame,
    fecha_corte: pd.Timestamp,
    mapping_barrio: Mapping[str, float],
) -> dict[str, Any]:
    barrios_train = set(mapping_barrio.keys())
    barrios_test = set(df_test["Neighbourhood"].unique())
    n_oov_test = len(barrios_test - barrios_train)
    n_edad_negativa_eliminadas = int((df_bruto["Age"] < 0).sum())
    valores_unicos_handcap = sorted(
        [bool(v) if isinstance(v, (bool, np.bool_)) else v
         for v in pd.unique(df_bruto["Handcap"])]
    )
    scheduled_hour_eliminada = "scheduled_hour" not in df_full.columns
    metadatos = {
        "generado_en": datetime.now(timezone.utc).isoformat(),
        "seed": seed,
        "n_filas_brutas": int(len(df_bruto)),
        "n_filas_total": int(len(df_full)),
        "n_filas_train": int(len(df_train)),
        "n_filas_test": int(len(df_test)),
        "fecha_corte_appointmentday": fecha_corte.date().isoformat(),
        "pct_atendieron": float((df_full["Showed_up"] == 1).mean()),
        "n_primeras_visitas": int(df_full["is_first_visit"].sum()),
        "pct_primeras_visitas": float(df_full["is_first_visit"].mean()),
        "n_barrios_train": len(barrios_train),
        "n_barrios_test_oov": n_oov_test,
        "n_anomalia_appt_lt_sched": int(
            (df_full["AppointmentDay"] < df_full["ScheduledDay"]).sum()
        ),
        "media_prior_noshow_rate": float(
            df_full["prior_noshow_rate"].dropna().mean()
        ),
```

```
        "n_edad_negativa_eliminadas": n_edad_negativa_eliminadas,
        "valores_unicos_handcap": valores_unicos_handcap,
        "scheduled_hour_eliminada_por_constante": scheduled_hour_eliminada,
        "n_patient_id_fraccionarios": int(
            df_full.attrs.get("n_patient_id_fraccionarios", 0)
        ),
    }
    ruta_json = Path(ruta_json)
    ruta_json.parent.mkdir(parents=True, exist_ok=True)
    with ruta_json.open("w", encoding="utf-8") as f:
        json.dump(metadatos, f, indent=2, ensure_ascii=False)
    logger.info("Metadatos NB01 guardados en %s", ruta_json)
    return metadatos

df.to_csv(CSV_FULL_CLEAN, index=False)
train.to_csv(CSV_TRAIN, index=False)
test.to_csv(CSV_TEST, index=False)
print("Ficheros guardados en datos/procesados/:")
for ruta in (CSV_FULL_CLEAN, CSV_TRAIN, CSV_TEST):
    size_mb = ruta.stat().st_size / (1024 * 1024)
    print(f"  {ruta.name:<22} {size_mb:6.2f} MB")
metadatos = prep.escribir_metadatos_nb01(
    REPORTES / "nb01_metadatos_v1.json",
    seed=SEED,
    df_bruto=df_bruto,
    df_full=df,
    df_train=train,
    df_test=test,
    fecha_corte=fecha_corte,
    mapping_barrio=mapping_barrio,
)
print()
print("Metadatos NB01:")
for k, v in metadatos.items():
    print(f"  {k}: {v}")

COLUMNAS_ESPERADAS = {
    "PatientId", "AppointmentID", "Gender", "ScheduledDay", "AppointmentDay",
    "Age", "Neighbourhood", "Scholarship", "Hipertension", "Diabetes",
    "Alcoholism", "Handcap", "SMS_received", "Showed_up", "Date.diff",
    "lead_time", "lead_time_bin", "age_band",
    "comorbidity_count", "chronic_flag",
    "scheduled_weekday", "scheduled_month",
    "appointment_weekday", "appointment_month",
    "prior_appointment_count", "prior_noshow_rate", "is_first_visit",
    "neighbourhood_encoded",
}
faltantes = COLUMNAS_ESPERADAS - set(df.columns)
extras = set(df.columns) - COLUMNAS_ESPERADAS
assert not faltantes, f"Faltan columnas esperadas: {faltantes}"
assert df['Showed_up'].dtype == 'int64', f"Showed_up debe ser int64, es {df['Showed_up'].dtype}"
assert df['Showed_up'].isin([0, 1]).all(), "Showed_up tiene valores fuera de {0, 1}"
assert df['PatientId'].dtype == 'int64', f"PatientId debe ser int64, es {df['PatientId'].dtype}"
assert df['neighbourhood_encoded'].notna().all(), "neighbourhood_encoded tiene NaN"
assert (df['prior_appointment_count'] >= 0).all(), "prior_appointment_count negativo"
mask_first = df['is_first_visit'] == 1
assert df.loc[mask_first, 'prior_noshow_rate'].isna().all(), "Hay primeras visitas con prior_noshow_rate no-NaN — fuga sospechosa"
assert df.loc[~mask_first, 'prior_noshow_rate'].notna().all(), "Hay no-primeras visitas con prior_noshow_rate NaN — inconsistencia"
assert len(df) == len(train) + len(test), f"Pérdida en split: full={len(df)}, train+test={len(train)+len(test)}"
assert train['AppointmentDay'].max() <= test['AppointmentDay'].min(), "Solapamiento temporal entre train y test"
n_barrios_train = len(set(train['Neighbourhood']))
n_barrios_test_oov = len(set(test['Neighbourhood']) - set(mapping_barrio.keys()))
assert n_barrios_train == len(mapping_barrio), "El mapping de barrios no coincide con los barrios únicos de train"
print("Todas las comprobaciones finales pasaron.")
print(f"  Columnas esperadas: {len(COLUMNAS_ESPERADAS)}, presentes en full_clean: {len(set(df.columns) & COLUMNAS_ESPERADAS)}")
print(f"  Columnas extra (informativas): {sorted(extras) if extras else 'ninguna'}")
print(f"  Forma final de full_clean: {df.shape}")
print(f"  Barrios únicos en train: {n_barrios_train}; en test fuera de train: {n_barrios_test_oov}")
```

B.2 Análisis exploratorio de datos

src/eda.py · notebooks/02_exploracion_datos.ipynb

```
from __future__ import annotations
import json
import logging
from datetime import datetime, timezone
from pathlib import Path
from typing import Any, Iterable, Mapping
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
logger = logging.getLogger(__name__)
COLS_CORRELACION_NUMERICAS = [
    "Age", "lead_time", "prior_appointment_count", "prior_noshow_rate",
    "neighbourhood_encoded", "scheduled_weekday", "scheduled_month",
    "appointment_weekday", "appointment_month", "comorbidity_count",
    "Scholarship", "Hipertension", "Diabetes", "Alcoholism", "Handcap",
    "chronic_flag", "is_first_visit",
    "SMS_received", "Showed_up",
]
ETIQUETAS_VARIABLES: dict[str, str] = {
    "Age": "edad",
    "lead_time": "días de antelación",
    "lead_time_bin": "antelación (bin)",
    "age_band": "banda de edad",
    "appointment_weekday": "día semana cita",
    "scheduled_weekday": "día semana programación",
    "scheduled_month": "mes programación",
    "appointment_month": "mes cita",
    "comorbidity_count": "nº comorbilidades",
    "chronic_flag": "cualquier comorbilidad",
    "Scholarship": "beca social (Bolsa Família)",
    "Hipertension": "hipertensión",
    "Diabetes": "diabetes",
    "Alcoholism": "alcoholismo",
    "Handcap": "discapacidad",
    "SMS_received": "recibió SMS",
    "Showed_up": "asistió",
    "prior_appointment_count": "nº citas previas",
    "prior_noshow_rate": "tasa no-show previa",
    "neighbourhood_encoded": "barrio (frecuencia)",
    "is_first_visit": "primera visita",
    "Gender": "género",
}
NOMBRES_DIA_SEMANA = ["lun", "mar", "mié", "jue", "vie", "sáb", "dom"]
COLS_BINARIAS = [
    "SMS_received", "Scholarship",
    "Hipertension", "Diabetes", "Alcoholism", "Handcap",
    "is_first_visit", "chronic_flag",
]

def configurar_estilo() -> None:
    sns.set_theme(
        style="whitegrid",
        context="notebook",
        palette="colorblind",
        font_scale=1.0,
    )
    plt.rcParams.update({
        "figure.dpi": 110,
        "savefig.dpi": 200,
        "savefig.bbox": "tight",
        "axes.titleweight": "semibold",
        "axes.titlesize": 12,
        "axes.labelsize": 11,
    })

def normalizar_binarios(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()
    convertidas = []
    for c in COLS_BINARIAS:
        if c in df.columns and df[c].dtype == bool:
            df[c] = df[c].astype(int)
            convertidas.append(c)
    if convertidas:
        logger.info("Columnas binarias normalizadas a int: %s", convertidas)
    return df
```

```
from __future__ import annotations
import logging
import sys
from pathlib import Path
import numpy as np
import pandas as pd
PROJECT_ROOT = Path.cwd().parent if Path.cwd().name == "notebooks" else Path.cwd()
if str(PROJECT_ROOT) not in sys.path:
    sys.path.insert(0, str(PROJECT_ROOT))
logging.basicConfig(
    level=logging.INFO,
    format="%(message)s",
    stream=sys.stdout,
    force=True,
)
np.random.seed(SEED)
FIGURAS.mkdir(parents=True, exist_ok=True)
REPORTES.mkdir(parents=True, exist_ok=True)
eda.configurar_estilo()
df = pd.read_csv(
    CSV_FULL_CLEAN,
    parse_dates=["ScheduledDay", "AppointmentDay"],
)
df = eda.normalizar_binarios(df)
print(f"Dataset cargado: {df.shape[0]:,} filas × {df.shape[1]} columnas".replace(",", "."))
print(f"Columnas disponibles: {sorted(df.columns)}")

def tasa_noshow_global(df: pd.DataFrame) -> dict[str, float]:
    n = len(df)
    n_asistio = int((df["Showed_up"] == 1).sum())
    n_noshow = n - n_asistio
    pct_asistio = n_asistio / n
    pct_noshow = 1 - pct_asistio
    logger.info(
        "Tasa global: %d filas → %d asistieron (%.2f%%), %d no-show (%.2f%%)",
        n, n_asistio, 100 * pct_asistio, n_noshow, 100 * pct_noshow,
    )
    return {
        "n_total": n,
        "n_asistio": n_asistio,
        "n_noshow": n_noshow,
        "pct_asistio": pct_asistio,
        "pct_noshow": pct_noshow,
    }

def tasa_noshow_por_variable(
    df: pd.DataFrame,
    columna: str,
    *,
    orden: Iterable | None = None,
    min_n: int = 0,
) -> pd.DataFrame:
    grupo = df.groupby(columna, dropna=False, observed=True)
    tabla = pd.DataFrame({
        "n": grupo.size(),
        "pct_asistio": grupo["Showed_up"].mean(),
    })
    tabla["pct_noshow"] = 1 - tabla["pct_asistio"]
    tabla = tabla.loc[tabla["n"] >= min_n]
    if orden is not None:
        tabla = tabla.reindex([v for v in orden if v in tabla.index])
    return tabla

def guardar_figura(fig: plt.Figure, ruta: str | Path) -> Path:
    ruta = Path(ruta)
    ruta.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(ruta)
    logger.info("Figura guardada: %s", ruta)
    return ruta

def plot_tasa_noshow_panel(
    df: pd.DataFrame,
    ruta_salida: str | Path,
    *,
    variables: list[str] | None = None,
) -> Path:
```

```
    if variables is None:
        variables = [
            "lead_time_bin", "age_band", "appointment_weekday",
            "SMS_received", "comorbidity_count", "Scholarship",
        ]
    ordenes: dict[str, list] = {
        "lead_time_bin": ["mismo_dia", "1-7d", "8-14d", "15+d"],
        "age_band": ["0-18", "19-35", "36-55", "56-70", "70+"],
    }
    tasa_global = (1 - df["Showed_up"].mean())
    n_var = len(variables)
    n_cols = 3
    n_rows = (n_var + n_cols - 1) // n_cols
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(13, 4 * n_rows))
    axes = np.array(axes).reshape(-1)
    for ax, var in zip(axes, variables):
        tabla = tasa_noshow_por_variable(df, var, orden=ordenes.get(var))
        x = [str(v) for v in tabla.index]
        if var == "appointment_weekday":
            x = [NOMBRES_DIA_SEMANA[int(v)] for v in tabla.index]
        bars = ax.bar(x, 100 * tabla["pct_noshow"], color=sns.color_palette()[0])
        ax.axhline(100 * tasa_global, color="grey", linestyle="--", linewidth=1,
                   label=f"global ({100 * tasa_global:.1f}%)")
        ax.set_title(f"No-show por {ETIQUETAS_VARIABLES.get(var, var)}")
        ax.set_ylabel("tasa de no-show (%)")
        for rect, n in zip(bars, tabla["n"]):
            ax.text(rect.get_x() + rect.get_width() / 2,
                    rect.get_height() + 0.4,
                    f"n={n:,}".replace(",", "."),
                    ha="center", va="bottom", fontsize=8, color="dimgray")
        ax.legend(loc="upper right", fontsize=8)
        ax.tick_params(axis="x", rotation=0)
    for ax in axes[len(variables):]:
        ax.set_visible(False)
    fig.suptitle("Tasa de no-asistencia por variable (descriptivo, no causal)",
                 fontsize=13, y=1.02)
    fig.tight_layout()
    return guardar_figura(fig, ruta_salida)

def plot_noshow_top_barrios(
    df: pd.DataFrame, ruta_salida: str | Path, n_top: int = 20,
) -> Path:
    tabla = tasa_noshow_por_variable(df, "Neighbourhood")
    tabla = tabla.sort_values("n", ascending=False).head(n_top)
    tabla = tabla.sort_values("pct_noshow")  # para que las barras suban
    fig, ax = plt.subplots(figsize=(9, 0.35 * len(tabla) + 1.5))
    ax.barh(tabla.index, 100 * tabla["pct_noshow"], color=sns.color_palette()[0])
    ax.axvline(100 * (1 - df["Showed_up"].mean()), color="grey",
               linestyle="--", linewidth=1, label="global")
    ax.set_xlabel("tasa de no-show (%)")
    ax.set_title(f"Tasa de no-asistencia — top {n_top} barrios por volumen")
    for i, (_, fila) in enumerate(tabla.iterrows()):
        ax.text(100 * fila["pct_noshow"] + 0.2, i,
                f"n={int(fila['n']):,}".replace(",", "."),
                va="center", fontsize=8, color="dimgray")
    ax.legend(loc="lower right", fontsize=8)
    fig.tight_layout()
    return guardar_figura(fig, ruta_salida)

info_global = eda.tasa_noshow_global(df)
for var in ["lead_time_bin", "age_band", "appointment_weekday",
            "SMS_received", "comorbidity_count", "Scholarship"]:
    print(f"\n--- Tasa de no-show por {var} ---")
    tabla = eda.tasa_noshow_por_variable(
        df, var,
        orden={
            "lead_time_bin": ["mismo_dia", "1-7d", "8-14d", "15+d"],
            "age_band": ["0-18", "19-35", "36-55", "56-70", "70+"],
        }.get(var),
    )
    print(tabla.round(4).to_string())
ruta_panel = eda.plot_tasa_noshow_panel(df, FIGURAS / "nb02_noshow_por_variables_v1.png")
ruta_barrios = eda.plot_noshow_top_barrios(
    df, FIGURAS / "nb02_noshow_top20_barrios_v1.png", n_top=20,
)

def matriz_correlacion(
    df: pd.DataFrame, columnas: list[str] | None = None,
```

```
) -> pd.DataFrame:
    columnas = columnas or COLS_CORRELACION_NUMERICAS
    cols_validas = [c for c in columnas if c in df.columns]
    faltantes = set(columnas) - set(cols_validas)
    if faltantes:
        logger.warning("Columnas ausentes en la matriz de correlación: %s", faltantes)
    sub = df[cols_validas].copy()
    for c in cols_validas:
        if sub[c].dtype == bool:
            sub[c] = sub[c].astype(int)
    return sub.corr(method="pearson")


def plot_correlacion_heatmap(
    corr: pd.DataFrame, ruta_salida: str | Path,
) -> Path:
    fig, ax = plt.subplots(figsize=(11, 9))
    mask = np.triu(np.ones_like(corr, dtype=bool), k=1)
    sns.heatmap(
        corr, mask=mask, ax=ax,
        cmap="RdBu_r", vmin=-1, vmax=1, center=0,
        annot=True, fmt=".2f", annot_kws={"size": 7},
        cbar_kws={"label": "Pearson r"},
        linewidths=0.5, linecolor="white",
    )
    ax.set_title("Matriz de correlación — variables candidatas para IPW y XGBoost")
    fig.tight_layout()
    return guardar_figura(fig, ruta_salida)


def plot_correlaciones_dual(
    corr: pd.DataFrame, ruta_salida: str | Path, *, top_n: int = 15,
) -> Path:
    if "Showed_up" not in corr.index or "SMS_received" not in corr.index:
        raise ValueError(
            "La matriz de correlación debe incluir Showed_up y SMS_received."
        )
    excluidas = {"Showed_up", "SMS_received"}
    candidatos = [c for c in corr.index if c not in excluidas]
    serie_outcome = corr.loc[candidatos, "Showed_up"].sort_values()
    serie_trat = corr.loc[candidatos, "SMS_received"].sort_values()
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(13, max(4, 0.32 * len(candidatos))))
    colores_a = ["tab:red" if v < 0 else "tab:blue" for v in serie_outcome.values]
    ax1.barh(serie_outcome.index, serie_outcome.values, color=colores_a)
    ax1.axvline(0, color="black", linewidth=0.7)
    ax1.set_title("Correlación con Showed_up (asistió=1)")
    ax1.set_xlabel("Pearson r")
    colores_b = ["tab:red" if v < 0 else "tab:blue" for v in serie_trat.values]
    ax2.barh(serie_trat.index, serie_trat.values, color=colores_b)
    ax2.axvline(0, color="black", linewidth=0.7)
    ax2.set_title("Correlación con SMS_received (tratamiento)")
    ax2.set_xlabel("Pearson r")
    fig.suptitle("Selección de confounders: variables asociadas a outcome y tratamiento",
                 fontsize=12, y=1.02)
    fig.tight_layout()
    return guardar_figura(fig, ruta_salida)


corr = eda.matriz_correlacion(df)
corr_outcome    =    corr["Showed_up"].drop("Showed_up").sort_values(key=lambda    s:    s.abs(),
ascending=False)
corr_tratamiento = corr["SMS_received"].drop("SMS_received").sort_values(key=lambda s: s.abs(),
ascending=False)
print("Top 8 correlaciones con Showed_up (por magnitud):")
print(corr_outcome.head(8).round(4).to_string())
print("\nTop 8 correlaciones con SMS_received (por magnitud):")
print(corr_tratamiento.head(8).round(4).to_string())
ruta_heatmap = eda.plot_correlacion_heatmap(corr, FIGURAS / "nb02_correlacion_heatmap_v1.png")
ruta_dual = eda.plot_correlaciones_dual(corr, FIGURAS / "nb02_correlaciones_dual_v1.png")


def tasa_sms_por_variable(df: pd.DataFrame, columna: str,
                          *, orden: Iterable | None = None) -> pd.DataFrame:
    grupo = df.groupby(columna, dropna=False, observed=True)
    tabla = pd.DataFrame({
        "n": grupo.size(),
        "pct_sms": grupo["SMS_received"].mean(),
    })
    if orden is not None:
        tabla = tabla.reindex([v for v in orden if v in tabla.index])
    return tabla
```

```
def plot_distribucion_sms(df: pd.DataFrame, ruta_salida: str | Path) -> Path:
    ordenes = {
        "lead_time_bin": ["mismo_dia", "1-7d", "8-14d", "15+d"],
        "age_band": ["0-18", "19-35", "36-55", "56-70", "70+"],
    }
    variables = ["age_band", "lead_time_bin", "Scholarship"]
    tasa_global_sms = df["SMS_received"].mean()
    fig, axes = plt.subplots(2, 2, figsize=(13, 9))
    axes = axes.flatten()
    for ax, var in zip(axes[:3], variables):
        tabla = tasa_sms_por_variable(df, var, orden=ordenes.get(var))
        x = [str(v) for v in tabla.index]
        bars = ax.bar(x, 100 * tabla["pct_sms"], color=sns.color_palette()[1])
        ax.axhline(100 * tasa_global_sms, color="grey", linestyle="--",
                   linewidth=1, label=f"global ({100 * tasa_global_sms:.1f}%)")
        ax.set_title(f"% recepción SMS por {ETIQUETAS_VARIABLES.get(var, var)}")
        ax.set_ylabel("% SMS = 1")
        for rect, n in zip(bars, tabla["n"]):
            ax.text(rect.get_x() + rect.get_width() / 2,
                    rect.get_height() + 0.4,
                    f"n={n:,}".replace(",", "."),
                    ha="center", va="bottom", fontsize=8, color="dimgray")
        ax.legend(loc="upper right", fontsize=8)
    ax = axes[3]
    tabla = tasa_sms_por_variable(df, "Neighbourhood")
    tabla = tabla.sort_values("n", ascending=False).head(10).sort_values("pct_sms")
    ax.barh(tabla.index, 100 * tabla["pct_sms"], color=sns.color_palette()[1])
    ax.axvline(100 * tasa_global_sms, color="grey", linestyle="--",
               linewidth=1, label="global")
    ax.set_xlabel("% SMS = 1")
    ax.set_title("% recepción SMS — top 10 barrios por volumen")
    ax.legend(loc="lower right", fontsize=8)
    fig.suptitle("¿Quién recibe SMS? Distribución por subgrupo (motiva IPW)",
                 fontsize=13, y=1.00)
    fig.tight_layout()
    return guardar_figura(fig, ruta_salida)

print(f"% SMS global = {100 * df['SMS_received'].mean():.2f}%\n")
ordenes = {
    "lead_time_bin": ["mismo_dia", "1-7d", "8-14d", "15+d"],
    "age_band": ["0-18", "19-35", "36-55", "56-70", "70+"],
}
for var in ["age_band", "lead_time_bin", "Scholarship"]:
    print(f"--- % SMS por {var} ---")
    print(eda.tasa_sms_por_variable(df, var, orden=ordenes.get(var)).round(4).to_string())
    print()
ruta_sms = eda.plot_distribucion_sms(df, FIGURAS / "nb02_sms_asignacion_v1.png")

def inversion_signo_sms(df: pd.DataFrame) -> dict[str, float]:
    grupo = df.groupby("SMS_received", observed=True)
    serie_noshow = 1 - grupo["Showed_up"].mean()
    n_por_brazo = grupo.size()
    pct_noshow_sms0 = float(serie_noshow.loc[0])
    pct_noshow_sms1 = float(serie_noshow.loc[1])
    diff_pp = pct_noshow_sms1 - pct_noshow_sms0  # positivo → reversal observada
    pct_sms_global = float(df["SMS_received"].mean())
    logger.info(
        "Tasas no-show sin ajustar: SMS=0 → %.2f%%, SMS=1 → %.2f%% (Δ = %+.2f pp)",
        100 * pct_noshow_sms0, 100 * pct_noshow_sms1, 100 * diff_pp,
    )
    if diff_pp > 0:
        logger.info(
            "Inversión de signo CONFIRMADA: los receptores de SMS no asisten "
            "MÁS sin ajustar. Esto es sesgo de selección, no efecto causal — "
            "documentado en NB02 §4 como motivación cualitativa de IPW."
        )
    else:
        logger.warning(
            "Inversión NO observada en este split (Δ = %+.2f pp). Revisar "
            "la sección 4 del EDA antes de redactar la narrativa.",
            100 * diff_pp,
        )
    return {
        "n_sms0": int(n_por_brazo.loc[0]),
        "n_sms1": int(n_por_brazo.loc[1]),
        "pct_sms_global": pct_sms_global,
        "pct_noshow_sms0": pct_noshow_sms0,
```

```
        "pct_noshow_sms1": pct_noshow_sms1,
        "diff_pp_sms1_menos_sms0": diff_pp,
    }

def plot_inversion_signo_sms(df: pd.DataFrame, ruta_salida: str | Path) -> Path:
    info = inversion_signo_sms(df)
    fig, ax = plt.subplots(figsize=(7, 5))
    etiquetas = ["sin SMS (SMS=0)", "con SMS (SMS=1)"]
    valores = [100 * info["pct_noshow_sms0"], 100 * info["pct_noshow_sms1"]]
    ns = [info["n_sms0"], info["n_sms1"]]
    colores = [sns.color_palette()[0], sns.color_palette()[3]]
    bars = ax.bar(etiquetas, valores, color=colores)
    for rect, val, n in zip(bars, valores, ns):
        ax.text(rect.get_x() + rect.get_width() / 2,
                rect.get_height() + 0.3,
                f"{val:.2f}%\n(n={n:,})".replace(",", "."),
                ha="center", va="bottom", fontsize=10)
    diff_pp = 100 * info["diff_pp_sms1_menos_sms0"]
    ax.set_ylim(0, max(valores) + 5)
    ax.set_ylabel("tasa de no-show (%)")
    ax.set_title(
        "Comparación SIN AJUSTAR: tasa de no-show por brazo de SMS\n"
        f"Δ = {diff_pp:+.2f} pp — selección, no efecto causal"
    )
    ax.text(0.5, -0.18,
            "Los receptores de SMS son sistemáticamente diferentes "
            "(mayor antelación, mayor riesgo basal).\n"
            "La estimación causal se realiza en NB04 mediante IPW estabilizado.",
            transform=ax.transAxes, ha="center", va="top",
            fontsize=9, color="dimgray", style="italic")
    fig.tight_layout()
    return guardar_figura(fig, ruta_salida)

info_inversion = eda.inversion_signo_sms(df)
print(f"\nNº SMS=0: {info_inversion['n_sms0']:,}".replace(",", "."))
print(f"Nº SMS=1: {info_inversion['n_sms1']:,}".replace(",", "."))
print(f"% no-show SMS=0: {100 * info_inversion['pct_noshow_sms0']:.2f}%")
print(f"% no-show SMS=1: {100 * info_inversion['pct_noshow_sms1']:.2f}%")
print(f"Δ (SMS=1 − SMS=0): {100 * info_inversion['diff_pp_sms1_menos_sms0']:+.2f} pp")
ruta_inversion = eda.plot_inversion_signo_sms(
    df, FIGURAS / "nb02_inversion_signo_sms_v1.png",
)

def resumen_lead_time(df: pd.DataFrame) -> dict[str, float]:
    s = df["lead_time"]
    return {
        "n_negativo": int((s < 0).sum()),
        "n_mismo_dia": int((s == 0).sum()),
        "pct_mismo_dia": float((s == 0).mean()),
        "mediana": float(s.median()),
        "media": float(s.mean()),
        "p90": float(np.percentile(s, 90)),
        "p99": float(np.percentile(s, 99)),
        "maximo": int(s.max()),
    }

def plot_distribucion_lead_time(df: pd.DataFrame, ruta_salida: str | Path) -> Path:
    p99 = float(np.percentile(df["lead_time"], 99))
    sub = df.loc[(df["lead_time"] >= 0) & (df["lead_time"] <= p99)].copy()
    n_neg = int((df["lead_time"] < 0).sum())
    n_cola = int((df["lead_time"] > p99).sum())
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(13, 4.5))
    ax1.hist(sub["lead_time"], bins=range(0, int(p99) + 2), color=sns.color_palette()[0],
             edgecolor="white", linewidth=0.3)
    ax1.set_xlabel("lead_time (días)")
    ax1.set_ylabel("nº de citas")
    n_graf = f"{len(sub):,}".replace(",", ".")
    n_cola_str = f"{n_cola:,}".replace(",", ".")
    ax1.set_title(
        f"Distribución de lead_time (0 ≤ x ≤ P99 = {p99:.0f} días)\n"
        f"n graficado = {n_graf}; descartado: {n_neg} con lead_time<0, "
        f"{n_cola_str} en la cola >P99"
    )
    ax1.axvline(0, color="grey", linestyle="--", linewidth=0.8)
    por_dia = sub.groupby("lead_time").agg(
        n=("Showed_up", "size"), pct_noshow=("Showed_up", lambda s: 1 - s.mean()),
    )
```

```
    por_dia = por_dia.loc[por_dia["n"] >= 200]
    ax2.plot(por_dia.index, 100 * por_dia["pct_noshow"],
             marker="o", markersize=3, color=sns.color_palette()[3])
    ax2.axhline(100 * (1 - df["Showed_up"].mean()),
                color="grey", linestyle="--", linewidth=1, label="global")
    ax2.set_xlabel("lead_time (días)")
    ax2.set_ylabel("tasa de no-show (%)")
    ax2.set_title("Tasa de no-show por día de antelación (n ≥ 200)")
    ax2.legend(fontsize=8, loc="lower right")
    fig.tight_layout()
    return guardar_figura(fig, ruta_salida)

info_lead_time = eda.resumen_lead_time(df)
print("Resumen de lead_time (días):")
for k, v in info_lead_time.items():
    print(f"  {k:>20s}: {v}")
ruta_lt = eda.plot_distribucion_lead_time(
    df, FIGURAS / "nb02_lead_time_distribucion_v1.png",
)

def estadisticas_pacientes(df: pd.DataFrame) -> dict[str, Any]:
    citas_por_paciente = df.groupby("PatientId").size()
    n_pacientes = int(citas_por_paciente.size)
    noshow_primeras = float(
        1 - df.loc[df["is_first_visit"] == 1, "Showed_up"].mean()
    )
    noshow_repetidas = float(
        1 - df.loc[df["is_first_visit"] == 0, "Showed_up"].mean()
    )
    return {
        "n_pacientes_unicos": n_pacientes,
        "media_citas_por_paciente": float(citas_por_paciente.mean()),
        "mediana_citas_por_paciente": float(citas_por_paciente.median()),
        "p95_citas_por_paciente": float(np.percentile(citas_por_paciente, 95)),
        "max_citas_por_paciente": int(citas_por_paciente.max()),
        "pct_pacientes_una_sola_cita": float((citas_por_paciente == 1).mean()),
        "noshow_primeras_visitas": noshow_primeras,
        "noshow_repetidas": noshow_repetidas,
        "diff_pp_primera_menos_repetida": noshow_primeras - noshow_repetidas,
    }

def plot_analisis_pacientes(df: pd.DataFrame, ruta_salida: str | Path) -> Path:
    citas_por_paciente = df.groupby("PatientId").size()
    info = estadisticas_pacientes(df)
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(13, 4.5))
    p95 = int(np.percentile(citas_por_paciente, 95))
    bins = np.arange(0.5, p95 + 1.5, 1)
    ax1.hist(citas_por_paciente, bins=bins, color=sns.color_palette()[0],
             edgecolor="white", linewidth=0.4)
    ax1.set_xlabel("nº de citas por paciente")
    ax1.set_ylabel("nº de pacientes")
    ax1.set_title(
        f"Distribución de citas por paciente (eje recortado a P95={p95})\n"
        f"{info['n_pacientes_unicos']:,} pacientes únicos, "
        f"máx = {info['max_citas_por_paciente']}".replace(",", ".")
    )
    etiquetas = ["primera visita", "visita repetida"]
    valores = [100 * info["noshow_primeras_visitas"], 100 * info["noshow_repetidas"]]
    ns = [
        int((df["is_first_visit"] == 1).sum()),
        int((df["is_first_visit"] == 0).sum()),
    ]
    bars = ax2.bar(etiquetas, valores,
                   color=[sns.color_palette()[0], sns.color_palette()[3]])
    for rect, val, n in zip(bars, valores, ns):
        ax2.text(rect.get_x() + rect.get_width() / 2,
                 rect.get_height() + 0.3,
                 f"{val:.2f}%\n(n={n:,})".replace(",", "."),
                 ha="center", va="bottom", fontsize=10)
    diff_pp = 100 * info["diff_pp_primera_menos_repetida"]
    ax2.set_ylabel("tasa de no-show (%)")
    ax2.set_title(f"No-show: primera vs repetida (Δ = {diff_pp:+.2f} pp)")
    ax2.set_ylim(0, max(valores) + 5)
    fig.tight_layout()
    return guardar_figura(fig, ruta_salida)

info_pacientes = eda.estadisticas_pacientes(df)
```

```
print("Resumen a nivel paciente:")
for k, v in info_pacientes.items():
    if isinstance(v, float):
        print(f"  {k:>32s}: {v:.4f}")
    else:
        print(f"  {k:>32s}: {v:,}".replace(",", "."))
ruta_pac = eda.plot_analisis_pacientes(
    df, FIGURAS / "nb02_pacientes_distribucion_v1.png",
)

def escribir_metadatos_nb02(
    ruta_json: str | Path,
    *,
    df: pd.DataFrame,
    info_global: Mapping[str, Any],
    info_inversion: Mapping[str, Any],
    info_lead_time: Mapping[str, Any],
    info_pacientes: Mapping[str, Any],
    top_corr_outcome: Mapping[str, float],
    top_corr_tratamiento: Mapping[str, float],
) -> dict[str, Any]:
    metadatos: dict[str, Any] = {
        "generado_en": datetime.now(timezone.utc).isoformat(),
        "n_total": info_global["n_total"],
        "pct_atendieron": info_global["pct_asistio"],
        "pct_noshow_global": info_global["pct_noshow"],
        "sms_pct_global": info_inversion["pct_sms_global"],
        "n_sms0": info_inversion["n_sms0"],
        "n_sms1": info_inversion["n_sms1"],
        "pct_noshow_sms0": info_inversion["pct_noshow_sms0"],
        "pct_noshow_sms1": info_inversion["pct_noshow_sms1"],
        "diff_pp_sms1_menos_sms0": info_inversion["diff_pp_sms1_menos_sms0"],
        "inversion_signo_confirmada": bool(
            info_inversion["diff_pp_sms1_menos_sms0"] > 0
        ),
        "lead_time_mediana": info_lead_time["mediana"],
        "lead_time_p90": info_lead_time["p90"],
        "lead_time_p99": info_lead_time["p99"],
        "lead_time_maximo": info_lead_time["maximo"],
        "n_lead_time_negativo": info_lead_time["n_negativo"],
        "n_lead_time_mismo_dia": info_lead_time["n_mismo_dia"],
        "pct_mismo_dia": info_lead_time["pct_mismo_dia"],
        "n_pacientes_unicos": info_pacientes["n_pacientes_unicos"],
        "media_citas_por_paciente": info_pacientes["media_citas_por_paciente"],
        "p95_citas_por_paciente": info_pacientes["p95_citas_por_paciente"],
        "max_citas_por_paciente": info_pacientes["max_citas_por_paciente"],
        "pct_pacientes_una_sola_cita": info_pacientes["pct_pacientes_una_sola_cita"],
        "noshow_primeras_visitas": info_pacientes["noshow_primeras_visitas"],
        "noshow_repetidas": info_pacientes["noshow_repetidas"],
        "top_corr_con_showed_up": dict(top_corr_outcome),
        "top_corr_con_sms_received": dict(top_corr_tratamiento),
        "correlaciones_metodo": "pearson",
        "correlaciones_nan_handling": "pairwise (pandas .corr() por defecto)",
        "n_prior_noshow_rate_no_nulos": int(df["prior_noshow_rate"].notna().sum()),
        "n_anomalia_appt_lt_sched": int(
            (df["AppointmentDay"] < df["ScheduledDay"]).sum()
        ),
        "dias_semana_presentes": sorted(
            int(d) for d in df["appointment_weekday"].unique()
        ),
    }
    ruta_json = Path(ruta_json)
    ruta_json.parent.mkdir(parents=True, exist_ok=True)
    with ruta_json.open("w", encoding="utf-8") as f:
        json.dump(metadatos, f, indent=2, ensure_ascii=False)
    logger.info("Metadatos NB02 guardados en %s", ruta_json)
    return metadatos

top_outcome = corr_outcome.head(5).to_dict()
top_trat = corr_tratamiento.head(5).to_dict()
metadatos = eda.escribir_metadatos_nb02(
    REPORTES / "nb02_metadatos_v1.json",
    df=df,
    info_global=info_global,
    info_inversion=info_inversion,
    info_lead_time=info_lead_time,
    info_pacientes=info_pacientes,
    top_corr_outcome=top_outcome,
```

```
    top_corr_tratamiento=top_trat,
)
print("\nMetadatos NB02 persistidos. Resumen ejecutivo:")
print(f"  Tasa no-show global       : {100 * metadatos['pct_noshow_global']:.2f}%")
print(f"  Δ no-show (SMS=1 − SMS=0) : {100 * metadatos['diff_pp_sms1_menos_sms0']:+.2f} pp "
      f"(inversión confirmada: {metadatos['inversion_signo_confirmada']})")
print(f"  Lead time mediana / P99   : {metadatos['lead_time_mediana']:.0f} / "
      f"{metadatos['lead_time_p99']:.0f} días")
print(f"  Pacientes únicos / N total: {metadatos['n_pacientes_unicos']:,} / "
      f"{metadatos['n_total']:,}".replace(",", "."))
print(f"  Anomalías appt<sched      : {metadatos['n_anomalia_appt_lt_sched']}")
print(f"  Días semana presentes     : {metadatos['dias_semana_presentes']} "
      f"(0=lun … 6=dom)")


FIGURAS_NB02 = [
    "nb02_noshow_por_variables_v1.png",
    "nb02_noshow_top20_barrios_v1.png",
    "nb02_correlacion_heatmap_v1.png",
    "nb02_correlaciones_dual_v1.png",
    "nb02_sms_asignacion_v1.png",
    "nb02_inversion_signo_sms_v1.png",
    "nb02_lead_time_distribucion_v1.png",
    "nb02_pacientes_distribucion_v1.png",
]
for nombre in FIGURAS_NB02:
    ruta = FIGURAS / nombre
    assert ruta.exists() and ruta.stat().st_size > 0, f"Figura faltante o vacía: {ruta}"
import json as _json
with open(REPORTES / "nb02_metadatos_v1.json", "r", encoding="utf-8") as _f:
    _md = _json.load(_f)
assert _md["inversion_signo_confirmada"] is True, (
    "El sidecar NO confirma la inversión de signo del SMS. "
    "Revisar la sección 4 antes de continuar."
)
assert _md["n_total"] == len(df), "Discrepancia entre filas y sidecar"
print(f"NB02 completado. {len(FIGURAS_NB02)} figuras + 1 sidecar JSON generados.")
print(f"  Figuras en : {FIGURAS}")
print(f"  Sidecar    : {REPORTES / 'nb02_metadatos_v1.json'}")
```

B.3 Modelo predictivo: XGBoost, calibración e interpretabilidad (SHAP)

src/modelado.py · notebooks/03_modelo_xgboost_calibracion_shap.ipynb

```
from __future__ import annotations
import json
import logging
import pickle
from dataclasses import dataclass, asdict, field
from datetime import datetime, timezone
from pathlib import Path
from typing import Any, Iterable, Mapping
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import shap
import xgboost as xgb
from sklearn.calibration import CalibratedClassifierCV, calibration_curve
from sklearn.frozen import FrozenEstimator
from sklearn.metrics import (
    average_precision_score,
    brier_score_loss,
    confusion_matrix,
    precision_score,
    recall_score,
    roc_auc_score,
)
logger = logging.getLogger(__name__)
FEATURES_NB03: list[str] = [
    "lead_time",
    "Age",
    "gender_F",
    "neighbourhood_encoded",
    "Scholarship",
    "Hipertension",
    "Diabetes",
    "Alcoholism",
    "Handcap",
```

```
    "comorbidity_count",
    "chronic_flag",
    "scheduled_weekday",
    "scheduled_month",
    "appointment_weekday",
    "appointment_month",
    "prior_appointment_count",
    "prior_noshow_rate",
    "is_first_visit",
]
ETIQUETAS_FEATURES: dict[str, str] = {
    "lead_time": "antelación (días)",
    "Age": "edad",
    "gender_F": "género (F=1)",
    "neighbourhood_encoded": "barrio (frecuencia)",
    "Scholarship": "beca Bolsa Família",
    "Hipertension": "hipertensión",
    "Diabetes": "diabetes",
    "Alcoholism": "alcoholismo",
    "Handcap": "discapacidad",
    "comorbidity_count": "nº comorbilidades",
    "chronic_flag": "cualquier comorbilidad",
    "scheduled_weekday": "día semana programación",
    "scheduled_month": "mes programación",
    "appointment_weekday": "día semana cita",
    "appointment_month": "mes cita",
    "prior_appointment_count": "nº citas previas observadas",
    "prior_noshow_rate": "tasa no-show previa observada",
    "is_first_visit": "primera visita observada",
}
GRID_MAX_DEPTH: list[int] = [4, 6, 8]
GRID_LEARNING_RATE: list[float] = [0.05, 0.1, 0.2]
N_ESTIMATORS_CAP: int = 1000
EARLY_STOPPING_ROUNDS: int = 50
SUBSAMPLE: float = 0.8
COLSAMPLE_BYTREE: float = 0.8
DELTA_BRIER_TIEBREAK: float = 1e-3   # si |Brier_iso - Brier_platt| < ε → Platt
N_FOLDS_WALK_FORWARD: int = 3
FRAC_TRAIN_XGB: float = 0.80   # primer 80% del periodo de train
N_SHAP_VISUALIZACION: int = 5000
CASOS_LOCALES_ETIQUETAS: dict[str, str] = {
    "caso1_alto_riesgo_y_noshow":
        "Caso 1 — alto riesgo predicho y no-show observado",
    "caso2_alto_riesgo_sin_noshow":
        "Caso 2 — alto riesgo predicho pero asistió",
    "caso3_bajo_riesgo_con_noshow":
        "Caso 3 — bajo riesgo predicho con no-show observado",
}

def configurar_estilo() -> None:
    sns.set_theme(
        style="whitegrid",
        context="notebook",
        palette="colorblind",
        font_scale=1.0,
    )
    plt.rcParams.update({
        "figure.dpi": 110,
        "savefig.dpi": 200,
        "savefig.bbox": "tight",
        "axes.titleweight": "semibold",
        "axes.titlesize": 12,
        "axes.labelsize": 11,
    })

%load_ext autoreload
%autoreload 2
import logging
import sys
from pathlib import Path
import numpy as np
import pandas as pd
ROOT = Path.cwd().parent if Path.cwd().name == "notebooks" else Path.cwd()
if str(ROOT) not in sys.path:
    sys.path.insert(0, str(ROOT))
logging.basicConfig(level=logging.INFO, format="%(levelname)s %(message)s")
SEED = rutas.SEED
np.random.seed(SEED)
```

```
modelado.configurar_estilo()
print(f"Seed fijado: {SEED}")


def cargar_train_test_nb03(
    ruta_train: str | Path, ruta_test: str | Path
) -> tuple[pd.DataFrame, pd.DataFrame]:
    parse = ["ScheduledDay", "AppointmentDay"]
    train = pd.read_csv(ruta_train, parse_dates=parse)
    test = pd.read_csv(ruta_test, parse_dates=parse)
    logger.info(
        "Train cargado: %d filas, Test: %d filas. Train spans %s → %s.",
        len(train), len(test),
        train["AppointmentDay"].min().date(),
        train["AppointmentDay"].max().date(),
    )
    return train, test


def verificar_codificacion_showed_up_pred(df: pd.DataFrame) -> None:
    pct_atendieron = float((df["Showed_up"] == 1).mean())
    if not 0.65 < pct_atendieron < 0.90:
        raise ValueError(
            f"Showed_up tiene una tasa de asistencia inesperada ({pct_atendieron:.1%}); "
            "verificar codificación antes de entrenar."
        )
    logger.info(
        "Showed_up verificado: %.1f%% asistencia. Target del modelo = 1 - Showed_up (no-show).",
        pct_atendieron * 100,
    )

train_df, test_df = modelado.cargar_train_test_nb03(rutas.CSV_TRAIN, rutas.CSV_TEST)
modelado.verificar_codificacion_showed_up_pred(train_df)
modelado.verificar_codificacion_showed_up_pred(test_df)
print(f"train: {train_df.shape}, test: {test_df.shape}")
print(f"train         AppointmentDay:         {train_df['AppointmentDay'].min().date()}        →
{train_df['AppointmentDay'].max().date()}")
print(f"test              AppointmentDay:       {test_df['AppointmentDay'].min().date()}       →
{test_df['AppointmentDay'].max().date()}")

def preparar_features(df: pd.DataFrame) -> tuple[pd.DataFrame, pd.Series]:
    df = df.copy()
    df["gender_F"] = (df["Gender"] == "F").astype(int)
    for col in (
        "Scholarship", "Hipertension", "Diabetes",
        "Alcoholism", "Handcap", "chronic_flag", "is_first_visit",
    ):
        if df[col].dtype == bool:
            df[col] = df[col].astype(int)
    X = df[FEATURES_NB03].copy()
    y = (1 - df["Showed_up"]).astype(int).rename("noshow")
    logger.info(
        "X: %s, prevalencia no-show: %.1f%%",
        X.shape, 100 * y.mean(),
    )
    return X, y

X_train_full, y_train_full = modelado.preparar_features(train_df)
X_test, y_test = modelado.preparar_features(test_df)
print(f"Features ({len(modelado.FEATURES_NB03)}): {modelado.FEATURES_NB03}")
print(f"\nX_train_full:       {X_train_full.shape},       prevalencia       no-show       train:
{y_train_full.mean():.3f}")
print(f"X_test:       {X_test.shape}, prevalencia no-show test:  {y_test.mean():.3f}")
print(f"\nNaN por columna (train, sólo columnas con NaN):")
print(X_train_full.isna().sum()[X_train_full.isna().sum() > 0])


def dividir_train_xgb_calib(
    X: pd.DataFrame,
    y: pd.Series,
    fechas: pd.Series,
    frac_xgb: float = FRAC_TRAIN_XGB,
) -> tuple[pd.DataFrame, pd.Series, pd.DataFrame, pd.Series, pd.Timestamp]:
    fecha_corte = fechas.quantile(frac_xgb, interpolation="lower")
    mask_xgb = fechas <= fecha_corte
    X_xgb, y_xgb = X.loc[mask_xgb], y.loc[mask_xgb]
    X_cal, y_cal = X.loc[~mask_xgb], y.loc[~mask_xgb]
    logger.info(
        "Train→ train_xgb (≤ %s): %d filas; train_calib (> %s): %d filas.",
```

```
        pd.Timestamp(fecha_corte).date(), len(X_xgb),
        pd.Timestamp(fecha_corte).date(), len(X_cal),
    )
    return X_xgb, y_xgb, X_cal, y_cal, pd.Timestamp(fecha_corte)

X_xgb, y_xgb, X_calib, y_calib, fecha_corte_xgb_calib = modelado.dividir_train_xgb_calib(
    X_train_full, y_train_full, train_df["AppointmentDay"], frac_xgb=modelado.FRAC_TRAIN_XGB,
)
print(f"Corte cronológico: {fecha_corte_xgb_calib.date()}")
print(f"train_xgb:  {X_xgb.shape}, prev no-show: {y_xgb.mean():.3f}")
print(f"train_calib: {X_calib.shape}, prev no-show: {y_calib.mean():.3f}")


@dataclass
class FoldWalkForward:
    fold: int
    fecha_corte_train: pd.Timestamp
    fecha_corte_val: pd.Timestamp
    idx_train: np.ndarray
    idx_val: np.ndarray
    def resumen(self) -> dict[str, Any]:
        return {
            "fold": self.fold,
            "train_hasta": self.fecha_corte_train.date().isoformat(),
            "val_hasta": self.fecha_corte_val.date().isoformat(),
            "n_train": int(len(self.idx_train)),
            "n_val": int(len(self.idx_val)),
        }

def generar_folds_walk_forward(
    fechas: pd.Series, n_folds: int = N_FOLDS_WALK_FORWARD
) -> list[FoldWalkForward]:
    fechas = pd.Series(fechas).reset_index(drop=True)
    cuantiles = np.linspace(0.5, 1.0, n_folds + 1)
    cortes = [fechas.quantile(q, interpolation="lower") for q in cuantiles]
    folds: list[FoldWalkForward] = []
    for k in range(n_folds):
        corte_train, corte_val = cortes[k], cortes[k + 1]
        idx_train = np.where(fechas <= corte_train)[0]
        mask_val = (fechas > corte_train) & (fechas <= corte_val)
        idx_val = np.where(mask_val)[0]
        folds.append(FoldWalkForward(
            fold=k + 1,
            fecha_corte_train=pd.Timestamp(corte_train),
            fecha_corte_val=pd.Timestamp(corte_val),
            idx_train=idx_train,
            idx_val=idx_val,
        ))
    for f in folds:
        logger.info("Fold %d: %s", f.fold, f.resumen())
    return folds

folds = modelado.generar_folds_walk_forward(
    train_df["AppointmentDay"].loc[X_xgb.index], n_folds=modelado.N_FOLDS_WALK_FORWARD,
)
for f in folds:
    print(f.resumen())


def _calcular_scale_pos_weight(y: pd.Series) -> float:
    n_pos = int((y == 1).sum())
    n_neg = int((y == 0).sum())
    spw = n_neg / n_pos
    logger.info("scale_pos_weight = neg/pos = %d/%d = %.3f", n_neg, n_pos, spw)
    return spw


def _instanciar_xgb(
    max_depth: int,
    learning_rate: float,
    scale_pos_weight: float,
    n_estimators: int = N_ESTIMATORS_CAP,
    early_stopping_rounds: int | None = EARLY_STOPPING_ROUNDS,
    seed: int = 42,
) -> xgb.XGBClassifier:
    return xgb.XGBClassifier(
        objective="binary:logistic",
        eval_metric="auc",
        n_estimators=n_estimators,
```

```
        max_depth=max_depth,
        learning_rate=learning_rate,
        subsample=SUBSAMPLE,
        colsample_bytree=COLSAMPLE_BYTREE,
        scale_pos_weight=scale_pos_weight,
        early_stopping_rounds=early_stopping_rounds,
        tree_method="hist",
        random_state=seed,
        n_jobs=-1,
        verbosity=0,
    )

def buscar_hiperparametros(
    X: pd.DataFrame,
    y: pd.Series,
    folds: list[FoldWalkForward],
    grid_max_depth: Iterable[int] = GRID_MAX_DEPTH,
    grid_learning_rate: Iterable[float] = GRID_LEARNING_RATE,
    seed: int = 42,
) -> pd.DataFrame:
    spw = _calcular_scale_pos_weight(y)
    filas: list[dict[str, Any]] = []
    for max_depth in grid_max_depth:
        for lr in grid_learning_rate:
            aucs: list[float] = []
            best_iters: list[int] = []
            for f in folds:
                Xtr, ytr = X.iloc[f.idx_train], y.iloc[f.idx_train]
                Xva, yva = X.iloc[f.idx_val], y.iloc[f.idx_val]
                modelo = _instanciar_xgb(
                    max_depth=max_depth,
                    learning_rate=lr,
                    scale_pos_weight=spw,
                    seed=seed,
                )
                modelo.fit(Xtr, ytr, eval_set=[(Xva, yva)], verbose=False)
                pred = modelo.predict_proba(Xva)[:, 1]
                aucs.append(float(roc_auc_score(yva, pred)))
                best_iters.append(int(modelo.best_iteration))
            fila = {
                "max_depth": max_depth,
                "learning_rate": lr,
                "auc_mean": float(np.mean(aucs)),
                "auc_std": float(np.std(aucs)),
                "best_iteration_mean": float(np.mean(best_iters)),
                **{f"auc_fold{i+1}": a for i, a in enumerate(aucs)},
                **{f"best_iter_fold{i+1}": b for i, b in enumerate(best_iters)},
            }
            logger.info(
                "Combo md=%d lr=%.2f → AUC=%.4f±%.4f, best_iter≈%.0f",
                max_depth, lr, fila["auc_mean"], fila["auc_std"], fila["best_iteration_mean"],
            )
            filas.append(fila)
    df = pd.DataFrame(filas).sort_values("auc_mean", ascending=False).reset_index(drop=True)
    return df

tabla_grid = modelado.buscar_hiperparametros(
    X_xgb, y_xgb, folds,
    grid_max_depth=modelado.GRID_MAX_DEPTH,
    grid_learning_rate=modelado.GRID_LEARNING_RATE,
    seed=SEED,
)
print("\nResultados del grid (ordenados por AUC media):")
display(tabla_grid)

def ajustar_modelo_final(
    X_xgb: pd.DataFrame,
    y_xgb: pd.Series,
    X_calib: pd.DataFrame,
    y_calib: pd.Series,
    mejores: Mapping[str, Any],
    seed: int = 42,
) -> xgb.XGBClassifier:
    spw = _calcular_scale_pos_weight(y_xgb)
    modelo = _instanciar_xgb(
        max_depth=int(mejores["max_depth"]),
        learning_rate=float(mejores["learning_rate"]),
        scale_pos_weight=spw,
```

```
        seed=seed,
    )
    modelo.fit(X_xgb, y_xgb, eval_set=[(X_calib, y_calib)], verbose=False)
    logger.info(
        "Modelo final ajustado (md=%d, lr=%.2f, best_iteration=%d).",
        modelo.max_depth, modelo.learning_rate, modelo.best_iteration,
    )
    return modelo

mejores = tabla_grid.iloc[0].to_dict()
print(f"Combo                  elegido:                  max_depth={int(mejores['max_depth'])},
learning_rate={mejores['learning_rate']}")
print(f"  AUC CV media: {mejores['auc_mean']:.4f} ± {mejores['auc_std']:.4f}")
print(f"  best_iteration medio CV: {mejores['best_iteration_mean']:.0f}\n")
modelo = modelado.ajustar_modelo_final(X_xgb, y_xgb, X_calib, y_calib, mejores, seed=SEED)
print(f"\nModelo   final:   max_depth={modelo.max_depth},   learning_rate={modelo.learning_rate},
best_iteration={modelo.best_iteration}")

def ajustar_calibradores(
    modelo: xgb.XGBClassifier,
    X_calib: pd.DataFrame,
    y_calib: pd.Series,
    delta_tiebreak: float = DELTA_BRIER_TIEBREAK,
) -> tuple[CalibratedClassifierCV, pd.DataFrame]:
    frozen = FrozenEstimator(modelo)
    resultados: dict[str, dict[str, Any]] = {}
    for metodo in ("sigmoid", "isotonic"):
        cal = CalibratedClassifierCV(frozen, method=metodo).fit(X_calib, y_calib)
        prob_cal = cal.predict_proba(X_calib)[:, 1]
        brier = float(brier_score_loss(y_calib, prob_cal))
        resultados[metodo] = {"calibrador": cal, "brier_calib": brier}
    brier_raw = float(brier_score_loss(
        y_calib, modelo.predict_proba(X_calib)[:, 1]
    ))
    brier_platt = resultados["sigmoid"]["brier_calib"]
    brier_iso = resultados["isotonic"]["brier_calib"]
    diff = brier_iso - brier_platt
    if abs(diff) < delta_tiebreak:
        elegido, motivo = "sigmoid", f"diferencia |Δ|={abs(diff):.5f} < ε={delta_tiebreak} →
default Platt"
    elif brier_platt <= brier_iso:
        elegido, motivo = "sigmoid", f"Platt mejor por {-diff:.5f}"
    else:
        elegido, motivo = "isotonic", f"Isotonic mejor por {diff:.5f}"
    logger.info(
        "Brier en train_calib: raw=%.5f, Platt=%.5f, Isotonic=%.5f. Elegido: %s (%s).",
        brier_raw, brier_platt, brier_iso, elegido, motivo,
    )
    tabla = pd.DataFrame([
        {"metodo": "sin_calibrar", "brier_train_calib": brier_raw, "seleccionado": False},
        {"metodo": "platt_sigmoid", "brier_train_calib": brier_platt, "seleccionado": elegido ==
"sigmoid"},
        {"metodo": "isotonic", "brier_train_calib": brier_iso, "seleccionado": elegido ==
"isotonic"},
    ])
    tabla.attrs["motivo_seleccion"] = motivo
    tabla.attrs["metodo_elegido"] = elegido
    return resultados[elegido]["calibrador"], tabla

calibrador, tabla_calibradores = modelado.ajustar_calibradores(modelo, X_calib, y_calib)
print(f"Calibrador elegido: {tabla_calibradores.attrs['metodo_elegido']}")
print(f"Motivo: {tabla_calibradores.attrs['motivo_seleccion']}\n")
display(tabla_calibradores)

@dataclass
class MetricasTest:
    auc_roc: float
    pr_auc: float
    brier_raw: float
    brier_calibrado: float
    prevalencia: float
    def as_dict(self) -> dict[str, float]:
        return asdict(self)

def calcular_metricas_test(
    modelo: xgb.XGBClassifier,
    calibrador: CalibratedClassifierCV,
```

```
    X_test: pd.DataFrame,
    y_test: pd.Series,
) -> tuple[MetricasTest, np.ndarray, np.ndarray]:
    prob_raw = modelo.predict_proba(X_test)[:, 1]
    prob_cal = calibrador.predict_proba(X_test)[:, 1]
    m = MetricasTest(
        auc_roc=float(roc_auc_score(y_test, prob_cal)),
        pr_auc=float(average_precision_score(y_test, prob_cal)),
        brier_raw=float(brier_score_loss(y_test, prob_raw)),
        brier_calibrado=float(brier_score_loss(y_test, prob_cal)),
        prevalencia=float(y_test.mean()),
    )
    logger.info(
        "Test: AUC=%.4f, PR-AUC=%.4f, Brier raw=%.4f, Brier calibrado=%.4f, prev=%.3f",
        m.auc_roc, m.pr_auc, m.brier_raw, m.brier_calibrado, m.prevalencia,
    )
    return m, prob_raw, prob_cal

metricas, prob_raw, prob_cal = modelado.calcular_metricas_test(modelo, calibrador, X_test, y_test)
print(f"\nMétricas en test:")
print(f"  AUC-ROC:           {metricas.auc_roc:.4f}")
print(f"  PR-AUC:            {metricas.pr_auc:.4f}")
print(f"  Brier raw:         {metricas.brier_raw:.4f}")
print(f"    Brier  calibrado:       {metricas.brier_calibrado:.4f}     (Δ={metricas.brier_raw   -
metricas.brier_calibrado:+.4f})")
print(f"  Prevalencia test:  {metricas.prevalencia:.4f}")

def matriz_confusion_tres_umbrales(
    y_test: pd.Series, prob_cal: np.ndarray
) -> pd.DataFrame:
    prev = float(y_test.mean())
    umbral_top20 = float(np.quantile(prob_cal, 0.80))
    filas = []
    for nombre, umbral in [
        ("top20", umbral_top20),
        ("literatura_0.5", 0.5),
        ("prevalencia", prev),
    ]:
        y_pred = (prob_cal >= umbral).astype(int)
        tn, fp, fn, tp = confusion_matrix(y_test, y_pred, labels=[0, 1]).ravel()
        filas.append({
            "umbral_nombre": nombre,
            "umbral_valor": float(umbral),
            "TN": int(tn), "FP": int(fp), "FN": int(fn), "TP": int(tp),
            "precision": float(precision_score(y_test, y_pred, zero_division=0)),
            "recall": float(recall_score(y_test, y_pred, zero_division=0)),
            "n_positivos_predichos": int(y_pred.sum()),
            "pct_positivos_predichos": float(y_pred.mean()),
        })
    df = pd.DataFrame(filas)
    logger.info(
        "Matriz de confusión calculada a 3 umbrales: top20=%.4f, 0.5, prev=%.4f",
        umbral_top20, prev,
    )
    return df

tabla_confusion = modelado.matriz_confusion_tres_umbrales(y_test, prob_cal)
display(tabla_confusion)

def plot_curva_calibracion(
    y_test: pd.Series,
    prob_raw: np.ndarray,
    prob_cal: np.ndarray,
    ruta_salida: str | Path,
    n_bins: int = 10,
) -> Path:
    frac_raw, mean_raw = calibration_curve(y_test, prob_raw, n_bins=n_bins, strategy="quantile")
    frac_cal, mean_cal = calibration_curve(y_test, prob_cal, n_bins=n_bins, strategy="quantile")
    fig, ax = plt.subplots(1, 1, figsize=(6, 6))
    ax.plot([0, 1], [0, 1], linestyle="--", color="grey", label="calibración perfecta")
    ax.plot(mean_raw, frac_raw, marker="o", label="raw (sin calibrar)")
    ax.plot(mean_cal, frac_cal, marker="s", label="calibrado")
    ax.set_xlabel("probabilidad media predicha por bin")
    ax.set_ylabel("frecuencia observada de no-show")
    ax.set_title("Curva de calibración (test, cuantiles)")
    ax.legend(loc="upper left")
    ax.set_xlim(0, 1)
```

```
    ax.set_ylim(0, 1)
    ruta_salida = Path(ruta_salida)
    ruta_salida.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(ruta_salida)
    plt.close(fig)
    logger.info("Curva de calibración guardada en %s", ruta_salida)
    return ruta_salida


def _curva_calibracion_uniforme_filtrada(
    y: np.ndarray,
    prob: np.ndarray,
    bordes: np.ndarray,
    min_obs_bin: int,
) -> tuple[np.ndarray, np.ndarray, np.ndarray]:
    asignacion = np.digitize(prob, bordes, right=False) - 1
    asignacion = np.clip(asignacion, 0, len(bordes) - 2)
    n_bins = len(bordes) - 1
    medias = np.full(n_bins, np.nan)
    fracs = np.full(n_bins, np.nan)
    cuentas = np.zeros(n_bins, dtype=int)
    for k in range(n_bins):
        mascara = asignacion == k
        cuentas[k] = int(mascara.sum())
        if cuentas[k] >= min_obs_bin:
            medias[k] = float(prob[mascara].mean())
            fracs[k] = float(y[mascara].mean())
    valido = ~np.isnan(medias)
    return medias[valido], fracs[valido], cuentas[valido]


def plot_curva_calibracion_sanity(
    y_internal: pd.Series,
    prob_cal_internal: np.ndarray,
    y_test: pd.Series,
    prob_cal_test: np.ndarray,
    ruta_salida: str | Path,
    n_bins: int = 10,
    min_obs_bin: int = 30,
) -> Path:
    y_i = pd.Series(y_internal).reset_index(drop=True).to_numpy()
    y_t = pd.Series(y_test).reset_index(drop=True).to_numpy()
    rango_max = float(max(prob_cal_internal.max(), prob_cal_test.max()))
    lim_sup = min(1.0, rango_max + 0.02)
    bordes = np.linspace(0.0, lim_sup, n_bins + 1)
    mean_i, frac_i, n_i = _curva_calibracion_uniforme_filtrada(
        y_i, prob_cal_internal, bordes, min_obs_bin=min_obs_bin
    )
    mean_t, frac_t, n_t = _curva_calibracion_uniforme_filtrada(
        y_t, prob_cal_test, bordes, min_obs_bin=min_obs_bin
    )
    fig, ax = plt.subplots(1, 1, figsize=(7, 6))
    ax.plot([0, lim_sup], [0, lim_sup], linestyle="--", color="grey",
            label="calibración perfecta")
    ax.plot(mean_i, frac_i, marker="o", label=f"slice interno (n={len(y_i)})")
    ax.plot(mean_t, frac_t, marker="s", label=f"test (n={len(y_t)})")
    ax.set_xlabel("probabilidad media predicha por bin (bins uniformes)")
    ax.set_ylabel("frecuencia observada de no-show")
    ax.set_title(
        f"Robustez de la calibración: slice interno vs test\n"
        f"(bins con < {min_obs_bin} obs descartados; eje zoom al rango efectivo)"
    )
    ax.legend(loc="upper left")
    ax.set_xlim(0, lim_sup)
    ax.set_ylim(0, lim_sup)
    ruta_salida = Path(ruta_salida)
    ruta_salida.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(ruta_salida)
    plt.close(fig)
    logger.info(
        "Sanity check guardado en %s (slice: %d bins válidos, test: %d)",
        ruta_salida, len(mean_i), len(mean_t),
    )
    return ruta_salida


modelado.plot_curva_calibracion(
    y_test, prob_raw, prob_cal,
    ruta_salida=rutas.FIGURAS / "nb03_calibracion_curva_v1.png",
)
fechas_xgb = train_df["AppointmentDay"].loc[X_xgb.index]
```

```
corte_interno = fechas_xgb.quantile(0.85, interpolation="lower")
mask_interno = fechas_xgb > corte_interno
X_interno = X_xgb.loc[mask_interno]
y_interno = y_xgb.loc[mask_interno]
prob_interno = calibrador.predict_proba(X_interno)[:, 1]
print(f"Slice      interno      (sanity):      {mask_interno.sum()}      filas,      fechas      >
{pd.Timestamp(corte_interno).date()}")
modelado.plot_curva_calibracion_sanity(
    y_interno, prob_interno, y_test, prob_cal,
    ruta_salida=rutas.FIGURAS / "nb03_calibracion_sanity_v1.png",
)
```

```
def calcular_shap_values_test(
    modelo: xgb.XGBClassifier, X_test: pd.DataFrame
) -> shap.Explanation:
    explicador = shap.TreeExplainer(modelo)
    valores = explicador(X_test)
    logger.info("SHAP values calculados sobre test: %s", valores.values.shape)
    return valores


def submuestra_estratificada_shap(
    X_test: pd.DataFrame,
    y_test: pd.Series,
    prob_cal: np.ndarray,
    n: int = N_SHAP_VISUALIZACION,
    seed: int = 42,
) -> np.ndarray:
    cuartil_riesgo = pd.qcut(prob_cal, q=4, labels=False, duplicates="drop")
    estrato        =        pd.Series(cuartil_riesgo).astype(str)        +        "_y"        +
y_test.astype(str).reset_index(drop=True)
    df_strata = pd.DataFrame({"estrato": estrato.values})
    n_real = min(n, len(df_strata))
    rng = np.random.RandomState(seed)
    grupos = df_strata.groupby("estrato").indices
    idxs: list[int] = []
    for estrato_nombre, idx_grupo in grupos.items():
        n_estrato = max(1, int(round(n_real * len(idx_grupo) / len(df_strata))))
        n_estrato = min(n_estrato, len(idx_grupo))
        elegidos = rng.choice(idx_grupo, size=n_estrato, replace=False)
        idxs.extend(elegidos.tolist())
    idxs = np.array(sorted(idxs))
    logger.info(
        "Submuestra SHAP estratificada: %d filas en %d estratos.",
        len(idxs), len(grupos),
    )
    return idxs


shap_values = modelado.calcular_shap_values_test(modelo, X_test)
idx_muestra     =     modelado.submuestra_estratificada_shap(X_test,     y_test,     prob_cal,
n=modelado.N_SHAP_VISUALIZACION, seed=SEED)
importancia_global = pd.Series(
    np.abs(shap_values.values).mean(axis=0),
    index=X_test.columns,
).sort_values(ascending=False)
print("Top 10 features por |SHAP| media (test completo):")
print(importancia_global.head(10).to_string())


def plot_shap_summary(
    shap_values: shap.Explanation,
    X_test: pd.DataFrame,
    idx_muestra: np.ndarray,
    ruta_salida: str | Path,
) -> Path:
    sub = shap_values[idx_muestra]
    feature_names_es = [ETIQUETAS_FEATURES.get(c, c) for c in X_test.columns]
    sub.feature_names = feature_names_es
    plt.figure(figsize=(9, 7))
    shap.plots.beeswarm(sub, show=False, max_display=18)
    fig = plt.gcf()
    fig.suptitle("Importancia y dirección de las variables (SHAP, beeswarm)", y=0.995)
    ruta_salida = Path(ruta_salida)
    ruta_salida.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(ruta_salida)
    plt.close(fig)
    logger.info("SHAP summary guardado en %s", ruta_salida)
    return ruta_salida
```

```
def plot_shap_dependence(
    shap_values: shap.Explanation,
    X_test: pd.DataFrame,
    idx_muestra: np.ndarray,
    features: Iterable[str],
    ruta_salida: str | Path,
) -> Path:
    features = list(features)
    sub_shap = shap_values.values[idx_muestra]
    sub_X = X_test.iloc[idx_muestra].reset_index(drop=True)
    n = len(features)
    fig, axes = plt.subplots(1, n, figsize=(5 * n, 4.5))
    if n == 1:
        axes = [axes]
    for ax, feat in zip(axes, features):
        j = list(X_test.columns).index(feat)
        ax.scatter(sub_X[feat], sub_shap[:, j], s=5, alpha=0.3)
        ax.axhline(0, color="grey", linestyle="--", linewidth=0.8)
        ax.set_xlabel(ETIQUETAS_FEATURES.get(feat, feat))
        ax.set_ylabel("SHAP value (impacto sobre log-odds de no-show)")
        ax.set_title(ETIQUETAS_FEATURES.get(feat, feat))
    fig.suptitle("Dependencia: feature vs su SHAP value")
    fig.tight_layout()
    ruta_salida = Path(ruta_salida)
    ruta_salida.parent.mkdir(parents=True, exist_ok=True)
    fig.savefig(ruta_salida)
    plt.close(fig)
    logger.info("SHAP dependence guardado en %s", ruta_salida)
    return ruta_salida


modelado.plot_shap_summary(
    shap_values, X_test, idx_muestra,
    ruta_salida=rutas.FIGURAS / "nb03_shap_beeswarm_v1.png",
)
top4 = [c for c in importancia_global.head(8).index if X_test[c].nunique() > 5][:4]
print(f"Dependence plots para: {top4}")
modelado.plot_shap_dependence(
    shap_values, X_test, idx_muestra, features=top4,
    ruta_salida=rutas.FIGURAS / "nb03_shap_dependence_v1.png",
)

def elegir_casos_locales(
    y_test: pd.Series, prob_cal: np.ndarray, seed: int = 42
) -> dict[str, int]:
    rng = np.random.RandomState(seed)
    p90 = np.quantile(prob_cal, 0.90)
    p25 = np.quantile(prob_cal, 0.25)
    y = y_test.reset_index(drop=True).values
    def primero(mascara: np.ndarray) -> int:
        candidatos = np.where(mascara)[0]
        return int(rng.choice(candidatos))
    casos = {
        "caso1_alto_riesgo_y_noshow": primero((prob_cal >= p90) & (y == 1)),
        "caso2_alto_riesgo_sin_noshow": primero((prob_cal >= p90) & (y == 0)),
        "caso3_bajo_riesgo_con_noshow": primero((prob_cal <= p25) & (y == 1)),
    }
    logger.info("Casos locales SHAP: %s", casos)
    return casos


def plot_shap_local_waterfall_individuales(
    shap_values: shap.Explanation,
    X_test: pd.DataFrame,
    casos: Mapping[str, int],
    prob_cal: np.ndarray,
    directorio_salida: str | Path,
    prefijo: str = "nb03_shap_local_waterfall",
    max_display: int = 8,
) -> dict[str, Path]:
    feature_names_es = [ETIQUETAS_FEATURES.get(c, c) for c in X_test.columns]
    directorio_salida = Path(directorio_salida)
    directorio_salida.parent.mkdir(parents=True, exist_ok=True)
    rutas: dict[str, Path] = {}
    for clave, idx in casos.items():
        explicacion = shap_values[idx]
        explicacion.feature_names = feature_names_es
        shap.plots.waterfall(explicacion, show=False, max_display=max_display)
        fig = plt.gcf()
        fig.set_size_inches(9, 6)
```

```
        etiqueta = CASOS_LOCALES_ETIQUETAS.get(clave, clave)
        prob = float(prob_cal[idx])
        fig.suptitle(
            f"{etiqueta}\nProbabilidad calibrada de no-show: {prob:.1%}",
            fontsize=12, y=1.02,
        )
        ruta = directorio_salida / f"{prefijo}_{clave}_v1.png"
        fig.savefig(ruta, bbox_inches="tight")
        plt.close(fig)
        logger.info("SHAP local %s guardado en %s (prob_cal=%.3f)", clave, ruta, prob)
        rutas[clave] = ruta
    return rutas

casos = modelado.elegir_casos_locales(y_test, prob_cal, seed=SEED)
print(f"Casos locales (iloc): {casos}")
rutas_waterfall = modelado.plot_shap_local_waterfall_individuales(
    shap_values, X_test, casos, prob_cal=prob_cal,
    directorio_salida=rutas.FIGURAS,
    prefijo="nb03_shap_local_waterfall",
)
for clave, ruta in rutas_waterfall.items():
    print(f"  {clave}: {ruta.name} (prob_cal = {prob_cal[casos[clave]]:.3f})")

def guardar_modelo(modelo: xgb.XGBClassifier, ruta: str | Path) -> Path:
    ruta = Path(ruta)
    ruta.parent.mkdir(parents=True, exist_ok=True)
    modelo.save_model(str(ruta))
    logger.info("Modelo XGBoost guardado en %s", ruta)
    return ruta

def guardar_calibrador(
    calibrador: CalibratedClassifierCV, ruta: str | Path
) -> Path:
    ruta = Path(ruta)
    ruta.parent.mkdir(parents=True, exist_ok=True)
    with ruta.open("wb") as f:
        pickle.dump(calibrador, f)
    logger.info("Calibrador guardado en %s", ruta)
    return ruta

def exportar_probabilidades_test(
    df_test: pd.DataFrame,
    prob_cal: np.ndarray,
    prob_raw: np.ndarray,
    ruta_csv: str | Path,
) -> Path:
    salida = pd.DataFrame({
        "PatientId": df_test["PatientId"].values,
        "AppointmentID": df_test["AppointmentID"].values,
        "AppointmentDay": df_test["AppointmentDay"].values,
        "Showed_up": df_test["Showed_up"].values,
        "noshow": (1 - df_test["Showed_up"]).values.astype(int),
        "prob_noshow_raw": prob_raw,
        "prob_noshow_calibrada": prob_cal,
    })
    ruta_csv = Path(ruta_csv)
    ruta_csv.parent.mkdir(parents=True, exist_ok=True)
    salida.to_csv(ruta_csv, index=False)
    logger.info("Probabilidades exportadas a %s (%d filas)", ruta_csv, len(salida))
    return ruta_csv

def escribir_metadatos_nb03(
    ruta_json: str | Path,
    *,
    seed: int,
    n_train: int,
    n_train_xgb: int,
    n_train_calib: int,
    n_test: int,
    fecha_corte_xgb_calib: pd.Timestamp,
    folds: list[FoldWalkForward],
    tabla_grid: pd.DataFrame,
    mejores_params: Mapping[str, Any],
    tabla_calibradores: pd.DataFrame,
    metricas_test: MetricasTest,
    tabla_confusion: pd.DataFrame,
    umbral_top20: float,
```

```
    features_usadas: list[str],
    scale_pos_weight: float,
    casos_locales: Mapping[str, int],
) -> dict[str, Any]:
    metadatos: dict[str, Any] = {
        "generado_en": datetime.now(timezone.utc).isoformat(),
        "seed": seed,
        "features_usadas": features_usadas,
        "n_features": len(features_usadas),
        "n_train": n_train,
        "n_train_xgb": n_train_xgb,
        "n_train_calib": n_train_calib,
        "n_test": n_test,
        "fecha_corte_xgb_calib": fecha_corte_xgb_calib.date().isoformat(),
        "scale_pos_weight": float(scale_pos_weight),
        "folds_walk_forward": [f.resumen() for f in folds],
        "mejores_hiperparametros": {
            "max_depth": int(mejores_params["max_depth"]),
            "learning_rate": float(mejores_params["learning_rate"]),
            "best_iteration_mean_cv": float(mejores_params["best_iteration_mean"]),
            "auc_mean_cv": float(mejores_params["auc_mean"]),
            "auc_std_cv": float(mejores_params["auc_std"]),
        },
        "grid_resultados": tabla_grid.to_dict(orient="records"),
        "calibracion": {
            "metodo_elegido": tabla_calibradores.attrs["metodo_elegido"],
            "motivo": tabla_calibradores.attrs["motivo_seleccion"],
            "tabla":
tabla_calibradores.drop(columns=["seleccionado"]).to_dict(orient="records"),
        },
        "test_metricas": metricas_test.as_dict(),
        "umbral_top20": float(umbral_top20),
        "matriz_confusion": tabla_confusion.to_dict(orient="records"),
        "casos_locales_idx_iloc": dict(casos_locales),
    }
    ruta_json = Path(ruta_json)
    ruta_json.parent.mkdir(parents=True, exist_ok=True)
    with ruta_json.open("w", encoding="utf-8") as f:
        json.dump(metadatos, f, indent=2, ensure_ascii=False, default=str)
    logger.info("Metadatos NB03 guardados en %s", ruta_json)
    return metadatos


ruta_modelo = modelado.guardar_modelo(modelo, rutas.MODELOS / "xgboost_v1.json")
ruta_calib = modelado.guardar_calibrador(calibrador, rutas.MODELOS / "calibrador_v1.pkl")
ruta_probs = modelado.exportar_probabilidades_test(
    test_df, prob_cal=prob_cal, prob_raw=prob_raw,
    ruta_csv=rutas.REPORTES / "noshow_probs_v1.csv",
)
umbral_top20    =    float(tabla_confusion.loc[tabla_confusion["umbral_nombre"]    ==    "top20",
"umbral_valor"].iloc[0])
spw_train = (y_train_full == 0).sum() / (y_train_full == 1).sum()
metadatos = modelado.escribir_metadatos_nb03(
    rutas.REPORTES / "nb03_metadatos_v1.json",
    seed=SEED,
    n_train=len(train_df),
    n_train_xgb=len(X_xgb),
    n_train_calib=len(X_calib),
    n_test=len(test_df),
    fecha_corte_xgb_calib=fecha_corte_xgb_calib,
    folds=folds,
    tabla_grid=tabla_grid,
    mejores_params=mejores,
    tabla_calibradores=tabla_calibradores,
    metricas_test=metricas,
    tabla_confusion=tabla_confusion,
    umbral_top20=umbral_top20,
    features_usadas=modelado.FEATURES_NB03,
    scale_pos_weight=float(spw_train),
    casos_locales=casos,
)
print(f"\nArtefactos guardados:")
print(f"  {ruta_modelo}")
print(f"  {ruta_calib}")
print(f"  {ruta_probs}")
print(f"  {rutas.REPORTES / 'nb03_metadatos_v1.json'}")


assert "SMS_received" not in modelado.FEATURES_NB03, "SMS_received NO debe estar en el modelo de
baseline."
```

```
assert "Showed_up" not in modelado.FEATURES_NB03, "Showed_up es el target, no feature."
assert metricas.auc_roc > 0.6, f"AUC sospechosamente baja: {metricas.auc_roc}"
assert metricas.brier_calibrado <= metricas.brier_raw + 0.01, "La calibración no debería empeorar
Brier sustancialmente."
assert (prob_cal >= 0).all() and (prob_cal <= 1).all(), "Probabilidades calibradas fuera de [0,
1]."
assert ruta_modelo.exists() and ruta_calib.exists() and ruta_probs.exists(), "Artefactos
persistidos faltantes."
for clave, ruta in rutas_waterfall.items():
    assert ruta.exists(), f"Waterfall faltante: {ruta}"
print("NB03 completado: modelo, calibrador y probabilidades persistidos.")
print(f"   AUC test = {metricas.auc_roc:.4f} | Brier calibrado = {metricas.brier_calibrado:.4f}")
print(f"   Waterfalls: {len(rutas_waterfall)} PNGs independientes.")
print(f"   Siguiente paso: NB04 (inferencia causal IPW).")
```

B.4 Inferencia causal por ponderación de propensión (IPW/ATO)

src/causal.py · notebooks/04_inferencia_causal_ipw.ipynb

```
from __future__ import annotations
import json
import logging
import pickle
from dataclasses import dataclass, field
from datetime import datetime, timezone
from pathlib import Path
from typing import Any
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
logger = logging.getLogger(__name__)
COVARIABLES_CONTINUAS: list[str] = [
    "lead_time",
    "Age",
    "prior_noshow_rate",
]
COVARIABLES_LINEALES: list[str] = [
    "gender_F",
    "neighbourhood_encoded",
    "Scholarship",
    "Hipertension",
    "Diabetes",
    "Alcoholism",
    "Handcap",
    "comorbidity_count",
    "chronic_flag",
    "scheduled_weekday",
    "scheduled_month",
    "appointment_weekday",
    "appointment_month",
    "prior_appointment_count",
    "is_first_visit",
]
COL_TRATAMIENTO: str = "SMS_received"
COL_OUTCOME: str = "Showed_up"
COL_CLUSTER: str = "PatientId"
ETIQUETAS_COVARIABLES: dict[str, str] = {
    "lead_time": "antelación (días)",
    "lead_time_sq": "antelación² (z²)",
    "Age": "edad",
    "Age_sq": "edad² (z²)",
    "prior_noshow_rate": "tasa no-show previa",
    "prior_noshow_rate_sq": "tasa no-show previa² (z²)",
    "gender_F": "género (F=1)",
    "neighbourhood_encoded": "barrio (frecuencia)",
    "Scholarship": "beca Bolsa Família",
    "Hipertension": "hipertensión",
    "Diabetes": "diabetes",
    "Alcoholism": "alcoholismo",
    "Handcap": "discapacidad",
    "comorbidity_count": "nº comorbilidades",
    "chronic_flag": "cualquier comorbilidad",
    "scheduled_weekday": "día semana programación",
    "scheduled_month": "mes programación",
    "appointment_weekday": "día semana cita",
    "appointment_month": "mes cita",
```

```
    "prior_appointment_count": "nº citas previas observadas",
    "is_first_visit": "primera visita observada",
    "lead_1_3": "antelación 1-3 días",
    "lead_4_7": "antelación 4-7 días",
    "lead_8_30": "antelación 8-30 días",
    "lead_30plus": "antelación >30 días",
}
N_BOOTSTRAP: int = 1000
SEED_BOOTSTRAP: int = 2026
TRIM_ESS_UMBRAL: float = 0.5
TRIM_ROBUSTEZ: list[tuple[float, float]] = [(1.0, 99.0), (5.0, 95.0)]
LOG_REG_C: float = 1.0
LOG_REG_MAX_ITER: int = 2000
LOG_REG_SOLVER: str = "lbfgs"
DPI_FIGURAS: int = 200
LEAD_TIME_BINS: list[float] = [-0.001, 0.5, 3.5, 7.5, 30.5, np.inf]
LEAD_TIME_BIN_LABELS: list[str] = [
    "lead_0", "lead_1_3", "lead_4_7", "lead_8_30", "lead_30plus",
]

def configurar_estilo() -> None:
    sns.set_theme(context="notebook", style="whitegrid", palette="deep")
    plt.rcParams.update({
        "figure.dpi": 100,
        "savefig.dpi": DPI_FIGURAS,
        "axes.titlesize": 12,
        "axes.labelsize": 10,
        "legend.fontsize": 9,
        "xtick.labelsize": 9,
        "ytick.labelsize": 9,
    })

from __future__ import annotations
import json
import logging
import sys
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import Image, display
RAIZ = Path.cwd().parent if Path.cwd().name == "notebooks" else Path.cwd()
sys.path.insert(0, str(RAIZ))
logging.basicConfig(level=logging.INFO,
                    format="%(asctime)s [%(levelname)s] %(message)s",
                    datefmt="%H:%M:%S")
causal.configurar_estilo()
REGENERAR_BOOTSTRAP = False

def cargar_dataset_completo(ruta: str | Path) -> pd.DataFrame:
    df = pd.read_csv(ruta, parse_dates=["ScheduledDay", "AppointmentDay"])
    logger.info(
        "full_clean cargado: %d filas, %d columnas, %d pacientes únicos.",
        len(df), df.shape[1], df[COL_CLUSTER].nunique(),
    )
    return df

def verificar_codificacion_showed_up(df: pd.DataFrame) -> None:
    pct_atendieron = float((df[COL_OUTCOME] == 1).mean())
    if not 0.65 < pct_atendieron < 0.90:
        raise ValueError(
            f"Showed_up tiene una tasa de asistencia inesperada "
            f"({pct_atendieron:.1%}); verificar codificación antes del IPW."
        )
    logger.info(
        "Showed_up verificado: %.2f%% asistencia (esperado ≈80%%).",
        pct_atendieron * 100,
    )

def verificar_codificacion_tratamiento(df: pd.DataFrame) -> dict[str, Any]:
    sms = df[COL_TRATAMIENTO]
    if sms.dtype == bool:
        sms = sms.astype(int)
    n_trat = int((sms == 1).sum())
    n_ctrl = int((sms == 0).sum())
    p_t = n_trat / (n_trat + n_ctrl)
    logger.info(
```

```
        "SMS_received: %d tratados (%.1f%%), %d controles (%.1f%%).",
        n_trat, p_t * 100, n_ctrl, (1 - p_t) * 100,
    )
    return {
        "p_tratamiento": float(p_t),
        "n_tratados": n_trat,
        "n_controles": n_ctrl,
    }

df = causal.cargar_dataset_completo(rutas.CSV_FULL_CLEAN)
causal.verificar_codificacion_showed_up(df)
info_t = causal.verificar_codificacion_tratamiento(df)
print(f"Filas: {len(df):,}  |  Pacientes únicos: {df[causal.COL_CLUSTER].nunique():,}")
print(f"Proporción marginal de tratados P(T=1) = {info_t['p_tratamiento']:.4f}")


@dataclass
class EstadisticasEstandarizacion:
    medias: dict[str, float]
    desviaciones: dict[str, float]
    def aplicar(self, serie: pd.Series, columna: str) -> pd.Series:
        return (serie - self.medias[columna]) / self.desviaciones[columna]


def _calcular_estadisticas_continuas(
    df: pd.DataFrame, continuas: list[str] = COVARIABLES_CONTINUAS
) -> EstadisticasEstandarizacion:
    medias, desv = {}, {}
    for col in continuas:
        serie = df[col].copy()
        if col == "prior_noshow_rate":
            serie = serie.fillna(0.0)
        medias[col] = float(serie.mean())
        desv[col] = float(serie.std(ddof=0))
    return EstadisticasEstandarizacion(medias=medias, desviaciones=desv)


def construir_matriz_propensidad(
    df: pd.DataFrame,
    estadisticas: EstadisticasEstandarizacion | None = None,
    lead_time_spec: str = "z_squared",
) -> tuple[pd.DataFrame, np.ndarray, np.ndarray, EstadisticasEstandarizacion]:
    if lead_time_spec not in {"z_squared", "categorical"}:
        raise ValueError(f"lead_time_spec desconocido: {lead_time_spec!r}")
    df = df.copy()
    df["gender_F"] = (df["Gender"] == "F").astype(int)
    binarias = [
        "Scholarship", "Hipertension", "Diabetes",
        "Alcoholism", "Handcap", "chronic_flag", "is_first_visit",
        COL_TRATAMIENTO,
    ]
    for col in binarias:
        if df[col].dtype == bool:
            df[col] = df[col].astype(int)
    df["prior_noshow_rate"] = df["prior_noshow_rate"].fillna(0.0)
    if estadisticas is None:
        estadisticas = _calcular_estadisticas_continuas(df)
    if lead_time_spec == "z_squared":
        continuas_efectivas = COVARIABLES_CONTINUAS
    else:  # categorical: Age y prior_noshow_rate siguen como z+z²
        continuas_efectivas = ["Age", "prior_noshow_rate"]
    bloques: list[pd.DataFrame] = []
    for col in continuas_efectivas:
        z = estadisticas.aplicar(df[col], col).rename(col)
        z_sq = (z ** 2).rename(f"{col}_sq")
        bloques.append(pd.concat([z, z_sq], axis=1))
    X_continuas = pd.concat(bloques, axis=1) if bloques else pd.DataFrame(index=df.index)
    X_lineales = df[COVARIABLES_LINEALES].copy()
    if lead_time_spec == "categorical":
        lead_bin = pd.cut(
            df["lead_time"], bins=LEAD_TIME_BINS, labels=LEAD_TIME_BIN_LABELS,
        )
        dummies = pd.get_dummies(lead_bin, prefix="", prefix_sep="", drop_first=True)
        dummies = dummies.astype(int)
        X_lineales = pd.concat([X_lineales, dummies], axis=1)
    X = pd.concat([X_continuas, X_lineales], axis=1)
    T = df[COL_TRATAMIENTO].to_numpy(dtype=int)
    Y = df[COL_OUTCOME].to_numpy(dtype=int)
    logger.info(
        "Matriz de propensidad construida (lead_time_spec=%s): "
```

```
        "X.shape=%s, %d tratados, %d controles.",
        lead_time_spec, X.shape, int(T.sum()), int((1 - T).sum()),
    )
    return X, T, Y, estadisticas

X, T, Y, stats = causal.construir_matriz_propensidad(df)
print(f"X: {X.shape[0]:,} filas × {X.shape[1]} covariables")
print("Covariables:", ", ".join(X.columns))

@dataclass
class ResultadoPropensidad:
    modelo: LogisticRegression
    e_x: np.ndarray                       # P(T=1 | X) por fila
    auc: float                            # AUC del propensity sobre X→T
    p_tratamiento: float                  # P(T=1) marginal
    columnas: list[str]                   # orden de columnas usado
    estadisticas: EstadisticasEstandarizacion
    diagnostico_distribucion: dict[str, float] = field(default_factory=dict)


def estimar_propensity_score(
    X: pd.DataFrame,
    T: np.ndarray,
    estadisticas: EstadisticasEstandarizacion,
    *,
    C: float = LOG_REG_C,
    max_iter: int = LOG_REG_MAX_ITER,
    solver: str = LOG_REG_SOLVER,
) -> ResultadoPropensidad:
    modelo = LogisticRegression(C=C, max_iter=max_iter, solver=solver)
    modelo.fit(X, T)
    e_x = modelo.predict_proba(X)[:, 1]
    auc = float(roc_auc_score(T, e_x))
    p_t = float(T.mean())
    diag = {
        "min": float(e_x.min()),
        "p01": float(np.quantile(e_x, 0.01)),
        "p05": float(np.quantile(e_x, 0.05)),
        "p50": float(np.quantile(e_x, 0.50)),
        "p95": float(np.quantile(e_x, 0.95)),
        "p99": float(np.quantile(e_x, 0.99)),
        "max": float(e_x.max()),
        "pct_lt_0_01": float((e_x < 0.01).mean()),
        "pct_gt_0_99": float((e_x > 0.99).mean()),
    }
    logger.info(
        "Propensity ajustada: AUC=%.3f, P(T=1)=%.3f, e(X) ∈ [%.3f, %.3f]; "
        "%.2f%% con e<0.01, %.2f%% con e>0.99.",
        auc, p_t, diag["min"], diag["max"],
        100 * diag["pct_lt_0_01"], 100 * diag["pct_gt_0_99"],
    )
    return ResultadoPropensidad(
        modelo=modelo,
        e_x=e_x,
        auc=auc,
        p_tratamiento=p_t,
        columnas=list(X.columns),
        estadisticas=estadisticas,
        diagnostico_distribucion=diag,
    )

def plot_overlap_propensidad(
    e_x: np.ndarray, T: np.ndarray, ruta_fig: str | Path
) -> None:
    fig, ax = plt.subplots(figsize=(8, 5))
    sns.kdeplot(e_x[T == 1], ax=ax, label="SMS = 1 (tratados)",
                fill=True, alpha=0.4, common_norm=False)
    sns.kdeplot(e_x[T == 0], ax=ax, label="SMS = 0 (controles)",
                fill=True, alpha=0.4, common_norm=False)
    ax.axvline(0.01, ls=":", color="gray", lw=0.8)
    ax.axvline(0.99, ls=":", color="gray", lw=0.8)
    ax.set_xlabel("Propensity score e(X) = P(SMS = 1 | X)")
    ax.set_ylabel("Densidad")
    ax.set_title("Solape del propensity score por brazo (positividad)")
    ax.legend()
    fig.tight_layout()
    fig.savefig(ruta_fig, dpi=DPI_FIGURAS, bbox_inches="tight")
    plt.close(fig)
```

```
    logger.info("Overlap plot guardado en %s.", ruta_fig)

prop = causal.estimar_propensity_score(X, T, stats)
print(f"AUC del propensity (separación tratados/controles): {prop.auc:.4f}")
print("Distribución de e(X):")
for k, v in prop.diagnostico_distribucion.items():
    print(f"  {k}: {v:.4f}")
fig_overlap = rutas.FIGURAS / "nb04_overlap_propensidad_v1.png"
causal.plot_overlap_propensidad(prop.e_x, T, fig_overlap)
display(Image(str(fig_overlap), width=700))


def pesos_estabilizados(e_x: np.ndarray, T: np.ndarray, p_t: float) -> np.ndarray:
    e = np.clip(e_x, 1e-6, 1 - 1e-6)  # evita 0/0 en colas extremas
    return np.where(T == 1, p_t / e, (1 - p_t) / (1 - e))


def resumen_pesos(w: np.ndarray, T: np.ndarray) -> dict[str, Any]:
    out: dict[str, Any] = {}
    for nombre, mask in (("tratados", T == 1), ("controles", T == 0)):
        sub = w[mask]
        out[nombre] = {
            "n": int(mask.sum()),
            "media": float(sub.mean()),
            "sd": float(sub.std(ddof=1)),
            "min": float(sub.min()),
            "p01": float(np.quantile(sub, 0.01)),
            "p50": float(np.quantile(sub, 0.50)),
            "p99": float(np.quantile(sub, 0.99)),
            "max": float(sub.max()),
        }
    return out


def calcular_ess(w: np.ndarray, T: np.ndarray) -> dict[str, Any]:
    out: dict[str, Any] = {}
    for nombre, mask in (("tratados", T == 1), ("controles", T == 0)):
        sub = w[mask]
        n = int(mask.sum())
        ess = float((sub.sum() ** 2) / np.sum(sub ** 2))
        out[nombre] = {
            "n": n,
            "ess": ess,
            "ess_sobre_n": ess / n,
        }
    out["trim_recomendado"] = bool(
        out["tratados"]["ess_sobre_n"] < TRIM_ESS_UMBRAL
        or out["controles"]["ess_sobre_n"] < TRIM_ESS_UMBRAL
    )
    logger.info(
        "ESS tratados=%.0f (%.1f%%); ESS controles=%.0f (%.1f%%); trim=%s.",
        out["tratados"]["ess"], 100 * out["tratados"]["ess_sobre_n"],
        out["controles"]["ess"], 100 * out["controles"]["ess_sobre_n"],
        out["trim_recomendado"],
    )
    return out


def plot_distribucion_pesos(
    w: np.ndarray, T: np.ndarray, ruta_fig: str | Path,
) -> None:
    fig, ax = plt.subplots(figsize=(8, 5))
    bins = np.linspace(np.log10(w).min(), np.log10(w).max(), 60)
    ax.hist(np.log10(w[T == 1]), bins=bins, alpha=0.5,
            label="SMS = 1 (tratados)", color="#1f77b4")
    ax.hist(np.log10(w[T == 0]), bins=bins, alpha=0.5,
            label="SMS = 0 (controles)", color="#d62728")
    ax.set_xlabel("log₁₀(peso estabilizado)")
    ax.set_ylabel("Frecuencia")
    ax.set_title("Distribución de pesos estabilizados por brazo")
    ax.legend()
    fig.tight_layout()
    fig.savefig(ruta_fig, dpi=DPI_FIGURAS, bbox_inches="tight")
    plt.close(fig)
    logger.info("Histograma de pesos guardado en %s.", ruta_fig)

w = causal.pesos_estabilizados(prop.e_x, T, prop.p_tratamiento)
resumen_w = causal.resumen_pesos(w, T)
print("[Pesos estabilizados]")
for brazo, sw in resumen_w.items():
```

```
    print(f"  {brazo}: media={sw['media']:.3f}, p99={sw['p99']:.3f}, max={sw['max']:,.0f}")
ess = causal.calcular_ess(w, T)
print("\n[ESS]")
for brazo in ("tratados", "controles"):
    print(f"  {brazo}: ESS={ess[brazo]['ess']:,.0f} "
          f"({100 * ess[brazo]['ess_sobre_n']:.1f}% de n={ess[brazo]['n']:,})")
print(f"  Regla ESS/n < {causal.TRIM_ESS_UMBRAL} → trim recomendado: {ess['trim_recomendado']}")
fig_pesos = rutas.FIGURAS / "nb04_distribucion_pesos_v1.png"
causal.plot_distribucion_pesos(w, T, fig_pesos)
display(Image(str(fig_pesos), width=700))
```

```
def _smd(x_t: np.ndarray, x_c: np.ndarray, w_t: np.ndarray | None = None,
         w_c: np.ndarray | None = None) -> float:
    if w_t is None:
        mu_t, mu_c = x_t.mean(), x_c.mean()
        v_t = x_t.var(ddof=0)
        v_c = x_c.var(ddof=0)
    else:
        mu_t = np.average(x_t, weights=w_t)
        mu_c = np.average(x_c, weights=w_c)
        v_t = np.average((x_t - mu_t) ** 2, weights=w_t)
        v_c = np.average((x_c - mu_c) ** 2, weights=w_c)
    pooled = np.sqrt((v_t + v_c) / 2)
    if pooled < 1e-12:
        return 0.0
    return float((mu_t - mu_c) / pooled)


def calcular_tabla_smd(
    X: pd.DataFrame, T: np.ndarray, w: np.ndarray,
) -> pd.DataFrame:
    filas = []
    mask_t = T == 1
    mask_c = T == 0
    for col in X.columns:
        x = X[col].to_numpy(dtype=float)
        smd_pre = _smd(x[mask_t], x[mask_c])
        smd_post = _smd(x[mask_t], x[mask_c], w[mask_t], w[mask_c])
        filas.append({
            "covariable": col,
            "etiqueta": ETIQUETAS_COVARIABLES.get(col, col),
            "smd_pre": smd_pre,
            "smd_post": smd_post,
            "abs_smd_pre": abs(smd_pre),
            "abs_smd_post": abs(smd_post),
        })
    tabla = pd.DataFrame(filas).sort_values(
        "abs_smd_post", ascending=False
    ).reset_index(drop=True)
    n_balance = int((tabla["abs_smd_post"] < 0.10).sum())
    logger.info(
        "Balance post-ponderación: %d / %d covariables con |SMD| < 0.10.",
        n_balance, len(tabla),
    )
    return tabla


def plot_love(
    tabla_smd: pd.DataFrame, ruta_fig: str | Path, metodo: str = "IPW",
) -> None:
    tabla = tabla_smd.sort_values("abs_smd_pre", ascending=True).reset_index(drop=True)
    y = np.arange(len(tabla))
    fig, ax = plt.subplots(figsize=(8, max(4, 0.32 * len(tabla))))
    ax.scatter(tabla["abs_smd_pre"], y, label="Sin ponderar",
               marker="o", s=40, color="#d62728")
    ax.scatter(tabla["abs_smd_post"], y, label=f"Ponderado ({metodo})",
               marker="s", s=40, color="#1f77b4")
    for i, fila in tabla.iterrows():
        ax.plot([fila["abs_smd_pre"], fila["abs_smd_post"]], [i, i],
                color="gray", lw=0.6, alpha=0.5)
    ax.axvline(0.10, ls="--", color="black", lw=0.8, label="|SMD| = 0.10")
    ax.set_yticks(y)
    ax.set_yticklabels(tabla["etiqueta"])
    ax.set_xlabel("|SMD|")
    ax.set_title(f"Balance de covariables: antes vs después del {metodo}")
    ax.legend(loc="lower right")
    fig.tight_layout()
    fig.savefig(ruta_fig, dpi=DPI_FIGURAS, bbox_inches="tight")
    plt.close(fig)
    logger.info("Love plot guardado en %s.", ruta_fig)
```

```
tabla_smd = causal.calcular_tabla_smd(X, T, w)
print("Top 5 covariables peor balanceadas tras IPW:")
print(tabla_smd[["etiqueta", "smd_pre", "smd_post"]].head(5).to_string(index=False))
n_bal = int((tabla_smd["abs_smd_post"] < 0.10).sum())
print(f"\nBalance post-IPW: {n_bal}/{len(tabla_smd)} covariables con |SMD| < 0,10; "
      f"max |SMD| = {tabla_smd['abs_smd_post'].max():.2f}")
tabla_smd.to_csv(rutas.REPORTES / "nb04_smd_tabla_v1.csv", index=False)
fig_love = rutas.FIGURAS / "nb04_love_plot_v1.png"
causal.plot_love(tabla_smd, fig_love)
display(Image(str(fig_love), width=700))
```

```
def decidir_especificacion_preferida(ess: dict[str, Any]) -> str:
    if ess["trim_recomendado"]:
        return "trimmed_1_99"
    return "untrimmed"
```

```
def _media_ponderada(x: np.ndarray, w: np.ndarray) -> float:
    return float(np.average(x, weights=w))
```

```
def ate_ponderado(Y: np.ndarray, T: np.ndarray, w: np.ndarray) -> float:
    att = _media_ponderada(Y[T == 1].astype(float), w[T == 1])
    atc = _media_ponderada(Y[T == 0].astype(float), w[T == 0])
    return (att - atc) * 100.0
```

```
def ate_trimmed(
    Y: np.ndarray, T: np.ndarray, w: np.ndarray,
    pct_inferior: float, pct_superior: float,
) -> tuple[float, np.ndarray]:
    lo = np.quantile(w, pct_inferior / 100.0)
    hi = np.quantile(w, pct_superior / 100.0)
    mask = (w >= lo) & (w <= hi)
    ate = ate_ponderado(Y[mask], T[mask], w[mask])
    return ate, mask
```

```
def construir_tabla_ate(
    Y: np.ndarray, T: np.ndarray, w: np.ndarray,
    spec_preferida: str,
) -> pd.DataFrame:
    filas: list[dict[str, Any]] = []
    ate_unt = ate_ponderado(Y, T, w)
    filas.append({
        "especificacion": "untrimmed",
        "pct_inferior": np.nan,
        "pct_superior": np.nan,
        "n_filas_usadas": len(w),
        "ate_pp": ate_unt,
        "es_preferida": spec_preferida == "untrimmed",
    })
    for lo, hi in TRIM_ROBUSTEZ:
        ate_t, mask = ate_trimmed(Y, T, w, lo, hi)
        nombre = f"trimmed_{int(lo)}_{int(hi)}"
        filas.append({
            "especificacion": nombre,
            "pct_inferior": lo,
            "pct_superior": hi,
            "n_filas_usadas": int(mask.sum()),
            "ate_pp": ate_t,
            "es_preferida": spec_preferida == nombre,
        })
    return pd.DataFrame(filas)
```

```
spec_pref = causal.decidir_especificacion_preferida(ess)
tabla_ate = causal.construir_tabla_ate(Y, T, w, spec_pref)
tabla_ate.to_csv(rutas.REPORTES / "nb04_ate_tabla_v1.csv", index=False)
print(tabla_ate.to_string(index=False))
sensibilidad_trim = tabla_ate["ate_pp"].max() - tabla_ate["ate_pp"].min()
print(f"\nSpec preferida por la regla: {spec_pref}")
print(f"Sensibilidad al trimming (max – min): {sensibilidad_trim:.2f} pp")
```

```
def pesos_overlap_ato(e_x: np.ndarray, T: np.ndarray) -> np.ndarray:
    return np.where(T == 1, 1 - e_x, e_x)
```

```
w_ato = causal.pesos_overlap_ato(prop.e_x, T)
ess_ato = causal.calcular_ess(w_ato, T)
```

```
print("[ESS bajo ATO]")
for brazo in ("tratados", "controles"):
    print(f"  {brazo}: {100 * ess_ato[brazo]['ess_sobre_n']:.1f}% de n={ess_ato[brazo]['n']:,}")
print(f"Peso máximo: IPW = {w.max():,.0f}  →  ATO = {w_ato.max():.4f} (acotado en [0, 1])")
tabla_smd_ato = causal.calcular_tabla_smd(X, T, w_ato)
n_bal_ato = int((tabla_smd_ato["abs_smd_post"] < 0.10).sum())
print(f"\nBalance post-ATO: {n_bal_ato}/{len(tabla_smd_ato)} covariables con |SMD| < 0,10; "
      f"max |SMD| = {tabla_smd_ato['abs_smd_post'].max():.4f}")
tabla_smd_ato.to_csv(rutas.REPORTES / "nb04_smd_ato_tabla_v1.csv", index=False)
fig_love_ato = rutas.FIGURAS / "nb04_love_plot_ato_v1.png"
causal.plot_love(tabla_smd_ato, fig_love_ato, metodo="ATO")
display(Image(str(fig_love_ato), width=700))


def ate_ato(Y: np.ndarray, T: np.ndarray, e_x: np.ndarray) -> float:
    w = pesos_overlap_ato(e_x, T)
    return ate_ponderado(Y, T, w)


fig, ax = plt.subplots(figsize=(8, 5))
bins = np.linspace(0, 1, 60)
ax.hist(w_ato[T == 1], bins=bins, alpha=0.5, label="SMS = 1 (tratados)", color="#1f77b4")
ax.hist(w_ato[T == 0], bins=bins, alpha=0.5, label="SMS = 0 (controles)", color="#d62728")
ax.set_xlabel("Peso ATO  (tratados: 1 − e(X); controles: e(X))")
ax.set_ylabel("Frecuencia")
ax.set_title("Distribución de pesos overlap (ATO) por brazo")
ax.legend()
fig.tight_layout()
fig.savefig(rutas.FIGURAS / "nb04_distribucion_pesos_ato_v1.png",
            dpi=causal.DPI_FIGURAS, bbox_inches="tight")
plt.show()
ato_punto = causal.ate_ato(Y, T, prop.e_x)
print(f"ATO puntual: {ato_punto:+.3f} pp  (positivo = SMS asociado con mayor asistencia)")


def _construir_indices_por_paciente(patient_ids: pd.Series) -> dict[Any, np.ndarray]:
    return {k: np.asarray(v) for k, v in patient_ids.groupby(patient_ids).indices.items()}


def cluster_bootstrap_ato(
    X: pd.DataFrame,
    T: np.ndarray,
    Y: np.ndarray,
    patient_ids: pd.Series,
    *,
    n_iter: int = N_BOOTSTRAP,
    seed: int = SEED_BOOTSTRAP,
    C: float = LOG_REG_C,
    max_iter: int = LOG_REG_MAX_ITER,
    solver: str = LOG_REG_SOLVER,
    verbose_cada: int = 100,
) -> np.ndarray:
    rng = np.random.default_rng(seed)
    pid_to_idx = _construir_indices_por_paciente(patient_ids)
    unique_pids = np.fromiter(pid_to_idx.keys(), dtype=patient_ids.dtype)
    X_np = X.to_numpy(dtype=float)
    atos = np.empty(n_iter, dtype=float)
    logger.info(
        "Cluster bootstrap ATO: %d iteraciones, %d pacientes únicos.",
        n_iter, len(unique_pids),
    )
    for i in range(n_iter):
        sampled = rng.choice(unique_pids, size=len(unique_pids), replace=True)
        idx = np.concatenate([pid_to_idx[pid] for pid in sampled])
        X_b = X_np[idx]
        T_b = T[idx]
        Y_b = Y[idx]
        modelo = LogisticRegression(C=C, max_iter=max_iter, solver=solver)
        modelo.fit(X_b, T_b)
        e_b = modelo.predict_proba(X_b)[:, 1]
        w_b = pesos_overlap_ato(e_b, T_b)
        atos[i] = ate_ponderado(Y_b, T_b, w_b)
        if verbose_cada and (i + 1) % verbose_cada == 0:
            logger.info(
                "  bootstrap %d/%d — ATO acumulado: media=%.3f pp, "
                "IC95%%≈[%.3f, %.3f] pp",
                i + 1, n_iter, atos[:i + 1].mean(),
                np.quantile(atos[:i + 1], 0.025),
                np.quantile(atos[:i + 1], 0.975),
            )
    return atos
```

```
def persistir_bootstrap(ates: np.ndarray, ruta: str | Path) -> None:
    np.save(ruta, ates)
    logger.info("Bootstrap persistido en %s (%d valores).", ruta, len(ates))

def resumen_bootstrap(ates: np.ndarray) -> dict[str, float]:
    return {
        "n_iter": int(len(ates)),
        "media": float(ates.mean()),
        "mediana": float(np.median(ates)),
        "sd": float(ates.std(ddof=1)),
        "ic95_inf": float(np.quantile(ates, 0.025)),
        "ic95_sup": float(np.quantile(ates, 0.975)),
        "min": float(ates.min()),
        "max": float(ates.max()),
    }

def plot_distribucion_ate_bootstrap(
    ates: np.ndarray, ate_punto: float, ruta_fig: str | Path,
    spec: str = "preferida",
) -> None:
    ic_inf = float(np.quantile(ates, 0.025))
    ic_sup = float(np.quantile(ates, 0.975))
    fig, ax = plt.subplots(figsize=(8, 5))
    ax.hist(ates, bins=40, color="#1f77b4", alpha=0.75, edgecolor="white")
    ax.axvline(ate_punto, color="black", lw=2,
               label=f"ATE puntual = {ate_punto:.2f} pp")
    ax.axvline(ic_inf, color="red", ls="--",
               label=f"IC95% inf = {ic_inf:.2f}")
    ax.axvline(ic_sup, color="red", ls="--",
               label=f"IC95% sup = {ic_sup:.2f}")
    ax.axvline(0, color="gray", lw=0.8)
    ax.set_xlabel("ATE en puntos porcentuales (asistencia)")
    ax.set_ylabel("Frecuencia (sobre 1000 bootstraps)")
    ax.set_title(
        f"Distribución bootstrap del ATE (cluster por PatientId, spec: {spec})"
    )
    ax.legend()
    fig.tight_layout()
    fig.savefig(ruta_fig, dpi=DPI_FIGURAS, bbox_inches="tight")
    plt.close(fig)
    logger.info("Histograma del bootstrap guardado en %s.", ruta_fig)

ruta_boot = rutas.BOOTSTRAP / "nb04_bootstrap_ato_v1.npy"
if REGENERAR_BOOTSTRAP or not ruta_boot.exists():
    atos = causal.cluster_bootstrap_ato(X=X, T=T, Y=Y,
                                        patient_ids=df[causal.COL_CLUSTER],
                                        n_iter=causal.N_BOOTSTRAP,
                                        seed=causal.SEED_BOOTSTRAP)
    causal.persistir_bootstrap(atos, ruta_boot)
else:
    atos = np.load(ruta_boot)
    print(f"Bootstrap cargado de {ruta_boot.name} ({len(atos)} iteraciones).")
res_boot = causal.resumen_bootstrap(atos)
print(f"ATO bootstrap: media = {res_boot['media']:+.3f} pp; "
      f"IC95% = [{res_boot['ic95_inf']:+.3f}, {res_boot['ic95_sup']:+.3f}] pp; "
      f"sd = {res_boot['sd']:.3f}")
print(f"IC95% cruza cero: {res_boot['ic95_inf'] < 0 < res_boot['ic95_sup']}")
fig_boot = rutas.FIGURAS / "nb04_bootstrap_ato_distribucion_v1.png"
causal.plot_distribucion_ate_bootstrap(atos, ato_punto, fig_boot, spec="ATO (Li et al. 2018)")
display(Image(str(fig_boot), width=700))

X_cat, T_cat, Y_cat, stats_cat = causal.construir_matriz_propensidad(
    df, lead_time_spec="categorical")
prop_cat = causal.estimar_propensity_score(X_cat, T_cat, stats_cat)
print(f"AUC propensity (spec categórica): {prop_cat.auc:.4f}")
w_cat = causal.pesos_overlap_ato(prop_cat.e_x, T_cat)
tabla_smd_cat = causal.calcular_tabla_smd(X_cat, T_cat, w_cat)
print(f"Balance post-ATO (categórica): "
      f"{int((tabla_smd_cat['abs_smd_post'] < 0.10).sum())}/{len(tabla_smd_cat)} "
      f"con |SMD| < 0,10; max |SMD| = {tabla_smd_cat['abs_smd_post'].max():.4f}")
ato_cat = causal.ate_ato(Y_cat, T_cat, prop_cat.e_x)
print(f"ATO (spec categórica): {ato_cat:+.3f} pp")
ruta_boot_cat = rutas.BOOTSTRAP / "nb04_bootstrap_ato_leadcat_v1.npy"
if REGENERAR_BOOTSTRAP or not ruta_boot_cat.exists():
    atos_cat = causal.cluster_bootstrap_ato(X=X_cat, T=T_cat, Y=Y_cat,
                                            patient_ids=df[causal.COL_CLUSTER],
```

```
                                            n_iter=causal.N_BOOTSTRAP,
                                            seed=causal.SEED_BOOTSTRAP)
    np.save(ruta_boot_cat, atos_cat)
else:
    atos_cat = np.load(ruta_boot_cat)
    print(f"Bootstrap categórico cargado ({len(atos_cat)} iteraciones).")
res_cat = causal.resumen_bootstrap(atos_cat)
print(f"IC95% (categórica): [{res_cat['ic95_inf']:+.3f}, {res_cat['ic95_sup']:+.3f}] pp")


def ic95_se_solapan(a_lo, a_hi, b_lo, b_hi):
    return max(a_lo, b_lo) <= min(a_hi, b_hi)
misma_direccion = (res_boot["media"] * res_cat["media"]) > 0
solapan = ic95_se_solapan(res_boot["ic95_inf"], res_boot["ic95_sup"],
                          res_cat["ic95_inf"], res_cat["ic95_sup"])
robusto = misma_direccion and solapan
print(f"Misma dirección: {misma_direccion}  |  IC95% se solapan: {solapan}")
print(f"→ Regla pre-especificada de robustez: {'ROBUSTO' if robusto else 'NO ROBUSTO'}")
fig, ax = plt.subplots(figsize=(9, 5))
bins_hist = np.linspace(min(atos.min(), atos_cat.min()) - 0.2,
                        max(atos.max(), atos_cat.max()) + 0.2, 50)
ax.hist(atos, bins=bins_hist, alpha=0.5, color="#1f77b4", edgecolor="white",
        label=f"z+z² (baseline): IC95% [{res_boot['ic95_inf']:.2f}, {res_boot['ic95_sup']:.2f}]")
ax.hist(atos_cat, bins=bins_hist, alpha=0.5, color="#ff7f0e", edgecolor="white",
        label=f"categórica: IC95% [{res_cat['ic95_inf']:.2f}, {res_cat['ic95_sup']:.2f}]")
ax.axvline(0, color="gray", lw=0.8)
ax.axvline(ato_punto, color="#1f77b4", ls="--", lw=1.5,
           label=f"ATO baseline = {ato_punto:.2f}")
ax.axvline(ato_cat, color="#ff7f0e", ls="--", lw=1.5,
           label=f"ATO categórica = {ato_cat:.2f}")
ax.set_xlabel("ATO (puntos porcentuales, outcome = Showed_up)")
ax.set_ylabel("Frecuencia (bootstrap)")
ax.set_title("Sensibilidad del ATO a la especificación de lead_time")
ax.legend(loc="upper left", fontsize=8)
fig.tight_layout()
fig.savefig(rutas.FIGURAS / "nb04_sensibilidad_lead_v1.png",
            dpi=causal.DPI_FIGURAS, bbox_inches="tight")
plt.show()


def ate_pp_a_rr(
    ate_pp: float, Y: np.ndarray, T: np.ndarray, w: np.ndarray,
) -> tuple[float, float, float]:
    att_T = _media_ponderada(Y[T == 1].astype(float), w[T == 1])
    att_C = _media_ponderada(Y[T == 0].astype(float), w[T == 0])
    rr = att_T / att_C if att_C > 0 else float("nan")
    return float(rr), float(att_T), float(att_C)


def evalue(rr: float) -> float:
    if rr <= 0 or not np.isfinite(rr):
        return float("nan")
    if rr == 1:
        return 1.0
    rr_eff = rr if rr > 1 else 1 / rr
    return float(rr_eff + np.sqrt(rr_eff * (rr_eff - 1)))


rr, asis_T, asis_C = causal.ate_pp_a_rr(ato_punto, Y, T, w_ato)
ev = causal.evalue(rr)
print(f"Asistencia ponderada (ATO): SMS=1 → {100 * asis_T:.2f}%  |  SMS=0 → {100 * asis_C:.2f}%")
print(f"RR aproximado = {rr:.4f}  →  E-value = {ev:.3f}")
print("Lectura: bastaría una confusión no observada de magnitud modesta para explicar "
      "el efecto baseline — coherente con la fragilidad documentada en §8.")


def persistir_propensidad(
    resultado: ResultadoPropensidad, ruta: str | Path,
) -> None:
    payload = {
        "modelo": resultado.modelo,
        "columnas": resultado.columnas,
        "estadisticas_medias": resultado.estadisticas.medias,
        "estadisticas_desviaciones": resultado.estadisticas.desviaciones,
        "p_tratamiento": resultado.p_tratamiento,
        "auc": resultado.auc,
    }
    with open(ruta, "wb") as f:
        pickle.dump(payload, f)
    logger.info("Propensidad persistida en %s.", ruta)
```

```
def persistir_sidecar(metadata: dict[str, Any], ruta: str | Path) -> None:
    metadata = {**metadata, "guardado_en": datetime.now(timezone.utc).isoformat()}
    with open(ruta, "w", encoding="utf-8") as f:
        json.dump(metadata, f, indent=2, ensure_ascii=False, default=str)
    logger.info("Sidecar JSON guardado en %s.", ruta)


causal.persistir_propensidad(prop, rutas.MODELOS / "propensidad_v1.pkl")
pesos_df = pd.DataFrame({
    "PatientId": df[causal.COL_CLUSTER].to_numpy(),
    "AppointmentID": df["AppointmentID"].to_numpy(),
    "SMS_received": T,
    "Showed_up": df[causal.COL_OUTCOME].to_numpy(),
    "propensity_score": prop.e_x,
    "peso_estabilizado": w,
    "peso_ato": w_ato,
})
ruta_pesos = rutas.REPORTES / "nb04_pesos_propension_v1.csv"
pesos_df.to_csv(ruta_pesos, index=False)
print(f"Pesos por fila → {ruta_pesos.name} ({len(pesos_df):,} filas)  [entrada de NB05]")
sidecar = {
    "notebook": "04_inferencia_causal_ipw",
    "estimand_resultante": "ATO (Li, Morgan & Zaslavsky 2018) — contingencia pre-especificada",
    "propensity_auc": float(prop.auc),
    "ess_ipw_sobre_n": {b: float(ess[b]["ess_sobre_n"]) for b in ("tratados", "controles")},
    "trim_disparado": bool(ess["trim_recomendado"]),
    "tabla_ate_ipw": tabla_ate.to_dict(orient="records"),
    "ato_punto_pp": float(ato_punto),
    "ato_ic95": [float(res_boot["ic95_inf"]), float(res_boot["ic95_sup"])],
    "ato_categorico_pp": float(ato_cat),
    "ato_categorico_ic95": [float(res_cat["ic95_inf"]), float(res_cat["ic95_sup"])],
    "robusto_a_spec_lead_time": bool(robusto),
    "e_value_ato_baseline": float(ev),
    "decision_nb06": ("El Monte Carlo de NB06 no consume este bootstrap: el efecto se "
                      "parametriza con escenarios de reducción relativa del no-show de la "
                      "literatura experimental publicada. El ATO se reporta como evidencia "
                      "de no-identificabilidad observacional."),
}
causal.persistir_sidecar(sidecar, rutas.REPORTES / "nb04_metadatos_v1.json")

assert causal.COL_TRATAMIENTO not in X.columns, "el tratamiento no puede ser covariable"
assert "neighbourhood_encoded" in X.columns, "barrio debe ir con frequency encoding"
assert len(df) > 100_000, "el análisis causal usa el dataset completo, no el train split"
assert not robusto, "si esto falla, la narrativa de §8 y la memoria deben revisarse"
print("Checklist de reglas críticas: OK")
print("NB04 completo.")
```

B.5 Segmentación de pacientes

src/segmentacion.py · notebooks/05_segmentacion_pacientes.ipynb

```
from __future__ import annotations
import json
import logging
from pathlib import Path
from typing import Any
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler
logger = logging.getLogger(__name__)
SEED: int = 42
DPI_FIGURAS: int = 200
LEAD_BIN_ORDINAL: dict[str, int] = {"mismo_dia": 0, "1-7d": 1, "8-14d": 2, "15+d": 3}
FEATURES_CLUSTERING: list[str] = [
    "prob_noshow_calibrada",
    "neighbourhood_encoded",
    "lead_bin_ord",
    "is_first_visit",
]
ETIQUETAS_FEATURES: dict[str, str] = {
    "prob_noshow_calibrada": "prob. no-show calibrada",
    "neighbourhood_encoded": "barrio (frecuencia)",
    "lead_bin_ord": "antelación (bin ordinal)",
    "is_first_visit": "primera visita observada",
```

```
}

def configurar_estilo() -> None:
    sns.set_theme(context="notebook", style="whitegrid", palette="deep")
    plt.rcParams.update({
        "figure.dpi": 100, "savefig.dpi": DPI_FIGURAS,
        "axes.titlesize": 12, "axes.labelsize": 10, "legend.fontsize": 9,
        "xtick.labelsize": 9, "ytick.labelsize": 9,
    })

from __future__ import annotations
import sys
from pathlib import Path
import numpy as np
import pandas as pd
from IPython.display import Image, display
RAIZ = Path.cwd().parent if Path.cwd().name == "notebooks" else Path.cwd()
sys.path.insert(0, str(RAIZ))
import logging
logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s",
                    datefmt="%H:%M:%S")
seg.configurar_estilo()
np.random.seed(seg.SEED)


def cargar_datos_segmentacion(
    ruta_probs: str | Path, ruta_full_clean: str | Path,
) -> pd.DataFrame:
    probs = pd.read_csv(ruta_probs, usecols=[
        "PatientId", "AppointmentID", "Showed_up", "prob_noshow_calibrada"])
    cols_fc = ["AppointmentID", "SMS_received", "lead_time_bin", "is_first_visit",
               "neighbourhood_encoded", "Age", "lead_time", "prior_noshow_rate",
               "comorbidity_count", "chronic_flag", "appointment_weekday"]
    fc = pd.read_csv(ruta_full_clean, usecols=cols_fc)
    df = probs.merge(fc, on="AppointmentID", how="left")
    df["lead_bin_ord"] = df["lead_time_bin"].map(LEAD_BIN_ORDINAL)
    if df["lead_bin_ord"].isna().any():
        raise ValueError("lead_time_bin con valores fuera del mapa ordinal.")
    logger.info("Datos de segmentación: %d citas (test), %d pacientes únicos.",
                len(df), df["PatientId"].nunique())
    return df

df = seg.cargar_datos_segmentacion(
    rutas.REPORTES / "noshow_probs_v1.csv",
    rutas.CSV_FULL_CLEAN,
)
print(f"Citas: {len(df):,}  |  pacientes únicos: {df['PatientId'].nunique():,}")
df[["prob_noshow_calibrada", "lead_time_bin", "is_first_visit",
    "neighbourhood_encoded", "SMS_received", "Showed_up"]].head()

def construir_matriz_clustering(
    df: pd.DataFrame,
) -> tuple[np.ndarray, StandardScaler]:
    X = df[FEATURES_CLUSTERING].to_numpy(dtype=float)
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
    return X_scaled, scaler


X_scaled, scaler = seg.construir_matriz_clustering(df)
print("Variables de clustering (estandarizadas):")
for f in seg.FEATURES_CLUSTERING:
    print(f"  · {seg.ETIQUETAS_FEATURES[f]}")
print(f"Matriz: {X_scaled.shape[0]:,} citas × {X_scaled.shape[1]} variables")

def evaluar_k(
    X_scaled: np.ndarray, k_range: range = range(2, 9), seed: int = SEED,
) -> pd.DataFrame:
    filas = []
    for k in k_range:
        km = KMeans(n_clusters=k, random_state=seed, n_init=10)
        labels = km.fit_predict(X_scaled)
        sil = silhouette_score(X_scaled, labels, sample_size=5000, random_state=seed)
        filas.append({"k": k, "inercia": float(km.inertia_), "silueta": float(sil)})
        logger.info("k=%d → inercia=%.0f, silueta=%.4f", k, km.inertia_, sil)
    return pd.DataFrame(filas)
```

```
def plot_seleccion_k(tabla: pd.DataFrame, ruta_fig: str | Path) -> None:
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(11, 4.2))
    ax1.plot(tabla["k"], tabla["inercia"], "o-", color="#1f77b4")
    ax1.set_xlabel("número de clústeres (k)")
    ax1.set_ylabel("inercia (suma de cuadrados intra-clúster)")
    ax1.set_title("Curva de codo")
    ax2.plot(tabla["k"], tabla["silueta"], "o-", color="#d62728")
    ax2.set_xlabel("número de clústeres (k)")
    ax2.set_ylabel("coeficiente de silueta")
    ax2.set_title("Silueta media frente a k")
    fig.tight_layout()
    fig.savefig(ruta_fig, dpi=DPI_FIGURAS, bbox_inches="tight")
    plt.close(fig)
    logger.info("Selección de k guardada en %s.", ruta_fig)

tabla_k = seg.evaluar_k(X_scaled)
print(tabla_k.to_string(index=False))
fig_k = rutas.FIGURAS / "nb05_seleccion_k_v1.png"
seg.plot_seleccion_k(tabla_k, fig_k)
display(Image(str(fig_k), width=850))

def ajustar_kmeans(
    X_scaled: np.ndarray, k: int, seed: int = SEED,
) -> tuple[np.ndarray, KMeans]:
    km = KMeans(n_clusters=k, random_state=seed, n_init=10)
    labels = km.fit_predict(X_scaled)
    return labels, km

def plot_pca(X_scaled: np.ndarray, labels: np.ndarray, ruta_fig: str | Path) -> None:
    pca = PCA(n_components=2, random_state=SEED)
    coords = pca.fit_transform(X_scaled)
    var = pca.explained_variance_ratio_
    fig, ax = plt.subplots(figsize=(7.5, 6))
    for c in sorted(np.unique(labels)):
        m = labels == c
        ax.scatter(coords[m, 0], coords[m, 1], s=6, alpha=0.35, label=f"Clúster {c}")
    ax.set_xlabel(f"PC1 ({100*var[0]:.0f}% var.)")
    ax.set_ylabel(f"PC2 ({100*var[1]:.0f}% var.)")
    ax.set_title("Proyección PCA de los clústeres de pacientes")
    ax.legend(markerscale=2)
    fig.tight_layout()
    fig.savefig(ruta_fig, dpi=DPI_FIGURAS, bbox_inches="tight")
    plt.close(fig)
    logger.info("PCA guardado en %s.", ruta_fig)

K = 3
labels, modelo_km = seg.ajustar_kmeans(X_scaled, K)
df["cluster"] = labels
print(f"K-means con k={K}. Tamaños de clúster:")
print(pd.Series(labels).value_counts().sort_index().to_string())
fig_pca = rutas.FIGURAS / "nb05_pca_clusters_v1.png"
seg.plot_pca(X_scaled, labels, fig_pca)
display(Image(str(fig_pca), width=650))

def perfilar_clusters(df: pd.DataFrame, labels: np.ndarray) -> pd.DataFrame:
    d = df.copy()
    d["cluster"] = labels
    perfil = d.groupby("cluster").agg(
        n=("AppointmentID", "size"),
        prob_noshow_media=("prob_noshow_calibrada", "mean"),
        noshow_observado=("Showed_up", lambda s: float((s == 0).mean())),
        lead_time_medio=("lead_time", "mean"),
        edad_media=("Age", "mean"),
        pct_primera_visita=("is_first_visit", "mean"),
        prior_noshow_medio=("prior_noshow_rate", "mean"),
        comorbilidades_media=("comorbidity_count", "mean"),
        pct_sms=("SMS_received", "mean"),
    ).reset_index()
    return perfil.round(4)

perfil = seg.perfilar_clusters(df, labels)
perfil["noshows_esperados"] = (perfil["n"] * perfil["noshow_observado"]).round(0).astype(int)
display(perfil)
ruta_perfil = rutas.REPORTES / "nb05_perfil_clusters_v1.csv"
perfil.to_csv(ruta_perfil, index=False)
print(f"Perfil guardado en {ruta_perfil.name}")
```

```
def etiquetar(fila):
    if fila["prob_noshow_media"] < 0.10:
        return "Bajo riesgo · mismo día"
    if fila["pct_primera_visita"] > 0.5:
        return "Alto riesgo · primera visita · alta antelación"
    return "Alto riesgo · recurrente · antelación media"
perfil["etiqueta"] = perfil.apply(etiquetar, axis=1)
assert perfil["etiqueta"].nunique() == len(perfil), \
    "etiquetas duplicadas: revisar la regla de etiquetado frente al perfil"
mapa_etiquetas = dict(zip(perfil["cluster"], perfil["etiqueta"]))
for _, f in perfil.iterrows():
    print(f"Clúster {f['cluster']} — {f['etiqueta']}")
    print(f"    n={f['n']:,}  prob. no-show={f['prob_noshow_media']:.1%}  "
          f"no-show obs.={f['noshow_observado']:.1%}  antelación={f['lead_time_medio']:.1f} d  "
          f"primera visita={f['pct_primera_visita']:.0%}  SMS={f['pct_sms']:.0%}")
    print(f"    inasistencias esperadas/periodo ≈ {f['noshows_esperados']:,}")


def resumen_sms_por_cluster(df: pd.DataFrame, labels: np.ndarray) -> pd.DataFrame:
    d = df.copy()
    d["cluster"] = labels
    d["noshow"] = (d["Showed_up"] == 0).astype(int)
    filas = []
    for c, sub in d.groupby("cluster"):
        s1, s0 = sub[sub["SMS_received"] == 1], sub[sub["SMS_received"] == 0]
        filas.append({
            "cluster": int(c),
            "n_sms": len(s1), "n_no_sms": len(s0),
            "noshow_sms": float(s1["noshow"].mean()) if len(s1) else np.nan,
            "noshow_no_sms": float(s0["noshow"].mean()) if len(s0) else np.nan,
        })
    out = pd.DataFrame(filas)
    out["dif_observada_pp"] = (out["noshow_sms"] - out["noshow_no_sms"]) * 100
    return out.round(4)


resumen_sms = seg.resumen_sms_por_cluster(df, labels)
resumen_sms["etiqueta"] = resumen_sms["cluster"].map(mapa_etiquetas)
display(resumen_sms)


total_noshows = int((df["Showed_up"] == 0).sum())
alto_riesgo = perfil[perfil["prob_noshow_media"] >= 0.10]
share_volumen = alto_riesgo["noshows_esperados"].sum() / total_noshows
share_agenda = alto_riesgo["n"].sum() / len(df)
print(f"Los clústeres de alto riesgo son el {share_agenda:.0%} de la agenda "
      f"y concentran el {share_volumen:.0%} de las inasistencias.")


def persistir_segmentacion(
    df: pd.DataFrame, labels: np.ndarray, ruta: str | Path,
) -> None:
    out = df[["PatientId", "AppointmentID", "prob_noshow_calibrada",
              "SMS_received", "Showed_up"]].copy()
    out["cluster"] = labels
    out.to_csv(ruta, index=False)
    logger.info("Asignación de clústeres guardada en %s (%d filas).", ruta, len(out))


def persistir_sidecar(metadata: dict[str, Any], ruta: str | Path) -> None:
    with open(ruta, "w", encoding="utf-8") as f:
        json.dump(metadata, f, indent=2, ensure_ascii=False, default=str)
    logger.info("Sidecar guardado en %s.", ruta)


ruta_asig = rutas.REPORTES / "nb05_clusters_v1.csv"
seg.persistir_segmentacion(df, labels, ruta_asig)
sidecar = {
    "notebook": "05_segmentacion_pacientes",
    "n_citas": int(len(df)),
    "conjunto": "test (con probabilidad calibrada de NB03)",
    "k_elegido": int(K),
    "criterio_k": "codo + interpretabilidad (silueta monótona, sin máximo)",
    "silueta_k": float(tabla_k.loc[tabla_k['k'] == K, 'silueta'].iloc[0]),
    "features_clustering": seg.FEATURES_CLUSTERING,
    "perfil": perfil.to_dict(orient="records"),
    "resumen_sms_por_cluster": resumen_sms.to_dict(orient="records"),
    "ate_por_cluster": ("No estimado como cantidad causal: la positividad rota a nivel de "
                        "población (NB04) no se resuelve en subgrupos. Se reporta riesgo y "
                        "potencial observados."),
    "share_inasistencias_alto_riesgo": float(share_volumen),
```

```
}
seg.persistir_sidecar(sidecar, rutas.REPORTES / "nb05_metadatos_v1.json")
print("Sidecar guardado.")


assert df["cluster"].nunique() == K, "número de clústeres inesperado"
assert not df[seg.FEATURES_CLUSTERING].isna().any().any(), "NaN en variables de clustering"
assert "SMS_received" not in seg.FEATURES_CLUSTERING, "el tratamiento no debe definir clústeres"
print("Checklist NB05: OK")
print("NB05 completo.")
```

B.6 Modelo de pricing por simulación de Monte Carlo

src/pricing.py · notebooks/06_pricing_montecarlo.ipynb

```
from __future__ import annotations
import json
import logging
from dataclasses import dataclass, asdict
from pathlib import Path
from typing import Any
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
logger = logging.getLogger(__name__)
N_ITER: int = 10_000
SEED_MONTECARLO: int = 2026
DPI_FIGURAS: int = 200


from __future__ import annotations
import sys
from pathlib import Path
import numpy as np
import pandas as pd
from IPython.display import Image, display
RAIZ = Path.cwd().parent if Path.cwd().name == "notebooks" else Path.cwd()
sys.path.insert(0, str(RAIZ))
import logging
logging.basicConfig(level=logging.INFO, format="%(asctime)s [%(levelname)s] %(message)s",
                    datefmt="%H:%M:%S")


@dataclass
class ParametrosMonteCarlo:
    reduccion_min: float = 0.18      # IC sup. RR no-show Robotham 2016 (0,82)
    reduccion_moda: float = 0.25     # Robotham 2016, "25% less likely to no-show"
    reduccion_max: float = 0.335     # Cochrane 2013, 32,2% → 21,4% entre brazos
    basal_min: float = 0.10
    basal_moda: float = 0.125
    basal_max: float = 0.20
    valor_min: float = 60.0
    valor_moda: float = 100.0
    valor_max: float = 160.0
    sms_min: float = 0.034
    sms_moda: float = 0.045
    sms_max: float = 0.096
    volumen_min: float = 200.0
    volumen_max: float = 800.0
    margen_min: float = 0.60
    margen_moda: float = 0.75
    margen_max: float = 0.90
    share_targeted: float = 1.0      # 1.0 = SMS uniforme a toda la agenda
    fixed_pct: float = 0.20          # fijo = 20% del percentil de valor neto
    operational_floor: float = 50.0  # mínimo mensual (comunicación + soporte)
    variable_conservador: float = 0.10
    variable_recomendado: float = 0.15
    variable_premium: float = 0.20


params = pricing.ParametrosMonteCarlo()
print("Parámetros del Monte Carlo (escenario SMS uniforme, share_targeted=1.0):")
for k, v in vars(params).items():
    print(f"  {k}: {v}")


def simular(
    params: ParametrosMonteCarlo,
    n_iter: int = N_ITER,
    seed: int = SEED_MONTECARLO,
) -> pd.DataFrame:
```

```
    rng = np.random.default_rng(seed)
    reduccion = rng.triangular(params.reduccion_min, params.reduccion_moda,
                               params.reduccion_max, n_iter)
    basal = rng.triangular(params.basal_min, params.basal_moda,
                           params.basal_max, n_iter)
    valor = rng.triangular(params.valor_min, params.valor_moda,
                           params.valor_max, n_iter)
    sms = rng.triangular(params.sms_min, params.sms_moda, params.sms_max, n_iter)
    volumen = rng.uniform(params.volumen_min, params.volumen_max, n_iter)
    margen = rng.triangular(params.margen_min, params.margen_moda,
                            params.margen_max, n_iter)
    tratadas = volumen * params.share_targeted
    recuperadas = tratadas * basal * reduccion
    valor_bruto = recuperadas * valor * margen
    coste_comunicacion = tratadas * sms
    valor_neto = valor_bruto - coste_comunicacion
    logger.info(
        "Monte Carlo: %d iteraciones. Valor neto mensual P50=%.0f€ "
        "(P10=%.0f€, P90=%.0f€).",
        n_iter, np.median(valor_neto),
        np.quantile(valor_neto, 0.10), np.quantile(valor_neto, 0.90),
    )
    return pd.DataFrame({
        "reduccion_relativa": reduccion,
        "tasa_basal": basal,
        "valor_consulta": valor,
        "coste_sms": sms,
        "volumen_mensual": volumen,
        "margen": margen,
        "citas_recuperadas": recuperadas,
        "valor_bruto": valor_bruto,
        "coste_comunicacion": coste_comunicacion,
        "valor_neto": valor_neto,
    })

sim = pricing.simular(params)
print(f"Iteraciones: {len(sim):,}")
sim[["citas_recuperadas", "valor_bruto", "coste_comunicacion", "valor_neto"]].describe().round(1)

def resumen_percentiles(valor_neto: np.ndarray) -> dict[str, float]:
    return {
        "p10": float(np.quantile(valor_neto, 0.10)),
        "p25": float(np.quantile(valor_neto, 0.25)),
        "p50": float(np.quantile(valor_neto, 0.50)),
        "p75": float(np.quantile(valor_neto, 0.75)),
        "p90": float(np.quantile(valor_neto, 0.90)),
        "media": float(np.mean(valor_neto)),
        "pct_negativo": float((valor_neto < 0).mean()),
    }

def plot_distribucion_valor(
    valor_neto: np.ndarray, percentiles: dict[str, float], ruta_fig: str | Path,
) -> None:
    fig, ax = plt.subplots(figsize=(9, 5))
    ax.hist(valor_neto, bins=60, color="#1f77b4", alpha=0.75, edgecolor="white")
    for etiqueta, clave, color in (
        ("P10", "p10", "#d62728"), ("P50 (mediana)", "p50", "black"),
        ("P90", "p90", "#2ca02c"),
    ):
        ax.axvline(percentiles[clave], color=color, ls="--", lw=1.6,
                   label=f"{etiqueta} = {percentiles[clave]:,.0f} €")
    ax.axvline(0, color="gray", lw=0.8)
    ax.set_xlabel("Valor neto mensual recuperado (€)")
    ax.set_ylabel("Frecuencia (10.000 iteraciones)")
    ax.set_title("Distribución simulada del valor mensual recuperado por clínica")
    ax.legend()
    fig.tight_layout()
    fig.savefig(ruta_fig, dpi=DPI_FIGURAS, bbox_inches="tight")
    plt.close(fig)
    logger.info("Histograma de valor guardado en %s.", ruta_fig)

pct = pricing.resumen_percentiles(sim["valor_neto"].to_numpy())
print("Valor neto mensual recuperado por clínica (€):")
for k, v in pct.items():
    print(f"  {k}: {v:,.1f}" + (" €" if k != "pct_negativo" else "  (fracción de iteraciones <
0)"))
fig_valor = rutas.FIGURAS / "nb06_distribucion_valor_v1.png"
```

```
pricing.plot_distribucion_valor(sim["valor_neto"].to_numpy(), pct, fig_valor)
display(Image(str(fig_valor), width=750))
```

```
def derivar_tarifas(
    percentiles: dict[str, float], params: ParametrosMonteCarlo,
) -> pd.DataFrame:
    filas = [
        ("Conservador", percentiles["p10"], params.variable_conservador,
         max(params.fixed_pct * percentiles["p10"], params.operational_floor)),
        ("Recomendado", percentiles["p50"], params.variable_recomendado,
         params.fixed_pct * percentiles["p50"]),
        ("Premium", percentiles["p90"], params.variable_premium,
         params.fixed_pct * percentiles["p90"]),
    ]
    return pd.DataFrame([
        {
            "plan": nombre,
            "percentil_valor_neto_eur": round(percentil, 2),
            "cuota_fija_eur_mes": round(fijo, 2),
            "componente_variable_pct": int(var * 100),
        }
        for nombre, percentil, var, fijo in filas
    ])

tarifas = pricing.derivar_tarifas(pct, params)
print(tarifas.to_string(index=False))
ruta_csv = rutas.REPORTES / "nb06_tarifas_v1.csv"
tarifas.to_csv(ruta_csv, index=False)
print(f"\nTarifas guardadas en {ruta_csv.name}")

drivers = ["reduccion_relativa", "tasa_basal", "valor_consulta", "volumen_mensual",
           "margen", "coste_sms"]
corr = sim[drivers + ["valor_neto"]].corr()["valor_neto"].drop("valor_neto").sort_values()
print("Correlación con el valor neto mensual:")
print(corr.round(3).to_string())

def persistir_sidecar(metadata: dict[str, Any], ruta: str | Path) -> None:
    with open(ruta, "w", encoding="utf-8") as f:
        json.dump(metadata, f, indent=2, ensure_ascii=False, default=str)
    logger.info("Sidecar guardado en %s.", ruta)

sidecar = {
    "notebook": "06_pricing_montecarlo",
    "n_iteraciones": int(pricing.N_ITER),
    "seed": int(pricing.SEED_MONTECARLO),
    "escenario": "SMS uniforme (share_targeted=1.0)",
    "parametros": vars(params),
    "percentiles_valor_neto": pct,
    "tarifas": tarifas.to_dict(orient="records"),
    "fuente_efecto": ("Reducción relativa del no-show de literatura experimental "
                      "(Robotham 2016; Gurol-Urganci 2013). NO se usa la estimación "
                      "causal propia de NB04 por no ser identificable de forma estable."),
    "escenarios_pendientes": ("SMS dirigido y SMS + escalado multicanal dependen de los "
                              "clústeres de NB05; se incorporan cuando NB05 esté disponible."),
}
import json
ruta_sidecar = rutas.REPORTES / "nb06_metadatos_v1.json"
pricing.persistir_sidecar(sidecar, ruta_sidecar)
print(f"Sidecar guardado en {ruta_sidecar.name}")

assert (sim["valor_neto"] > sim["valor_bruto"]).sum() == 0, "neto no puede superar al bruto"
assert pct["p10"] <= pct["p50"] <= pct["p90"], "percentiles deben ser monótonos"
assert len(tarifas) == 3, "deben derivarse tres tarifas"
print("Checklist NB06: OK")
print("NB06 completo.")
```

B.7 Benchmarking de parámetros de mercado

src/mercado.py · notebooks/07_benchmarking_mercado.ipynb

```
from __future__ import annotations
import json
import logging
from pathlib import Path
from typing import Any
import matplotlib.pyplot as plt
```

```
import numpy as np
import pandas as pd
import seaborn as sns
logger = logging.getLogger(__name__)
FECHA_ACCESO: str = "2026-06-14"
DPI_FIGURAS: int = 200
PARAMETROS_MONTECARLO: list[dict[str, Any]] = [
    {
        "parametro": "reduccion_relativa_noshow",
        "descripcion": "Reducción relativa del no-show por recordatorio",
        "distribucion": "triangular",
        "min": 0.18, "moda": 0.25, "max": 0.335,
        "unidad": "proporción",
        "tipo": "verificado",
        "fuente": "Robotham 2016 (RR no-show 0,75 [0,68–0,82]); Gurol-Urganci 2013 pub3
(32,2%→21,4%)",
    },
    {
        "parametro": "tasa_basal_noshow",
        "descripcion": "Tasa basal de inasistencia de la clínica",
        "distribucion": "triangular",
        "min": 0.10, "moda": 0.125, "max": 0.20,
        "unidad": "proporción",
        "tipo": "verificado",
        "fuente": "Hernández-García 2018 (12,5%, referencia contextual); rango sectorial",
    },
    {
        "parametro": "valor_consulta",
        "descripcion": "Valor económico de una consulta privada",
        "distribucion": "triangular",
        "min": 60.0, "moda": 100.0, "max": 160.0,
        "unidad": "€",
        "tipo": "verificado",
        "fuente": "IMDA (160 €); Clínica Buenavista (60/100 €)",
    },
    {
        "parametro": "coste_sms",
        "descripcion": "Coste unitario del SMS",
        "distribucion": "triangular",
        "min": 0.034, "moda": 0.045, "max": 0.096,
        "unidad": "€",
        "tipo": "verificado",
        "fuente": "Esendex (0,034–0,096 €); LabsMobile (0,045 €)",
    },
    {
        "parametro": "volumen_mensual",
        "descripcion": "Volumen mensual de citas de la clínica",
        "distribucion": "uniforme",
        "min": 200.0, "moda": np.nan, "max": 800.0,
        "unidad": "citas/mes",
        "tipo": "supuesto",
        "fuente": "Supuesto declarado (sensibilidad sobre el tamaño de la clínica)",
    },
    {
        "parametro": "margen_contribucion",
        "descripcion": "Margen de contribución por consulta recuperada",
        "distribucion": "triangular",
        "min": 0.60, "moda": 0.75, "max": 0.90,
        "unidad": "proporción",
        "tipo": "supuesto",
        "fuente": "Supuesto declarado (estructura de coste de servicios sanitarios)",
    },
]
EVIDENCIA_PRECIOS: list[dict[str, Any]] = [
    {
        "categoria": "consulta", "proveedor": "Clínica Buenavista (Madrid)",
        "servicio": "Fisioterapia (valoración)", "precio_eur": 60.0,
        "url": "https://clinicabuenavista.com/servicios/precios/",
    },
    {
        "categoria": "consulta", "proveedor": "Clínica Buenavista (Madrid)",
        "servicio": "Ginecología", "precio_eur": 100.0,
        "url": "https://clinicabuenavista.com/servicios/precios/",
    },
    {
        "categoria": "consulta", "proveedor": "Clínica Buenavista (Madrid)",
        "servicio": "Urología", "precio_eur": 100.0,
        "url": "https://clinicabuenavista.com/servicios/precios/",
    },
```

```
    {
        "categoria": "consulta", "proveedor": "IMDA",
        "servicio": "Dermatología (primera consulta)", "precio_eur": 160.0,
        "url": "https://www.imda.es/tarifas/",
    },
    {
        "categoria": "sms", "proveedor": "Esendex",
        "servicio": "Pack prepago 50.000 SMS", "precio_eur": 0.034,
        "url": "https://www.esendex.es/precios/",
    },
    {
        "categoria": "sms", "proveedor": "LabsMobile",
        "servicio": "25.000 SMS a España", "precio_eur": 0.045,
        "url":        "https://www.labsmobile.com/es/blog/cuanto-puede-costar-una-campana-de-sms-
marketing",
    },
    {
        "categoria": "sms", "proveedor": "Esendex",
        "servicio": "Pack prepago 500 SMS", "precio_eur": 0.096,
        "url": "https://www.esendex.es/precios/",
    },
]
FUENTES_EFECTO: list[dict[str, Any]] = [
    {
        "fuente": "Robotham et al. (2016), BMJ Open",
        "metrica": "RR no-show 0,75 [0,68–0,82]; −25% de inasistencia",
        "rol": "Ancla MODA (0,25) y MIN (0,18); meta-análisis más reciente sobre no-show",
        "url": "https://doi.org/10.1136/bmjopen-2016-012116",
    },
    {
        "fuente": "Gurol-Urganci et al. (2013), Cochrane CD007458.pub3",
        "metrica": "Asistencia 67,8%→78,6%; no-show 32,2%→21,4%; RR asistencia 1,14",
        "rol": "Ancla MAX (0,335); reducción relativa entre brazos",
        "url": "https://doi.org/10.1002/14651858.CD007458.pub3",
    },
    {
        "fuente": "Guy et al. (2012), Health Services Research",
        "metrica": "OR asistencia 1,48 [1,23–1,72] (8 ECA)",
        "rol": "Comprobación de coherencia; no es ancla de la triangular",
        "url": "https://doi.org/10.1111/j.1475-6773.2011.01342.x",
    },
    {
        "fuente": "Hernández-García et al. (2018), J Healthc Qual Res",
        "metrica": "Absentismo 12,5% (consulta externa, Zaragoza)",
        "rol": "Referencia contextual de la tasa basal española (moda 0,125)",
        "url": "https://doi.org/10.1016/j.cali.2017.12.006",
    },
]

def configurar_estilo() -> None:
    sns.set_theme(context="notebook", style="whitegrid", palette="deep")
    plt.rcParams.update({
        "figure.dpi": 100, "savefig.dpi": DPI_FIGURAS,
        "axes.titlesize": 12, "axes.labelsize": 10, "legend.fontsize": 9,
        "xtick.labelsize": 9, "ytick.labelsize": 9,
    })

import sys
from pathlib import Path
sys.path.append(str(Path.cwd().parent / 'src'))
sys.path.append(str(Path.cwd() / 'src'))
import pandas as pd
import mercado
import rutas
mercado.configurar_estilo()
pd.set_option('display.max_colwidth', 60)
RUTA_NB06_META = rutas.REPORTES / 'nb06_metadatos_v1.json'

def tabla_parametros() -> pd.DataFrame:
    df = pd.DataFrame(PARAMETROS_MONTECARLO)
    df["fecha_acceso"] = FECHA_ACCESO
    return df

params = mercado.tabla_parametros()
params[['parametro', 'distribucion', 'min', 'moda', 'max', 'unidad', 'tipo']]

params[['parametro', 'fuente', 'fecha_acceso']]
```

```
def tabla_evidencia_precios() -> pd.DataFrame:
    df = pd.DataFrame(EVIDENCIA_PRECIOS)
    df["fecha_acceso"] = FECHA_ACCESO
    return df

evidencia = mercado.tabla_evidencia_precios()
evidencia[['categoria', 'proveedor', 'servicio', 'precio_eur', 'url']]

def plot_precios_consulta(
    evidencia: pd.DataFrame, params: pd.DataFrame, ruta_fig: str | Path,
) -> None:
    cons = evidencia[evidencia["categoria"] == "consulta"].sort_values("precio_eur")
    fila = params.set_index("parametro").loc["valor_consulta"]
    fig, ax = plt.subplots(figsize=(8.5, 4.6))
    etiquetas = cons["proveedor"] + "\n" + cons["servicio"]
    ax.bar(etiquetas, cons["precio_eur"], color="#1f77b4", alpha=0.8,
           edgecolor="white")
    for x, (_, r) in enumerate(cons.iterrows()):
        ax.text(x, r["precio_eur"] + 2, f"{r['precio_eur']:.0f} €",
                ha="center", va="bottom", fontsize=9)
    for valor, nombre, color in (
        (fila["min"], "mín.", "#2ca02c"),
        (fila["moda"], "moda", "black"),
        (fila["max"], "máx.", "#d62728"),
    ):
        ax.axhline(valor, color=color, ls="--", lw=1.3,
                   label=f"Triangular — {nombre} {valor:.0f} €")
    ax.set_ylabel("Precio de consulta (€)")
    ax.set_title("Precios de consulta privada publicados y rango del Monte Carlo")
    ax.legend(loc="upper left")
    fig.tight_layout()
    fig.savefig(ruta_fig, dpi=DPI_FIGURAS, bbox_inches="tight")
    plt.close(fig)
    logger.info("Gráfico de precios de consulta guardado en %s.", ruta_fig)

ruta_fig = rutas.FIGURAS / 'nb07_precios_consulta_v1.png'
mercado.plot_precios_consulta(evidencia, params, ruta_fig)
print('Figura guardada en', ruta_fig)

from IPython.display import Image
Image(filename=str(ruta_fig))

def tabla_fuentes_efecto() -> pd.DataFrame:
    df = pd.DataFrame(FUENTES_EFECTO)
    df["fecha_acceso"] = FECHA_ACCESO
    return df

fuentes = mercado.tabla_fuentes_efecto()
fuentes[['fuente', 'metrica', 'rol']]

def reconciliar_con_nb06(
    params: pd.DataFrame, ruta_nb06_meta: str | Path,
) -> pd.DataFrame:
    with open(ruta_nb06_meta, encoding="utf-8") as f:
        meta = json.load(f)
    p06 = meta["parametros"]
    mapa = {
        "reduccion_relativa_noshow": "reduccion",
        "tasa_basal_noshow": "basal",
        "valor_consulta": "valor",
        "coste_sms": "sms",
        "margen_contribucion": "margen",
    }
    filas = []
    for parametro, prefijo in mapa.items():
        doc = params.set_index("parametro").loc[parametro]
        for stat in ("min", "moda", "max"):
            clave = f"{prefijo}_{stat}"
            documentado = float(doc[stat])
            usado = float(p06[clave])
            filas.append({
                "parametro": parametro, "estadistico": stat,
                "documentado": documentado, "usado_nb06": usado,
                "coincide": np.isclose(documentado, usado),
```

```
            })
    for stat, clave in (("min", "volumen_min"), ("max", "volumen_max")):
        doc = params.set_index("parametro").loc["volumen_mensual"]
        filas.append({
            "parametro": "volumen_mensual", "estadistico": stat,
            "documentado": float(doc[stat]), "usado_nb06": float(p06[clave]),
            "coincide": np.isclose(float(doc[stat]), float(p06[clave])),
        })
    tabla = pd.DataFrame(filas)
    if not tabla["coincide"].all():
        discrepantes = tabla[~tabla["coincide"]]
        raise ValueError(
            f"Parámetros de mercado no coinciden con el NB06:\n{discrepantes}")
    logger.info("Reconciliación con NB06: %d parámetros coinciden.", len(tabla))
    return tabla

reconciliacion = mercado.reconciliar_con_nb06(params, RUTA_NB06_META)
print('Todos los parámetros coinciden con el NB06:',
      bool(reconciliacion['coincide'].all()))
reconciliacion

def persistir_tabla(df: pd.DataFrame, ruta: str | Path) -> None:
    df.to_csv(ruta, index=False)
    logger.info("Tabla guardada en %s (%d filas).", ruta, len(df))

def persistir_sidecar(metadata: dict[str, Any], ruta: str | Path) -> None:
    with open(ruta, "w", encoding="utf-8") as f:
        json.dump(metadata, f, indent=2, ensure_ascii=False, default=str)
    logger.info("Sidecar guardado en %s.", ruta)

mercado.persistir_tabla(params, rutas.REPORTES / 'nb07_parametros_mercado_v1.csv')
mercado.persistir_tabla(evidencia, rutas.REPORTES / 'nb07_evidencia_precios_v1.csv')
metadata = {
    'notebook': '07_benchmarking_mercado',
    'fecha_acceso': mercado.FECHA_ACCESO,
    'n_parametros': int(len(params)),
    'n_verificados': int((params['tipo'] == 'verificado').sum()),
    'n_supuestos': int((params['tipo'] == 'supuesto').sum()),
    'n_precios_observados': int(len(evidencia)),
    'reconciliacion_nb06_ok': bool(reconciliacion['coincide'].all()),
    'enfoque': 'Recogida documental de tarifas públicas + triangulación; '
               'sin extracción programática para preservar reproducibilidad.',
}
mercado.persistir_sidecar(metadata, rutas.REPORTES / 'nb07_metadatos_v1.json')
metadata
```