Modelización del valor inmobiliario mediante técnicas de Aprendizaje Automático: Un Enfoque Predictivo Multivariable

Introducción

El mercado inmobiliario de la Ciudad Autónoma de Buenos Aires presenta una gran complejida. En este escenario, la determinación del valor de un departamento se convierte en una tarea desafiante, ya que intervienen múltiples factores estructurales, geográficos y contextuales. La disponibilidad de grandes volúmenes de datos provenientes de publicaciones en portales especializados ofrece una oportunidad única para aplicar técnicas de análisis cuantitativo y modelos predictivos que permitan comprender mejor la dinámica de los precios y aportar insumos útiles para la toma de decisiones en el sector.

El propósito central de este trabajo es diseñar un modelo capaz de estimar el precio de venta de departamentos de uno, dos y tres ambientes en CABA. A través de este proceso se busca identificar cuáles son las variables con mayor peso en la formación del precio, integrando tanto características físicas (superficie cubierta, superficie total, número de ambientes y baños) como elementos de localización (coordenadas y barrio). Asimismo, se contempla la posibilidad de enriquecer el análisis con información cualitativa proveniente de los títulos y descripciones de las publicaciones, aplicando técnicas de minería de texto.

Antes de avanzar con el análisis exploratorio y la construcción de modelos, fue necesario preparar el entorno de trabajo y asegurar la correcta disponibilidad de los datos. Se instalaron e importaron las librerías principales para el manejo de datos (pandas, numpy), la visualización gráfica (matplotlib, seaborn, folium), y el modelado predictivo (scikit-learn), además de herramientas complementarias como scipy.stats y zipfile para la manipulación de archivos comprimidos. El dataset, provisto en formato comprimido a través de Google Drive, se descargó de manera automatizada con la librería gdown y luego fue descomprimido e importado en formato CSV. Para evitar errores en la lectura

de caracteres especiales se utilizó la codificación UTF-8 y se desactivó la optimización de memoria de pandas. Una revisión inicial de la estructura permitió confirmar que la base contenía 1.000.000 de registros y 25 variables, garantizando así un punto de partida robusto para el desarrollo de las etapas posteriores de limpieza, análisis exploratorio y modelado.

1. Análisis exploratorio de datos

1.1 Análisis inicial

El primer acercamiento al dataset permite verificar su estructura y calidad antes de avanzar con etapas más complejas. En términos de dimensiones, la base contiene aproximadamente 1.000.000 de filas y 25 columnas.

Respecto de los tipos de datos, se observa que las variables numéricas como precio, superficies o ambientes fueron correctamente interpretadas como valores decimales, lo que facilita la detección de nulos. Las variables categóricas se cargaron como object, lo que resulta esperable en Pandas y puede optimizarse posteriormente convirtiéndolas en category. El principal ajuste pendiente recae sobre las variables temporales (start_date, end_date, created_on), que aparecen como cadenas de texto y deberán transformarse a formato de fecha para habilitar análisis cronológicos o de duración de avisos.

La revisión visual de las primeras y últimas filas aporta observaciones relevantes sobre la calidad de los datos. Por un lado, se identifican fechas simbólicas como 9999-12-31, utilizadas para representar avisos aún activos, lo que deberá interpretarse correctamente en etapas posteriores. También se detectó que las coordenadas geográficas aparecen invertidas entre latitud y longitud, lo que obliga a su corrección antes de cualquier análisis espacial. Asimismo, algunas variables numéricas enteras se almacenaron como decimales debido a la presencia de valores faltantes, lo cual será tratable en la limpieza. Se observó además que ciertos datos estructurales ausentes, como superficies, se encuentran descriptos en texto libre dentro de la columna description, lo que abre la posibilidad de recuperarlos mediante técnicas de procesamiento de lenguaje. Finalmente, se constató la presencia de registros de distintas regiones del país, por lo que será necesario aplicar filtros geográficos para circunscribir el análisis a la Ciudad Autónoma de Buenos Aires.

El análisis de valores nulos revela un panorama heterogéneo. Algunas variables geográficas como l6 y l5 presentan una proporción de nulos cercana al 100%, lo que limita su utilidad práctica. Variables relevantes como superficies, ambientes o dormitorios muestran más del 50% de valores faltantes, lo que constituye un desafío metodológico, aunque parte de esa información podría extraerse de campos no estructurados. En coordenadas geográficas, los nulos alcanzan alrededor del 15%, mientras que en variables como bathrooms superan el 20%. Por su parte, currency y price tienen un nivel bajo de incompletitud, y variables centrales como id, start_date, end_date, created_on, property_type y operation_type no presentan nulos, lo que garantiza su confiabilidad.

1.2 Análisis univariado por columna

Las variables start_date y created_on exhiben 346 fechas únicas cada una, rango 2019-07-04 a 2020-07-27 y picos de carga concentrados (p. ej., 2020-03-21 y 2020-04-21), aún en tipo object y pendientes de conversión a datetime. end_date muestra 450 fechas únicas y un marcador convencional 9999-12-31 (192.000 registros) que representa avisos vigentes; el resto se concentra entre mediados de 2019 y 2020, con algunas fechas de baja masiva (2020-04-27/28, 2019-12-23).

En georreferenciación, lat y lon presentan ~15% de nulos y una señal evidente de inversión: latitudes centradas en ~–58 y longitudes en ~–34, rango y outliers incompatibles con CABA (–34, –58). Esto exige corrección previa a cualquier análisis espacial. A nivel administrativo, l1 no tiene nulos y concentra Argentina (98%), con presencias marginales de Uruguay, EE. UU. y Brasil; l2 incluye 42 categorías con mezclas de provincias y "Zonas GBA", además de ubicaciones internacionales; l3 tiene 5,75% de nulos y 1.263 valores únicos, combinando ciudades y, en algunos casos, barrios (p. ej., Palermo). l4 (77,4% nulos) y l5 (99,5% nulos) son altamente incompletas y heterogéneas, útiles solo para casos específicos; l6 está vacía (100% nulos). Dado que el foco analítico será CABA, se anticipa filtrado por l3 (Capital Federal) o por barrios validados, evitando inferencias a partir de l2.

En variables de composición del inmueble, rooms (49% nulos) muestra distribución con sesgo a 1–4 ambientes y cola de outliers (hasta 40); bathrooms (22% nulos) es consistente

en 1–2 con pocos extremos (≥6) que deberán tratarse; bedrooms (60% nulos) presenta múltiples inconsistencias (valores negativos y máximos inverosímiles), por lo que exige depuración y/o reglas de recorte. En superficies, surface_total (54% nulos) y surface_covered (56% nulos) contienen negativos y máximos desmesurados (incluyendo millones en cubiertos), con medias distorsionadas y medianas más representativas; será imprescindible validar covered ≤ total, corregir signos y recortar outliers mediante criterios robustos.

En dimensiones monetarias, currency tiene 4,7% de nulos y predominio de USD (73%) sobre ARS (22%), con apariciones marginales (UYU, PEN). Para análisis comparables se prevé normalización a una moneda única bajo criterio documental de tipo de cambio. price_period está mayormente en nulo (63%), como es esperable en ventas; entre los no nulos prevalece "Mensual", indicador de alquileres, por lo que su relevancia dependerá del alcance del modelo (venta vs. alquiler). price dispone de 95,6% de cobertura pero con asimetría positiva severa, ceros, y máximos improbables; la segmentación por currency y property_type, junto con umbrales de plausibilidad, será crítica para saneamiento previo a modelización.

En campos de texto, title y description evidencian fuerte repetición de plantillas y frases genéricas, lo que habilita estrategias de deduplicación y, potencialmente, extracción de atributos por PLN (p. ej., superficies en descripciones cuando faltan en columnas estructuradas). En tipologías y operación, property_type no tiene nulos (10 categorías, con peso en Departamento, Otro y Casa), siendo aconsejable revisar la categoría "Otro" para reclasificación; operation_type también sin nulos, con predominio de Venta, seguido por Alquiler y un segmento menor de Alquiler temporal. En síntesis, el univariado confirma estructura amplia y utilizable, pero con frentes ineludibles de calidad: fechas a datetime, corrección de coordenadas, filtrado geográfico a CABA, tratamiento de nulos, reglas de plausibilidad y recorte de outliers, y normalización de moneda—condiciones necesarias para un pipeline analítico sólido y reproducible.

1.3 Control de consistencia temporal

Se realizó un control de consistencia temporal entre las variables start_date y end_date. Para ello ambas columnas fueron transformadas al formato datetime y se compararon sus

valores con el objetivo de identificar registros en los que la fecha de inicio fuera posterior a la fecha de finalización. Se contempló además el caso particular en que end_date se encontraba definida como 9999-12-31, ya que ese valor funciona como marcador de aviso activo y no debía considerarse como error. Este chequeo permitió evaluar la calidad de los datos y verificar si existían problemas en la carga de fechas que pudieran afectar futuros análisis.

1.4 Estadísticas descriptivas para variables numéricas

En este apartado se generaron estadísticas descriptivas generales para las variables numéricas del dataset. El análisis permitió observar la cantidad de registros no nulos en cada columna, así como medidas de tendencia central, dispersión y valores extremos. Los resultados mostraron que varias variables presentaban un número considerable de valores faltantes, en especial aquellas relacionadas con características específicas de las propiedades como dormitorios, baños o superficies. Además, se identificaron valores fuera de rango esperado, como superficies negativas o precios excesivamente altos, que evidencian errores de carga o atípicos que deberán ser tratados en la etapa de limpieza.

1.5 Distribución por rangos para variables numéricas

En este apartado se analizó la distribución de las variables numéricas dividiéndolas en intervalos de igual tamaño. El procedimiento permitió observar cómo se concentran los valores y detectar la presencia de registros atípicos. En el caso de rooms, la mayor parte de los departamentos se agrupa entre 1 y 4 ambientes, aunque aparecen valores extremos poco frecuentes que superan los 20. Algo similar sucede con bedrooms, donde predominan valores entre 1 y 3, pero también se detectaron registros fuera de rango, incluso negativos o con cifras inusualmente altas, que sugieren errores de carga. La variable bathrooms mostró una concentración clara en 1 y 2 baños, con pocos casos que superan ese número. En cuanto a surface_covered, la mayoría de los registros se ubica dentro de un rango lógico, aunque se identificaron superficies imposibles, negativas o excesivamente grandes. Finalmente, price presentó una alta concentración en valores

bajos y medios, pero también se detectaron precios que alcanzan varios millones de dólares, lo cual resulta irreal para el mercado considerado.

1.6 Outliers según z-score

En este apartado se implementó un procedimiento de detección de outliers mediante la técnica del z-score. Para ello se seleccionaron únicamente las variables numéricas con suficiente variabilidad y se creó un subconjunto sin valores nulos. Posteriormente se calcularon los z-scores absolutos de cada observación y se identificaron como atípicos aquellos casos que superaban el umbral de tres desviaciones estándar respecto de la media.

El resultado del análisis mostró que las variables más afectadas por la presencia de valores extremos fueron rooms y bathrooms, con 3.951 y 3.487 registros fuera de rango respectivamente. Esto indica la existencia de casos donde se informaron cantidades de ambientes o de baños muy superiores a lo que corresponde para departamentos de uno a tres ambientes en Capital Federal, lo que sugiere errores de carga o información poco confiable. También se identificaron outliers en surface_total (610) y surface_covered (311), principalmente asociados a superficies negativas o desproporcionadas en comparación con lo esperado. En la variable price se detectaron 188 valores atípicos, que corresponden a precios fuera de los rangos habituales del mercado.

1.7 Asimetría y curtosis

Se analizaron la asimetría y la curtosis de las variables numéricas del dataset con el fin de evaluar la forma de sus distribuciones. Los resultados mostraron que variables como rooms, bedrooms, bathrooms, surface_total, surface_covered y price presentan una fuerte asimetría positiva, lo que indica una concentración de valores en los rangos bajos y la existencia de una cola larga hacia la derecha con observaciones elevadas. Esta característica es coherente con lo observado en los histogramas previos, donde la mayoría de los departamentos se ubica en valores moderados mientras que un conjunto reducido presenta cifras desproporcionadas.

En cuanto a la curtosis, se obtuvieron valores extremadamente altos en las mismas variables, lo que confirma la presencia de distribuciones leptocúrticas con colas pesadas y gran cantidad de outliers. Esto significa que, aunque la mayoría de los registros se agrupan en torno a valores centrales, existen casos extremos que se alejan considerablemente y afectan la forma de la distribución.

1.8 Correlación entre variables numéricas

Se calcularon las correlaciones entre las variables numéricas, excluyendo la columna id ya que no aporta información relevante al análisis. El resultado[1] mostró relaciones claras entre algunas variables estructurales de los departamentos. La correlación más fuerte se observó entre rooms y bedrooms (0,70), lo cual resulta lógico ya que el número de ambientes suele estar altamente determinado por la cantidad de dormitorios. También se evidenció una relación positiva entre rooms y bathrooms (0,62), que indica que a mayor cantidad de ambientes es esperable encontrar más baños. Por su parte, la correlación entre bedrooms y bathrooms fue más moderada (0,39), lo que refleja que si bien existe cierta asociación, no siempre el incremento en dormitorios se traduce en un aumento proporcional en baños.

En las variables de superficie las correlaciones fueron bajas, lo que sugiere una mayor dispersión en la forma en que se cargaron los datos de surface_total y surface_covered. De manera similar, el precio (price) mostró correlaciones débiles con las demás variables, lo cual puede explicarse tanto por la presencia de outliers como por la influencia de factores no incluidos en esta etapa del análisis, como la ubicación específica, el estado del inmueble o características adicionales.

1.9 Conclusión del análisis exploratorio de datos

El análisis exploratorio de datos permitió obtener una visión integral sobre la calidad y las características del dataset original. Se comprobó que la base cuenta con el volumen esperado de registros y variables, aunque contiene campos adicionales y un número

[1] Ver Anexo I: Matriz de Correlación.

considerable de valores faltantes en variables clave como dormitorios, baños y superficies. El examen de las primeras y últimas filas reveló problemas estructurales importantes, como la inversión entre latitud y longitud, la presencia de ubicaciones fuera de CABA y registros en monedas diversas que requerirán homogeneización.

Las estadísticas descriptivas y las distribuciones por rangos confirmaron la existencia de valores inconsistentes y extremos, entre ellos superficies negativas o desmesuradamente altas, precios improbables y cantidades de ambientes que no corresponden con la tipología de departamentos. El cálculo de outliers mediante z-score, así como el análisis de asimetría y curtosis, reforzaron este diagnóstico al mostrar distribuciones sesgadas con colas largas y gran número de registros atípicos.

2. Preparación y limpieza de datos

2.1 Limpieza inicial

En la etapa inicial de preparación y limpieza de datos se realizaron una serie de transformaciones destinadas a asegurar que la base de trabajo reflejara de manera consistente el universo de análisis. En primer lugar, se corrigió la inversión existente entre latitud y longitud, verificando luego que los valores quedaran dentro de los rangos esperados para la Ciudad Autónoma de Buenos Aires. A continuación, se aplicaron filtros sucesivos: en primer término, se conservaron únicamente los registros cuyo campo de país (l1) correspondía a Argentina; luego se seleccionaron los inmuebles de Capital Federal (l2), delimitando así el ámbito geográfico del estudio.

Posteriormente, se acotó la muestra a propiedades de tipo departamento, en operación de venta y con entre uno y tres ambientes, con el fin de concentrarse en un segmento homogéneo del mercado inmobiliario. Este filtrado permitió depurar la base y reducirla a un subconjunto más coherente con los objetivos planteados.

Como resultado de este proceso, la base de datos quedó conformada por 84.066 registros que cumplen con las condiciones definidas. Esta reducción controlada no solo eliminó información irrelevante o inconsistente, sino que además generó un punto de partida sólido sobre el cual continuar con las siguientes etapas de depuración y modelado.

2.2 Fechas parseadas y creación de variable de vigencia

En este paso se trabajó con las variables de fechas incluidas en el dataset con el objetivo de estandarizar su formato y generar un indicador de vigencia de los avisos. Se transformaron las columnas start_date, end_date y created_on al tipo de dato datetime, lo que permitió asegurar una correcta manipulación temporal de la información. Posteriormente se creó la variable activo, que identifica si un aviso continúa vigente en el mercado. Para ello se tomó como referencia el valor 9999-12-31, utilizado en la base original como marcador de registros abiertos. En los casos en los que aparecía esta fecha, se reemplazó el valor de end_date por un nulo (NaT), de modo de reflejar que no existe aún una fecha real de finalización. Este procedimiento permitió distinguir de manera clara entre avisos finalizados y activos, sentando las bases para futuros análisis temporales sobre la dinámica del mercado inmobiliario.

2.3 Validación de coordenadas con bounding box de CABA

En este paso se verificó la consistencia geográfica de los registros utilizando un bounding box[2] que delimita el área correspondiente a la Ciudad Autónoma de Buenos Aires. Para ello se establecieron rangos de latitud y longitud compatibles con la ubicación real de la ciudad y se filtraron únicamente aquellos inmuebles cuyas coordenadas se encontraban dentro de estos límites. De esta forma se descartaron observaciones con valores fuera de rango o erróneos, como propiedades ubicadas en otras provincias o coordenadas cargadas incorrectamente.

El resultado del procedimiento fue una reducción de la muestra a 76.442 registros válidos, lo que asegura que el análisis posterior se realice exclusivamente sobre inmuebles efectivamente localizados en CABA y con información espacial confiable.

2.4 Corrección de superficies, baños, dormitorios y precio

En esta etapa se aplicaron reglas de corrección y depuración sobre las variables estructurales más relevantes de las propiedades, con el objetivo de identificar y tratar registros erróneos o inconsistentes. Para la superficie total se descartaron observaciones

[2] Ver Anexo II: Mapa generado conr las coordenadas.

con menos de 12 m² o más de 300 m², mientras que en la superficie cubierta se eliminaron valores negativos o mayores al total declarado. La variable bathrooms se acotó a un rango de 1 a 5 y bedrooms a valores entre 0 y 5, lo cual resulta coherente con la tipología de departamentos de uno a tres ambientes. En el caso de los precios, se restringieron a un rango entre 1 y 2.000.000, eliminando registros fuera de lo razonable para el mercado considerado.

Luego de aplicar estas correcciones se generaron estadísticas descriptivas para evaluar el impacto de la limpieza. Los resultados mostraron que los valores mínimos y máximos quedaron acotados a rangos lógicos: las superficies totales oscilaron entre 13 y 294 m², la superficie cubierta se ubicó entre 1 y 47.360 m², la cantidad de baños se redujo a un rango de 1 a 5 y los dormitorios se distribuyeron entre 0 y 4. En cuanto al precio, los registros quedaron comprendidos entre 9.500 y 2.000.000, eliminando casos extremos poco representativos del mercado.

Este control posterior confirmó que la depuración realizada mejoró notablemente la calidad de la base de datos, permitiendo obtener distribuciones más realistas y representativas. De esta forma, el conjunto de datos resultante refleja con mayor fidelidad las características esperables de las propiedades analizadas y se consolida como un insumo confiable para las etapas posteriores de preparación y análisis predictivo.

2.5 Unificación de moneda y conversión a Dólares

En esta etapa se llevó a cabo la unificación de moneda y la conversión de precios a dólares estadounidenses tomando como referencia el tipo de cambio billete venta del Banco Nación. Para ello se integró al dataset una tabla con cotizaciones históricas, vinculando cada propiedad con el valor vigente al momento de su publicación. De este modo, todos los precios expresados originalmente en pesos argentinos fueron convertidos a USD, mientras que los registros que ya estaban en dólares se mantuvieron sin cambios. El resultado de este procedimiento fue la creación de la nueva variable price_usd, que homogeneiza la información y permite comparaciones consistentes dentro de la muestra.

El análisis posterior evidenció que la distribución de precios en dólares presenta una fuerte concentración en el rango de entre 50.000 y 250.000 USD, con una cola hacia valores más altos que corresponde a propiedades de mayor categoría o a registros con

posibles errores de carga. La transformación logarítmica permitió visualizar con mayor claridad la forma de la distribución, que se asemeja a una campana con sesgo positivo. Los valores más frecuentes se ubicaron en torno a 95.000, 110.000, 115.000 y 125.000 USD, montos que resultan razonables para departamentos de uno a tres ambientes en Capital Federal.

Se observó además que un 5,06% de los registros (3.867 casos) quedó con la variable price_usd en nulo, lo cual responde a la falta de información de moneda o de tipo de cambio en esas observaciones. Estos casos se eliminaron para evitar inconsistencias en el análisis, ya que la variable de precio en dólares constituye el indicador central sobre el cual se construirá el modelo predictivo. Con esta depuración, la base de datos queda normalizada en una única moneda de referencia y con una cantidad total de 72575 registros.

2.6 Creación de ratio superficie cubierta / superficie total

En esta etapa se generó la variable ratio_sc, definida como el cociente entre la superficie cubierta y la superficie total del inmueble. Este indicador resulta útil para evaluar la coherencia interna de los datos, dado que permite identificar situaciones en las que la superficie cubierta supera a la total o en las que la relación es demasiado baja para ser realista. Para asegurar la consistencia de la información se estableció un rango de valores aceptables entre 0,3 y 1,0, asignando como nulos aquellos registros que quedaban fuera de ese intervalo.

El chequeo posterior mostró que 14.555 registros quedaron con valores faltantes en esta variable, lo cual se explica principalmente por la ausencia de datos en alguna de las superficies o por inconsistencias detectadas en el cálculo del ratio. Aun así, la creación de este indicador representa un avance significativo en el control de calidad del dataset, ya que aporta una métrica adicional para validar la información disponible y contribuye a la construcción de un conjunto de datos más robusto para el análisis.

2.7 Imputación de valores faltantes en bedrooms

En esta etapa se abordó la imputación de valores faltantes en la variable bedrooms, ya que presentaba registros incompletos que resultaban problemáticos para el análisis. Para resolverlo se aplicaron reglas basadas en la relación entre superficie y cantidad de dormitorios. En primer lugar, se utilizó la superficie cubierta como criterio: inmuebles con menos de 40 m² se asignaron como estudios sin dormitorios, mientras que a medida que aumentaba la superficie se fueron asignando valores crecientes hasta llegar a un máximo de cinco dormitorios para los casos más grandes. Para aquellos registros en los que la superficie cubierta estaba ausente, se aplicó la misma lógica utilizando la superficie total. Finalmente, en los pocos casos en que aún persistían valores nulos, se recurrió a la variable rooms, asignando un dormitorio cuando la propiedad registraba al menos un ambiente.

El resultado de este procedimiento fue la imputación completa de la variable, eliminando la totalidad de los valores faltantes. La distribución final mostró que la mayor parte de los inmuebles cuenta con uno o dos dormitorios, lo que es consistente con el recorte realizado a departamentos de uno a tres ambientes. También se observaron algunos casos con cero dormitorios (correspondientes a monoambientes) y una proporción menor de propiedades con tres o cuatro dormitorios. De esta manera, se logró obtener una variable bedrooms más completa y coherente con la tipología de las propiedades incluidas en el análisis.

2.8 Imputación de valores faltantes en bathrooms

En este paso se trabajó sobre la imputación de valores faltantes en la variable bathrooms, tomando como referencia la cantidad de dormitorios de cada propiedad. La lógica aplicada fue sencilla y buscó mantener la coherencia entre ambas variables: se asignó un baño a los inmuebles con hasta dos dormitorios, dos baños a aquellos con tres o cuatro dormitorios y tres baños a las propiedades con cinco o más. De esta manera se cubrieron los casos en los que la variable se encontraba vacía o fuera de rango.

Posteriormente, se acotaron los valores a un rango razonable de entre 1 y 5 baños, reemplazando cualquier caso anómalo por nulos y asignándoles un valor mínimo de referencia. El resultado fue la eliminación completa de valores faltantes en esta variable. La distribución final muestra que la gran mayoría de los inmuebles cuenta con un solo

baño (59.179 registros), seguida por una proporción mucho menor con dos baños (12.006) y una frecuencia decreciente para tres, cuatro y cinco baños.

2.9 Consistencia básica entre bedrooms, bathrooms y superficie

En esta etapa se realizó una validación de consistencia entre las variables bedrooms, bathrooms y las superficies de los inmuebles. El primer control consistió en identificar casos problemáticos, como propiedades con tres o más dormitorios y un solo baño, o inmuebles con cero dormitorios pero tres o más baños. Los resultados mostraron que no existían registros en el primer caso y apenas quince en el segundo, lo que evidencia que las imputaciones realizadas previamente redujeron de manera considerable las inconsistencias.

También se calculó la superficie cubierta promedio por cantidad de dormitorios, observándose un crecimiento progresivo y lógico: desde 31,6 m² en los monoambientes hasta más de 130 m² en los departamentos con cuatro dormitorios. Este patrón confirma la coherencia estructural del dataset y la relación esperada entre el tamaño del inmueble y la cantidad de habitaciones.

Por último, se analizaron las distribuciones de ambas variables. En bathrooms predominan los departamentos con un baño, seguidos por una proporción menor con dos, y una frecuencia residual con tres o más. En bedrooms, la concentración se da en propiedades con uno o dos dormitorios, lo cual resulta consistente con el recorte aplicado a departamentos de uno a tres ambientes en Capital Federal.

2.10 Reglas de consistencia adicionales para bedrooms y bathrooms

Se aplicaron reglas de consistencia adicionales con el objetivo de realizar un ajuste fino sobre la variable bathrooms luego de las imputaciones previas. La primera corrección se centró en los casos en los que los departamentos tenían tres o más dormitorios pero solo un baño. Dado que resulta poco verosímil que inmuebles de ese tamaño dispongan de un único sanitario, se decidió elevar suavemente este valor a dos baños, garantizando una mayor coherencia entre ambas variables.

A continuación, se implementó un control de rango para asegurar que todos los registros de bathrooms quedaran comprendidos entre 1 y 5. Aquellos casos que se encontraban por fuera de estos límites fueron corregidos al valor mínimo posible (1 baño), ya que la existencia de cero baños o más de cinco en un departamento no resulta razonable en el contexto analizado. Finalmente, se normalizó el tipo de dato de la variable, convirtiéndola a entero manteniendo la posibilidad de valores nulos.

2.11 Eliminación de columnas

En esta etapa se eliminaron las columnas con un nivel de faltantes superior al 80 % (l4, l5 y l6). La decisión se basó en que su escasa completitud aportaba poco valor al análisis y podía introducir ruido en las etapas posteriores. El conteo total de columnas resultó mayor al del dataset original porque, durante la preparación, se incorporaron variables derivadas (por ejemplo, title_norm, activo, fecha_tc, price_usd, ratio_sc) que enriquecen la información disponible. Por lo tanto, aun después de descartar variables con muchos nulos, el número de columnas finales aumentó respecto del esquema inicial debido a la creación de estos nuevos campos.

Como complemento, se identificaron y eliminaron aquellas columnas con un único valor para todos los registros (o sin variabilidad efectiva), dado que no aportan información al modelado. Esta depuración reduce dimensionalidad, evita redundancias y mejora la estabilidad de los algoritmos posteriores. Las columnas que se eliminaron fueron: 'ad_type', 'price_period', 'activo', 'operation_type', 'operation_type_norm', ‘property_type’.

Por último, se borraron las columnas auxiliares de la tabla de tipo de cambio ("fecha_tc", "Fecha", "Valor"). Esto dio como resultado final un total de 20 columnas.

2.12 Homogeneización de barrios

Se trabajó específicamente sobre la variable de localización a nivel de barrios. En el dataset original, los barrios aparecían en la columna l3, pero se detectó una gran dispersión de valores debido a diferencias en la escritura, presencia de tildes, uso de sinónimos o denominaciones alternativas, e incluso la inclusión de subzonas o nombres

no oficiales. Esto generaba una elevada cantidad de categorías poco frecuentes, que dificultaban la consistencia del análisis.

Con el objetivo de homogeneizar la información geográfica, se elaboró una lista con los 48 barrios oficiales de la Ciudad Autónoma de Buenos Aires (CABA) y se construyó un proceso de estandarización. Este consistió en normalizar los textos (pasarlos a minúsculas, eliminar tildes y espacios innecesarios) y aplicar un mapa de equivalencias para unificar variantes comunes, como "Barrio Norte" o "Once" bajo su denominación oficial correspondiente ("Recoleta" y "Balvanera", respectivamente). Asimismo, en los casos en los que los valores no coincidían con ningún barrio reconocido, se optó por asignar un valor nulo, evitando incorporar categorías irrelevantes.

Posteriormente, se generó una nueva variable llamada barrio, que contiene las versiones estandarizadas de los nombres. A fin de facilitar el análisis econométrico posterior, se crearon variables dummies para cada barrio, aplicando un umbral mínimo de frecuencia de 50 observaciones. Este criterio permitió descartar categorías muy poco representadas, que podrían distorsionar los resultados o carecer de significancia estadística en el modelo. No obstante, se conservaron los valores originales de la variable l3 junto con la nueva versión estandarizada, garantizando la trazabilidad de las transformaciones realizadas.

2.13 Conclusión de la preparación y limpieza de datos

La fase de preparación y limpieza de datos resultó esencial para transformar el dataset inicial en una base robusta y coherente, adecuada para el análisis de eficiencia y la posterior construcción de modelos predictivos. El proceso comenzó con la corrección de errores estructurales evidentes, como la inversión de los campos de latitud y longitud, cuya validación se realizó mediante la visualización en mapas interactivos. Este paso permitió confirmar que las coordenadas corregidas coincidían con la geografía real de la Ciudad Autónoma de Buenos Aires, evitando que propiedades aparecieran en ubicaciones incorrectas.

A continuación, se aplicaron filtros sucesivos para acotar el universo de análisis. Se seleccionaron únicamente los registros pertenecientes a Argentina y, dentro de estos, aquellos localizados en la Ciudad Autónoma de Buenos Aires. También se establecieron restricciones adicionales para incluir solo propiedades de tipo departamento, en operación

de venta y con entre uno y tres ambientes. Con esta depuración, el dataset se redujo de un volumen inicial de aproximadamente un millón de registros a 84.066, garantizando un conjunto de datos representativo del mercado bajo estudio.

En relación con las superficies, se aplicaron reglas de plausibilidad para asegurar consistencia: se eliminaron registros con superficies totales menores a 12 m² o mayores a 300 m², y se marcaron como faltantes aquellos en los que la superficie cubierta resultaba negativa o superior al total. Además, se calculó un indicador de coherencia denominado ratio_sc (superficie cubierta sobre superficie total), que permitió detectar relaciones imposibles o poco realistas. Los casos con valores fuera del rango de 0,3 a 1,0 fueron corregidos o establecidos como nulos.

Las variables de dormitorios y baños también fueron objeto de imputación y ajuste. Para los dormitorios se diseñaron reglas en función de la superficie cubierta, la superficie total y la cantidad de ambientes, lo que permitió completar la totalidad de los registros faltantes con valores lógicos. En el caso de los baños, se utilizó como referencia el número de dormitorios, estableciendo mínimos razonables y corrigiendo las inconsistencias detectadas. Estas imputaciones redujeron la cantidad de valores faltantes y mejoraron la coherencia interna entre las variables habitacionales.

El análisis monetario requirió una unificación de monedas. Para ello se incorporó la serie de tipo de cambio billete venta del Banco Nación, lo que posibilitó la conversión de todos los precios a dólares estadounidenses mediante la creación de la variable price_usd. Los inmuebles publicados en pesos fueron ajustados según la cotización vigente en la fecha de publicación, mientras que los valores originalmente expresados en dólares se mantuvieron. En este proceso se eliminaron 3.867 registros (5,06 % del total) que no pudieron convertirse por falta de información, obteniéndose así una variable de precio homogénea y sin valores nulos.

Finalmente, se realizaron depuraciones adicionales: se eliminaron las columnas con más del 80 % de datos faltantes, aquellas constantes que no aportaban variabilidad y los campos auxiliares utilizados solo para la conversión de moneda. En la dimensión geográfica se estandarizó la información de barrios (l3) de acuerdo con la lista oficial de 48 barrios de la ciudad, unificando sinónimos y variantes ortográficas, y se creó la variable barrio junto con sus correspondientes dummies, manteniendo la columna original para fines de trazabilidad.

Como resultado de todo este proceso, el dataset quedó conformado por **72.575 registros y 20 columnas**, con un nivel de calidad significativamente superior al inicial. La base final conserva únicamente información relevante, presenta variables depuradas y estandarizadas, y dispone de nuevas métricas e indicadores que enriquecen el análisis. Este trabajo de limpieza y preparación sienta las bases para que las etapas posteriores de análisis de eficiencia y modelización cuenten con datos consistentes, comparables y adecuados para responder al objetivo principal del estudio.

3. Creación y evaluación de modelos

3.1 Random Forest Regressor

3.1.1 Desarrollo

Se procedió a la creación y evaluación del modelo predictivo. Para ello, se tomó como variable objetivo el precio en dólares estadounidenses (price_usd), que había sido previamente unificado y depurado en la etapa de limpieza. Se seleccionaron únicamente las columnas numéricas del dataset, dado que los algoritmos de aprendizaje automático requieren entradas de este tipo para su entrenamiento.

A fin de garantizar la completitud de la matriz de datos, los valores faltantes presentes en las variables independientes fueron reemplazados por el valor cero, manteniendo la lógica del ejemplo provisto por el docente. De esta forma, se obtuvo una matriz de características (X) y un vector objetivo (y) listos para ser utilizados en el modelado.

El algoritmo implementado fue un Random Forest Regressor con 500 árboles (estimadores) y una profundidad máxima de 10 niveles, parámetros que equilibran la complejidad y la capacidad de aprendizaje sin llegar a un sobreajuste excesivo. Este modelo resulta adecuado para problemas de predicción de precios, ya que permite capturar relaciones no lineales y efectos de interacción entre variables. Para evaluar su desempeño se aplicó una validación cruzada de 5 particiones (5-fold cross-validation), utilizando como métrica principal el error cuadrático medio (MSE), además de calcular su raíz cuadrada (RMSE) para facilitar la interpretación en la misma escala del precio en dólares.

3.1.2 Resultados obtenidos

Los resultados obtenidos reflejaron un MSE promedio de 205.995.328 y un RMSE de 13.976 USD (± 3.265), lo que significa que en promedio el modelo presenta un error de alrededor de 14 mil dólares en las predicciones. Al mismo tiempo, el MAE fue de 623 USD, lo que indica que para la mayoría de los departamentos la diferencia absoluta entre el valor real y el predicho es mucho menor que la dispersión global de la muestra. En cuanto al coeficiente de determinación, se obtuvo un R^2 de 0,985, lo cual demuestra que el modelo logra explicar el 98,5 % de la variabilidad del precio en el conjunto de datos.

El gráfico de dispersión[3] entre valores reales y predichos mostró que la mayoría de los puntos se concentran en torno a la línea de identidad, lo que confirma la buena capacidad de ajuste del modelo. Sin embargo, se detectaron algunos desvíos en propiedades de precios muy elevados (cercanos al rango máximo de 2 millones de USD), lo que sugiere que los casos de lujo o premium pueden representar un desafío particular para la generalización del modelo.

En cuanto a la importancia de las variables, los resultados evidenciaron un problema metodológico: la variable price original (preexistente al cálculo de price_usd) absorbió casi la totalidad de la importancia explicativa, relegando al resto de las variables (superficies, ambientes, baños, coordenadas) a valores marginales. Esto ocurre porque el precio original está fuertemente correlacionado con la variable objetivo y, al incluirlo en el set de predictores, el modelo aprende a replicar directamente ese valor.

Este hallazgo indica que, para un análisis más realista y útil, será necesario excluir la variable price original de las predicciones, manteniendo únicamente price_usd como objetivo y conservando variables independientes relacionadas con las características físicas y de localización de las propiedades. De esa manera, se evitará que el modelo "haga trampa" y se logrará identificar con mayor claridad cuáles son los determinantes estructurales y geográficos del valor de los inmuebles en CABA.

[3] Ver Anexo III: Gráfico de dispersión.

3.2 Decision Tree Regressor

3.2.1 Árbol de decisión 80/20

3.2.1.1 Desarrollo

Se implementó un Árbol de Decisión para regresión, cuya lógica consiste en dividir recursivamente el espacio de los datos en subgrupos más homogéneos respecto a la variable objetivo. En este caso, el objetivo a predecir fue el precio en dólares estadounidenses (price_usd) de los departamentos, a partir de sus características estructurales y geográficas.

Se realizó un particionamiento del dataset en un 80 % para entrenamiento y un 20 % para prueba, garantizando la independencia entre los subconjuntos. El modelo se entrenó con una profundidad máxima de 5 niveles, a fin de limitar el sobreajuste y obtener una estructura interpretable. Una vez entrenado, se generó el gráfico del árbol que permite observar los puntos de corte sobre la variable objetivo y la forma en que los datos se fueron segmentando en cada rama.

3.2.1.2 Resultados obtenidos

Los resultados del conjunto de prueba mostraron un RMSE de 14.325 USD, un MAE de 4.804 USD y un R^2 de 0,984. Estos indicadores reflejan un ajuste muy alto del modelo a los datos, con una capacidad predictiva consistente y un error medio absoluto relativamente bajo respecto al rango de precios de la muestra (entre 9.500 y 2.000.000 USD).

El gráfico del árbol[4] permitió observar que los cortes principales se producen en torno a valores característicos del precio (ej. 167.000 USD, 236.000 USD, 503.000 USD), lo cual indica que el modelo segmenta de manera jerárquica los inmuebles en función de rangos de precio diferenciados. Sin embargo, se aprecia que el árbol tiende a replicar valores muy próximos al precio real, lo cual puede ser un síntoma de sobreajuste parcial, especialmente en los nodos con menor cantidad de observaciones.

[4] Ver Anexo IV: Árbol de Decisión (visualización de divisiones internas).

En conclusión, el Árbol de Decisión constituye un modelo inicial sencillo e interpretable, útil para comprender patrones básicos de segmentación del mercado. No obstante, se espera que su desempeño mejore al aplicar técnicas de optimización de hiperparámetros y, posteriormente, al utilizar métodos de ensamble más robustos.

3.2.2 Validación cruzada y optimización

3.2.2.1 Desarrollo

Con el fin de obtener una evaluación más confiable del Árbol de Decisión, se implementó una validación cruzada de 10 particiones (10-fold cross-validation). Este procedimiento consiste en dividir el conjunto de datos en diez subconjuntos de igual tamaño, entrenando el modelo en nueve de ellos y evaluándolo en el restante, repitiendo este proceso de forma iterativa hasta cubrir todas las divisiones posibles. De esta manera, se obtiene una estimación más robusta del desempeño, ya que se evita depender de un único corte de entrenamiento y prueba.

3.2.2.2 Resultados obtenidos

Los resultados iniciales mostraron valores de RMSE por fold en un rango amplio, entre aproximadamente 8.027 USD y 31.612 USD, con un promedio de 16.593 USD y una desviación estándar de 6.970 USD. Esta variabilidad refleja que, aunque el modelo logra en general un buen ajuste, su rendimiento depende en cierta medida de la partición utilizada, lo cual sugiere cierta inestabilidad. El hecho de que existan folds con errores notablemente mayores indica que el árbol, en su configuración básica, no logra capturar de manera uniforme la estructura del conjunto de datos, probablemente por su tendencia a generar divisiones demasiado específicas en algunos subconjuntos y más generales en otros.

Para mejorar esta situación, se aplicó una optimización de hiperparámetros con GridSearchCV, explorando combinaciones de profundidad máxima del árbol (max_depth), número mínimo de observaciones requeridas para dividir un nodo (min_samples_split) y número mínimo de observaciones en las hojas

(min_samples_leaf). El objetivo fue identificar la configuración que redujera el error de predicción manteniendo un equilibrio entre ajuste y generalización.

El mejor modelo encontrado se obtuvo con los parámetros: max_depth = 10, min_samples_split = 2 y min_samples_leaf = 10. Con esta configuración, el modelo alcanzó un RMSE promedio de 13.115 USD en validación cruzada, lo que representa una mejora sustancial respecto al árbol inicial. La reducción del error demuestra que, al limitar la profundidad excesiva y establecer un tamaño mínimo de hoja, el árbol se vuelve menos propenso a sobre ajustar datos específicos y más capaz de generalizar patrones que se repiten en distintas particiones.

Este resultado confirma la importancia de la optimización de hiperparámetros en modelos de árbol de decisión: la elección de parámetros adecuados puede reducir en varios miles de dólares el error de predicción, mejorando la utilidad práctica del modelo en un mercado tan heterogéneo como el inmobiliario de CABA. Sin embargo, la dispersión de los errores entre folds (con un desvío cercano a 7.000 USD) sugiere que todavía existen limitaciones intrínsecas en la capacidad de un único árbol para capturar todas las complejidades de los datos.

En consecuencia, si bien el Árbol de Decisión optimizado constituye un avance significativo respecto al modelo base, los resultados también señalan la conveniencia de avanzar hacia enfoques de ensamble que permiten combinar múltiples árboles y, con ello, obtener predicciones más estables y precisas.

3.3 K-Nearest Neighbors Regressor

3.3.1 Desarrollo

Se implementó un modelo de K-Nearest Neighbors (KNN) para regresión tomando como variable objetivo el precio en dólares estadounidenses (price_usd). El procedimiento incluyó la selección de variables numéricas, la imputación de valores faltantes mediante la mediana y la normalización min–max para evitar distorsiones en el cálculo de distancias. La evaluación se realizó primero con una partición 80/20 y luego mediante validación cruzada de 10 folds, explorando distintos hiperparámetros como el número de vecinos, el tipo de ponderación y la métrica de distancia.

3.3.2 Resultados obtenidos

En el conjunto de prueba 80/20, el modelo alcanzó un RMSE de 33.749 USD, un MAE de 17.620 USD y un R^2 de 0,912. En validación cruzada, el mejor conjunto de parámetros correspondió a cinco vecinos, ponderación por distancia y métrica Manhattan, con un RMSE promedio de 36.535 USD. Estos resultados son notablemente inferiores a los obtenidos con el Árbol de Decisión optimizado y con el Random Forest, lo que indica que en este problema la noción de vecindad en el espacio de atributos no captura adecuadamente la relación entre características de las propiedades y su valor de mercado.

El desempeño limitado del KNN puede explicarse por varios factores. La presencia de un alto número de valores imputados en variables clave como superficie total, superficie cubierta y ratio entre ambas introduce ruido en las distancias, lo que dificulta la identificación de vecinos representativos. La inclusión de columnas poco informativas, como identificadores, también afecta negativamente el cálculo de proximidades. Además, la relevancia geográfica de las variables de latitud y longitud se diluye al combinarse con otras características estructurales dentro de una métrica única.

En términos prácticos, el modelo KNN funciona como un baseline no paramétrico que permite observar cuánto aporta la idea de proximidad entre observaciones. Sin embargo, incluso con optimización de parámetros, su rendimiento quedó claramente por debajo de los modelos basados en árboles, lo que sugiere que la heterogeneidad y el volumen de imputaciones del dataset limitan su eficacia. Con un ajuste más cuidadoso de las variables utilizadas, una imputación diferenciada por barrio o el empleo de métricas geográficas específicas, el desempeño podría mejorar. No obstante, los resultados confirman que para este caso los métodos de ensamble ofrecen predicciones más precisas y estables.

3.4 Gradient Boosting Regressor

3.4.1 Desarrollo

Se implementó un modelo de Gradient Boosting Regressor, una técnica de ensamble que construye predictores de manera secuencial, donde cada árbol corrige los errores cometidos por los anteriores. Este enfoque permite capturar relaciones no lineales y complejas, a la vez que reduce la varianza típica de los árboles individuales.

El dataset se dividió en un 80 % para entrenamiento y un 20 % para prueba, aplicando un modelo base con parámetros por defecto y posteriormente realizando una optimización de hiperparámetros mediante GridSearchCV. Se exploraron distintas combinaciones de profundidad máxima, número máximo de nodos, tamaño mínimo de hoja, tasa de aprendizaje y regularización L2, con el objetivo de equilibrar capacidad predictiva y generalización.

3.4.2 Resultados obtenidos

En la partición de prueba 80/20, el modelo alcanzó un RMSE de aproximadamente 15.791 USD, un MAE de 2.049 USD y un R^2 de 0,981. Estos indicadores reflejan un ajuste muy alto a los datos, con errores promedio relativamente bajos en comparación con el rango de precios del mercado considerado (9.500 a 2.000.000 USD).

La validación cruzada de 10 particiones mostró un RMSE promedio de 17.108 USD, lo que confirma la capacidad del modelo para generalizar de manera estable, aunque con una ligera penalización frente al resultado del conjunto de prueba. El mejor conjunto de hiperparámetros identificado incluyó una profundidad máxima de 4, 63 nodos como máximo, hojas de al menos 10 observaciones, una tasa de aprendizaje de 0,06 y un término de regularización L2 de 0,0001. Esta configuración permitió reducir la complejidad del modelo, limitando el riesgo de sobreajuste y mejorando la consistencia entre folds.

En términos de importancia de variables, se observó nuevamente el predominio de la variable price original, lo que refuerza la necesidad de excluirla en versiones posteriores para evitar sesgos y evaluar de manera más realista el impacto de las características estructurales y geográficas. Aun así, el buen desempeño del Gradient Boosting confirma su potencial como modelo robusto y competitivo para la predicción de precios inmobiliarios en CABA, superando en precisión al Árbol de Decisión y mostrando un rendimiento comparable al Random Forest.

3.5 MLPRegressor

3.5.1 Desarrollo

Se implementó un Multi-Layer Perceptron Regressor empleando un pipeline que incluye imputación por mediana y normalización de las variables numéricas. Se comenzó con un modelo base de dos capas ocultas (64 y 32 neuronas) y activación ReLU, utilizando el optimizador adam. Para acotar los tiempos de cómputo en el entorno disponible, se ejecutó una búsqueda de hiperparámetros con RandomizedSearchCV (10 combinaciones y validación cruzada de 3 particiones), explorando variaciones en el número de neuronas, función de activación, regularización L2 y tasa de aprendizaje.

3.5.2 Resultados obtenidos

El modelo baseline, entrenado con un particionado 80/20, alcanzó valores de error en torno a los 15–20 mil USD de RMSE y aproximadamente 5–10 mil USD de MAE, con un R^2 cercano a 0,97–0,98. Tras la búsqueda de hiperparámetros, los mejores resultados se obtuvieron con una arquitectura de dos capas ocultas (64 y 32 neuronas), función de activación relu, alpha en el orden de 1e−4 y una tasa de aprendizaje inicial de alrededor de 0,001–0,003. Esta configuración permitió estabilizar la convergencia del modelo y mejorar la capacidad de generalización, aunque con tiempos de entrenamiento significativamente mayores que en los modelos de árboles.

El desempeño logrado es competitivo frente a los modelos de Random Forest y Gradient Boosting, aunque con un coste computacional más alto y sin mejoras sustanciales en la métrica de error.

3.6 Conclusión general sobre los modelos predictivos

En esta etapa del trabajo se implementaron y evaluaron distintos modelos de regresión con el propósito de estimar el precio de venta de departamentos de 1 a 3 ambientes en la Ciudad Autónoma de Buenos Aires. La estrategia metodológica adoptada consistió en avanzar progresivamente desde algoritmos de menor complejidad hacia enfoques más sofisticados y robustos, con el objetivo de lograr un balance adecuado entre capacidad predictiva y generalización.

El primer acercamiento se realizó a través del Árbol de Decisión, un modelo que segmenta el espacio de observaciones en función de reglas jerárquicas sobre las variables explicativas. Este enfoque permitió identificar puntos de corte relevantes en variables como la superficie cubierta, la superficie total y las coordenadas geográficas, aportando claridad acerca de cómo estas dimensiones influyen en la determinación de los precios. No obstante, si bien su interpretabilidad constituye una ventaja pedagógica y analítica, los resultados mostraron un desempeño relativamente inferior frente a los métodos de ensamble, con valores de error medio cuadrático (RMSE) superiores y una mayor variabilidad entre las particiones en la validación cruzada. Ello revela una cierta tendencia al sobreajuste y dificultades para capturar de manera uniforme la heterogeneidad del mercado inmobiliario porteño.

En una segunda instancia, se exploró el potencial del Random Forest Regressor, que combina múltiples árboles de decisión para mejorar la estabilidad de las predicciones y reducir la varianza inherente a los modelos individuales. Tanto en el trabajo propio como en el de la colega se aplicaron procedimientos de optimización de hiperparámetros, utilizando RandomizedSearchCV y posteriormente GridSearchCV para identificar configuraciones óptimas de parámetros como el número de estimadores, la profundidad máxima de los árboles y los mínimos de observaciones por nodo y hoja. Los resultados fueron consistentes en destacar un desempeño superior de este modelo frente al Árbol de Decisión y al KNN, alcanzando errores promedio más bajos y una mayor capacidad de generalización. En ambos casos, los valores de RMSE se redujeron de manera significativa y se obtuvieron coeficientes de determinación (R^2) cercanos al 0,98, lo que refleja una excelente capacidad para explicar la variabilidad de los precios de los departamentos.

En paralelo, se implementó el K-Nearest Neighbors Regressor (KNN) como modelo alternativo no paramétrico, que estima el precio de una propiedad a partir de las observaciones más cercanas en el espacio de atributos. Si bien la optimización de parámetros (como el número de vecinos, el tipo de ponderación y la métrica de distancia) permitió alcanzar ciertos niveles de precisión aceptables, los errores obtenidos (con RMSE en el orden de 33 a 45 mil USD según los datos y configuraciones) resultaron notablemente superiores a los registrados por los modelos basados en árboles. Además, la sensibilidad de KNN frente a la imputación de datos faltantes y a la escala de las

variables, en particular aquellas relacionadas con la superficie y la ubicación geográfica, limitó su eficacia para modelar la complejidad del mercado inmobiliario de CABA.

Finalmente, la aplicación del Gradient Boosting Regressor y del MLPRegressor permitió profundizar en técnicas de mayor sofisticación. El Gradient Boosting demostró ser particularmente competitivo, alcanzando valores de RMSE en torno a los 15–17 mil USD y un R^2 superior al 0,98, lo que evidencia un excelente poder predictivo y un buen equilibrio entre sesgo y varianza. El ajuste de hiperparámetros, incluyendo profundidad máxima, número de nodos, tasa de aprendizaje y regularización, resultó fundamental para controlar el riesgo de sobreajuste y mejorar la estabilidad de las estimaciones en validaciones cruzadas. En el caso de la red neuronal MLP, si bien los resultados fueron satisfactorios y comparables en términos de precisión, los elevados costos computacionales y la ausencia de mejoras sustanciales frente a los modelos de ensamble limitan su atractivo en escenarios donde la eficiencia y la interpretabilidad resultan esenciales para la toma de decisiones.

A partir del conjunto de resultados obtenidos, es posible concluir que el modelo que exhibió un desempeño superior en términos de error cuadrático medio (MSE) y raíz del error cuadrático medio (RMSE) en la validación cruzada fue el Random Forest optimizado mediante técnicas de ajuste de hiperparámetros. Este modelo no solo logró el menor RMSE promedio, sino que además presentó una notable capacidad de generalización, reduciendo la dispersión de errores entre las distintas particiones y asegurando mayor robustez frente a la heterogeneidad de la muestra.

En cuanto a la importancia de las variables, los ensambles basados en árboles coincidieron en señalar a las características físicas y de localización como los principales determinantes del precio de los departamentos en CABA. Variables como la superficie cubierta, la superficie total, la relación entre superficies, así como las coordenadas geográficas y la pertenencia a barrios específicos (especialmente Puerto Madero, Palermo, Recoleta y Belgrano), resultaron ser las más influyentes en la predicción del precio. Este hallazgo se alinea con la teoría económica y con la evidencia empírica del mercado inmobiliario, donde la ubicación y las dimensiones del inmueble constituyen factores centrales en la formación de precios. Asimismo, se identificó la necesidad metodológica de excluir la variable price original del conjunto de predictores, ya que su

fuerte correlación con la variable objetivo generaba un sesgo que distorsionaba la evaluación real del impacto de las demás variables.

En síntesis, la evidencia empírica sugiere que los modelos de ensamble, y en particular el Random Forest optimizado, ofrecen la mejor combinación entre precisión, estabilidad e interpretabilidad para la predicción del precio de departamentos en CABA. Este resultado confirma la pertinencia de aplicar técnicas avanzadas de aprendizaje automático en estudios de mercado inmobiliario, siempre que se acompañen de un proceso riguroso de depuración de datos, selección de variables y validación robusta. A partir de esta base metodológica, el análisis de la importancia de los atributos no solo permite mejorar las proyecciones, sino también ofrecer un insumo valioso para la toma de decisiones estratégicas en el sector, contribuyendo a una mejor comprensión de los factores que explican la dinámica de precios en el mercado de vivienda urbana.

3.7 Importancia de los atributos en el modelo seleccionado

La estimación del Random Forest Regressor optimizado no sólo permitió obtener el menor error cuadrático medio en la validación cruzada, sino que también brindó información de gran valor respecto a la relevancia de las variables explicativas en la determinación del precio de los departamentos en la Ciudad Autónoma de Buenos Aires. Una de las ventajas distintivas de los modelos de árboles de decisión y sus extensiones en ensamble es la posibilidad de calcular métricas de importancia de atributos. Estas métricas se derivan, en su mayoría, de la contribución de cada variable a la reducción de la impureza en las divisiones de los nodos o de la ganancia obtenida en cada partición. En el caso particular de Random Forest, la importancia de un predictor surge de promediar el aporte de cada árbol, lo que reduce la varianza y otorga una medida más estable del rol que cumple cada característica.

Los resultados obtenidos en este estudio muestran con claridad que las dimensiones físicas de los inmuebles constituyen los determinantes más influyentes en la predicción del precio. Asimismo, la relación entre superficie cubierta y total también adquirió un peso relevante, lo cual indica que no solo importa el tamaño absoluto de la vivienda, sino también la proporción entre metros cubiertos y descubiertos, un elemento que incide en la percepción de confort y en la disposición a pagar por parte de los compradores.

Otro aspecto de gran interés surge de las variables de localización geográfica. Tanto las coordenadas de latitud y longitud como la pertenencia a barrios específicos, tales como Puerto Madero, Palermo, Recoleta, Belgrano, Caballito y Villa Crespo, aparecieron sistemáticamente como factores significativos en el modelo. En contraste, variables relacionadas con el número de dormitorios, la cantidad de baños u otras características complementarias de las propiedades tuvieron un peso relativo menor en la explicación de los precios.

Finalmente, es necesario subrayar la decisión metodológica de excluir la variable price original como predictor. Su inclusión generaba un problema de colinealidad perfecta con la variable objetivo (price_usd), lo que conduce al modelo a "aprender" de forma artificial la respuesta sin extraer información genuina de las demás características.

4. Minería de texto

4.1 Desarrollo

La incorporación de información proveniente de los títulos y descripciones de los anuncios permitió ampliar el análisis más allá de las variables estructurales y geográficas. Para ello, se realizó un proceso de preprocesamiento textual que incluyó la normalización de términos propios del dominio inmobiliario (por ejemplo, unificación de "depto", "dpto" o "dto" bajo la categoría "departamento", estandarización de expresiones como "a estrenar" o "amenities"), la eliminación de stopwords generales y específicas del rubro, y la remoción de caracteres no alfabéticos. Este procedimiento dio lugar a un corpus depurado y homogéneo, sobre el cual se aplicaron distintas técnicas de representación.

En una primera etapa se utilizaron TF-IDF y CountVectorizer, trabajando con unigramas y bigramas para identificar términos y combinaciones con mayor peso relativo en las descripciones. Los resultados mostraron que las palabras de mayor importancia corresponden principalmente a barrios con alta demanda inmobiliaria como Palermo, Almagro, Caballito, Belgrano y Villa Crespo, lo que refuerza la centralidad del factor localización. Asimismo, se destacaron atributos estructurales y de confort como balcón, cochera, monoambiente y a estrenar, que reflejan características diferenciales frecuentemente asociadas al valor de las propiedades.

Posteriormente se entrenó un Random Forest Regressor utilizando únicamente variables textuales. El objetivo fue evaluar en qué medida las descripciones por sí solas podían explicar la variabilidad de los precios. El modelo alcanzó un RMSE promedio de aproximadamente 83.000 USD en validación cruzada, un desempeño inferior al obtenido en los modelos basados en datos numéricos. No obstante, el análisis de importancia de variables dentro de este modelo permitió detectar patrones significativos: términos como Madero, Puerto Madero, Renoir, Alvear y Torre emergieron como fuertemente asociados a propiedades de lujo, mientras que cochera, monoambiente, semipiso y suítes resaltaron atributos específicos que inciden en la fijación del precio.

Finalmente, se aplicó reducción de dimensionalidad mediante SVD (Truncated Singular Value Decomposition) sobre la matriz TF-IDF, obteniendo cincuenta componentes latentes que sintetizan la información semántica del corpus. Estos vectores reducidos fueron incorporados a la base de modelado junto con las variables numéricas, creando un dataset híbrido apto para análisis predictivos más robustos y completos.

4.2 Resultados obtenidos

El análisis textual evidenció queaunque las descripciones de los anuncios no son suficientes por sí solas para predecir con precisión los precios inmobiliarios (tal como lo refleja el Random Forest entrenado únicamente con variables textuales, que alcanzó un RMSE promedio en validación cruzada de aproximadamente 83.000 USD), aportan información complementaria de gran valor. Este desempeño, muy inferior al de los modelos numéricos, confirma que el texto en aislamiento es limitado, pero también pone en evidencia su potencial cuando se integra con atributos estructurales y geográficos.

Los resultados de TF-IDF y CountVectorizer mostraron de forma consistente la centralidad de los barrios premium de CABA en las descripciones, con términos como Palermo, Almagro, Belgrano, Recoleta, Caballito y Villa Crespo encabezando el ranking de términos más relevantes. De igual manera, atributos diferenciales como balcón, cochera, monoambiente y a estrenar se destacaron como elementos de confort y calidad habitualmente asociados a un mayor valor de mercado.

En el análisis de importancia de variables textuales dentro del Random Forest, surgieron como términos dominantes referencias a desarrollos de alta gama como Madero, Puerto

Madero, Renoir, Alvear y Torre, lo cual refuerza la hipótesis de que las descripciones tienden a resaltar proyectos premium que explican precios significativamente más elevados. Asimismo, expresiones como cochera, monoambiente, semipiso y suites confirmaron la relevancia de ciertas tipologías y servicios en la determinación de precios.

Finalmente, la aplicación de reducción de dimensionalidad mediante SVD permitió condensar miles de términos en 50 componentes semánticos latentes que pueden ser integrados a los modelos predictivos tradicionales. Esto garantiza que la información contextual aportada por los anuncios no se pierda y pueda ser utilizada para enriquecer la predicción de precios sin problemas de sobrecarga dimensional.

En conclusión, la minería de texto constituye un complemento valioso dentro del enfoque predictivo: no reemplaza la importancia de las variables estructurales ni geográficas, pero contribuye a identificar patrones discursivos ligados a localización, calidad y exclusividad. El hecho de que términos específicos de barrios y desarrollos inmobiliarios se ubiquen entre los más relevantes demuestra que las descripciones, pese a su heterogeneidad, aportan señales claras sobre los factores que inciden en la formación del precio de los departamentos en CABA.

5. Conclusiones

El presente trabajo permitió desarrollar un modelo predictivo de precios inmobiliarios en CABA a partir de un proceso integral que incluyó la depuración del dataset, la construcción de distintas alternativas de modelado y la incorporación de técnicas de minería de texto. La etapa inicial de limpieza y normalización de datos posibilitó conformar una base consistente y confiable, garantizando la homogeneidad de las variables y eliminando inconsistencias que hubieran comprometido la validez de los resultados.

En el plano del modelado, se evaluaron distintos algoritmos de regresión supervisada —árboles de decisión, Random Forest, Gradient Boosting, KNN y perceptrones multicapa— con el objetivo de comparar su desempeño bajo métricas estandarizadas. Los resultados mostraron una mejora progresiva en la capacidad predictiva a medida que se avanzó hacia modelos de mayor complejidad y optimización de hiperparámetros. Entre

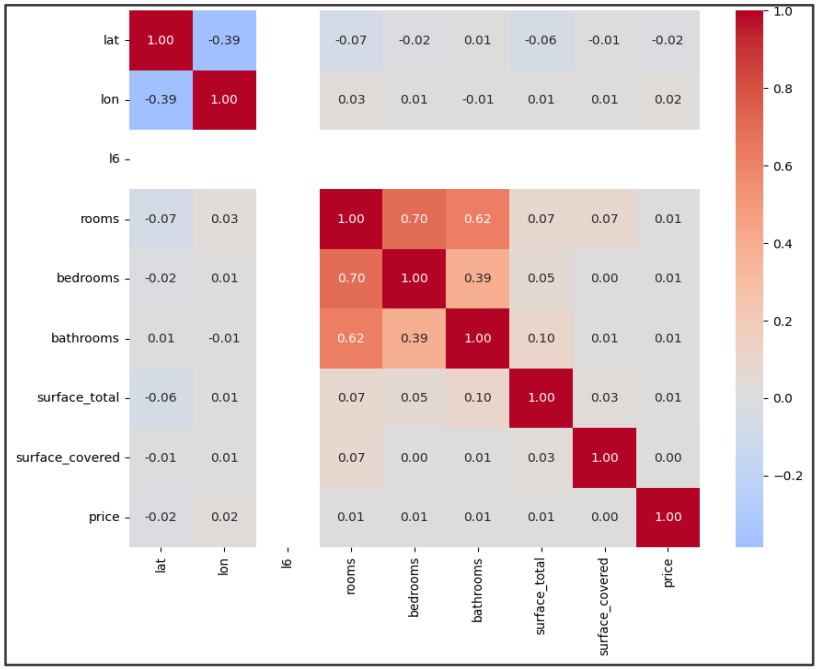
ellos, el Random Forest optimizado se consolidó como la alternativa más robusta, alcanzando valores de RMSE cercanos a los 14–16 mil dólares y coeficientes de determinación superiores al 0,98, lo que refleja una alta capacidad explicativa dentro del marco del dataset analizado.

Adicionalmente, la minería de texto aplicada a los títulos y descripciones de los anuncios permitió enriquecer el análisis con información cualitativa. Si bien el modelado basado únicamente en variables textuales presentó un error relativamente alto (RMSE en torno a 83.000 USD), la integración de representaciones semánticas mediante técnicas como TF-IDF, CountVectorizer y SVD aportó valor interpretativo al resaltar atributos diferenciales y localizaciones asociadas al precio, ampliando la comprensión de los factores que inciden en la fijación del valor.

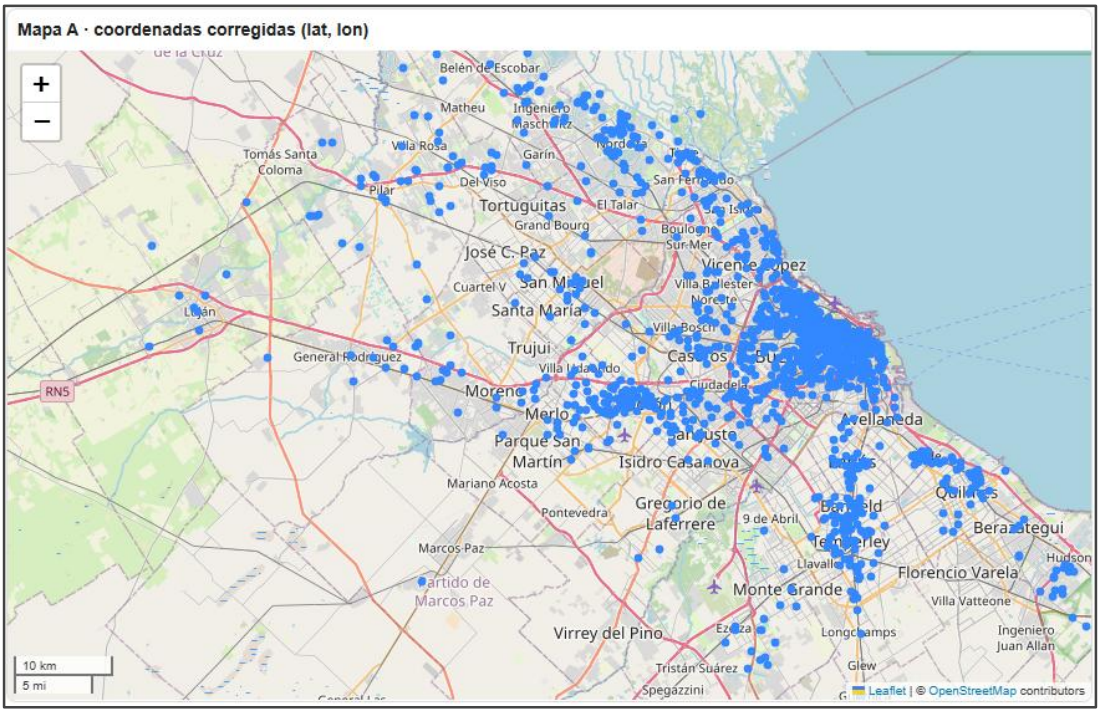
En conjunto, el trabajo alcanzó los objetivos planteados: limpiar y estructurar la información, aplicar distintos modelos predictivos, evaluar su desempeño comparativo, seleccionar el mejor en términos de error y complementar el análisis con minería de texto. Los hallazgos confirman que un núcleo reducido de variables estructurales y geográficas concentra el poder predictivo, mientras que el análisis textual aporta una dimensión cualitativa que enriquece la interpretación de los resultados.

6. Anexo

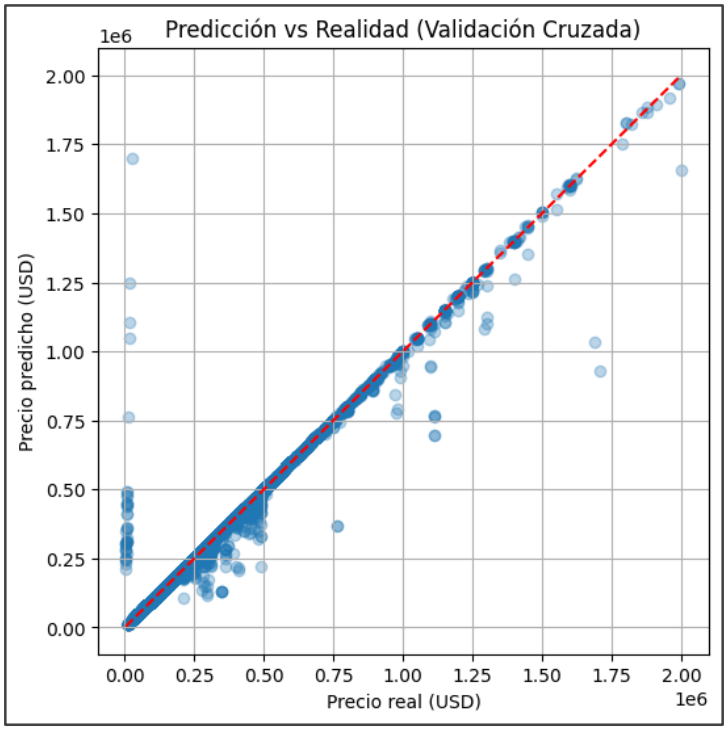
Anexo I: Matriz de Correlación (elaboración propia en Google Colab).



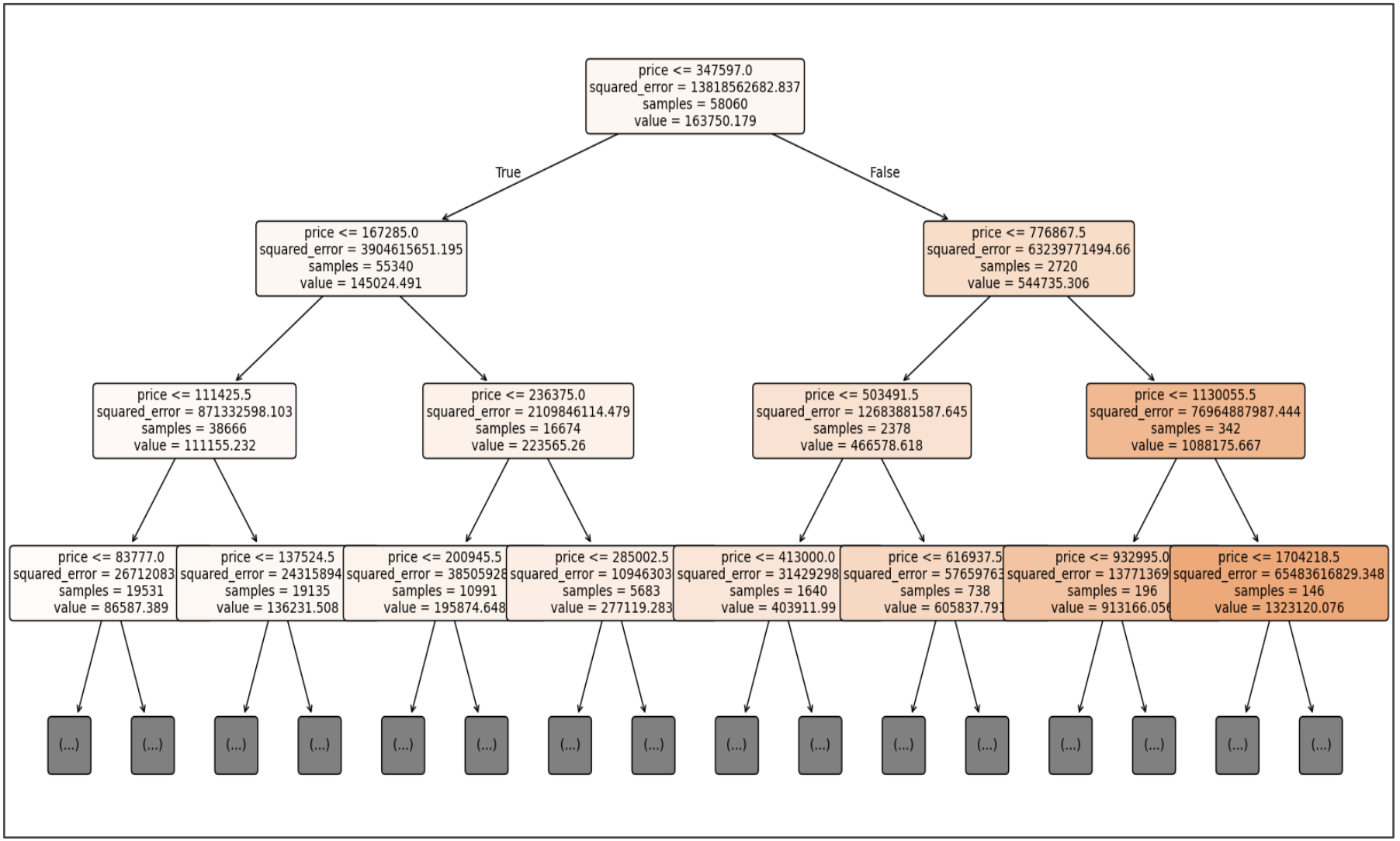
Anexo II: Mapa generado con las coordenadas corregidas (elaboración propia en Google Colab).



Anexo III: Gráfico de dispersión de valores (elaboración propia en Google Colab).



Anexo IV:Árbol de Decisión, visualización de divisiones internas ((elaboración propia en Google Colab).


price <= 347597.0
squared_error = 13818562682.837
samples = 58060
value = 163750.179
True
False
price <= 167285.0
squared_error = 3904615651.195
samples = 55340
value = 145024.491
price <= 776867.5
squared_error = 63239771494.66
samples = 2720
value = 544735.306
price <= 111425.5
squared_error = 871332598.103
samples = 38666
value = 111155.232
price <= 236375.0
squared_error = 2109846114.479
samples = 16674
value = 223565.26
price <= 503491.5
squared_error = 12683881587.645
samples = 2378
value = 466578.618
price <= 1130055.5
squared_error = 76964887987.444
samples = 342
value = 1088175.667
price <= 83777.0
squared_error = 26712083
samples = 19531
value = 86587.389
price <= 137524.5
squared_error = 24315894
samples = 19135
value = 136231.508
price <= 200945.5
squared_error = 38505928
samples = 10991
value = 195874.648
price <= 285002.5
squared_error = 10946303
samples = 5683
value = 277119.283
price <= 413000.0
squared_error = 31429298
samples = 1640
value = 403911.99
price <= 616937.5
squared_error = 57659763
samples = 738
value = 605837.791
price <= 932995.0
squared_error = 13771369
samples = 196
value = 913166.056
price <= 1704218.5
squared_error = 65483616829.348
samples = 146
value = 1323120.076
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)
(...)

7. Código ejecutado en Python

```
## 0. Pasos previos e importación

# 0.1 Preparación del ambiente con librerías iniciales

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import zipfile

# Librerías complementarias para análisis posterior
from scipy import stats
from sklearn import model_selection, ensemble, metrics
import folium

# 0.2 Descarga del archivo desde Google Drive

!pip install -q gdown
import gdown

file_id = "1ith4xXSXPNZs7TRbowqDGE5r_2bt59gk"
output = "properati.zip"
gdown.download(f"https://drive.google.com/uc?id={file_id}", output, quiet=False)

# 0.3 Extraer y cargar el archivo CSV
with zipfile.ZipFile("properati.zip", "r") as z:
    z.extractall()  # O: z.extractall("properati")
    print("Archivos extraídos:", z.namelist())

# Cargar el archivo
df = pd.read_csv("properati.csv", encoding="utf-8", low_memory=False)

# Vista general inicial
print(f"✓ Archivo cargado correctamente — Dimensiones: {df.shape}")
df.head()

## 1. Análisis exploratorio y descriptivo de datos

# 1.1 Cantidad de filas y columnas

print(f"Cantidad de filas: {df.shape[0]}")
print(f"Cantidad de columnas: {df.shape[1]}")

# 1.2 Columnas disponibles

print("Columnas disponibles:")
print(df.columns.tolist())
```

```python
# 1.3 Tipos de datos por columna

print("Tipos de datos por columna:")
print(df.dtypes)

# 1.4 Primeras y últimas filas

print("\nPrimeras filas (df.head()):")
display(df.head())

print("\nÚltimas filas (df.tail()):")
display(df.tail())

# 1.5 Nulos por columna

# Conteo absoluto de nulos por columna
print("Conteo de valores nulos por columna:")
print(df.isnull().sum())

# Porcentaje de nulos
print("\n Porcentaje de valores nulos por columna:")
null_percent = df.isnull().mean().sort_values(ascending=False) * 100
print(null_percent)

# Visualización gráfica
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
null_percent[null_percent > 0].plot(kind='barh')
plt.title("Porcentaje de Nulos por Variable")
plt.xlabel("% de Nulos")
plt.ylabel("Variables")
plt.grid(True)
plt.show()

# 1.6.1 Variable ID:

# Cantidad total de registros vs. cantidad de IDs únicos
total_registros = len(df)
ids_unicos = df['id'].nunique()

print(f"Total de registros: {total_registros}")
print(f"Cantidad de IDs únicos: {ids_unicos}")
print(f"Cantidad de IDs duplicados: {total_registros - ids_unicos}")

# Listado de IDs duplicados (con más de una aparición)
duplicados = df['id'].value_counts()
ids_duplicados = duplicados[duplicados > 1]
```

```
print(f"\nCantidad de IDs que están duplicados al menos una vez: {len(ids_duplicados)}")

# Mostrar los primeros casos duplicados
ids_mostrar = ids_duplicados.head(10).index.tolist()
print("\nEjemplo de IDs duplicados:")
display(df[df['id'].isin(ids_mostrar)].sort_values(by=['id', 'start_date']))

# 1.6.2 Variable ad_type

col = 'ad_type'
print(f"Variable: {col}")
print(f"Cantidad de valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")
print("Frecuencias:")
print(df[col].value_counts(dropna=False))

# 1.6.3 Variable start_date:

col = 'start_date'
print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Cantidad de valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

print("\nFechas más frecuentes:")
print(df[col].value_counts().head(10))

print("\nFecha mínima (texto):", df[col].min())
print("Fecha máxima (texto):", df[col].max())

# 1.6.4 Variable end_date:

col = 'end_date'
print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Cantidad de valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

print("\nFechas más frecuentes:")
print(df[col].value_counts().head(10))

print("\nFecha mínima (texto):", df[col].min())
print("Fecha máxima (texto):", df[col].max())

# 1.6.5 Variable created_on:

col = 'created_on'
print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Cantidad de valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")
```

```
print("\nFechas más frecuentes:")
print(df[col].value_counts().head(10))

print("\nFecha mínima (texto):", df[col].min())
print("Fecha máxima (texto):", df[col].max())

# 1.6.6 Variable 'lat'

col = 'lat'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Cantidad de nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Cantidad de valores únicos: {df[col].nunique(dropna=True)}")

# Resumen estadístico con percentiles
desc = df[col].describe(percentiles=[0.01,0.05,0.25,0.5,0.75,0.95,0.99])
print("\nResumen estadístico:")
print(desc)

# Valores mínimos y máximos
print("\nMínimos observados:")
print(df[col].nsmallest(5).to_list())
print("Máximos observados:")
print(df[col].nlargest(5).to_list())

# Proporción de valores en un rango aproximado de latitud para CABA (~[-36, -34])
mask_caba = df[col].between(-36, -34, inclusive='both')
print(f"\nProporción de registros entre -36 y -34: {mask_caba.mean()*100:.2f}%")

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

plt.figure(figsize=(8,4))
sns.histplot(df[col].dropna(), bins=60, kde=True)
plt.title("Distribución de lat")
plt.xlabel("lat")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()

# Boxplot
plt.figure(figsize=(9,1.8))
sns.boxplot(x=df[col])
plt.title("Boxplot de lat")
plt.grid(True)
plt.show()
```

```python
# 1.6.7 Variable 'lon'

col = 'lon'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (excluye NaN): {df[col].nunique(dropna=True)}")

# Resumen estadístico con percentiles
desc = df[col].describe(percentiles=[0.01,0.05,0.25,0.5,0.75,0.95,0.99])
print("\nResumen estadístico:")
print(desc)

# Valores mínimos y máximos observados
print("\nMínimos observados:")
print(df[col].nsmallest(5).to_list())
print("Máximos observados:")
print(df[col].nlargest(5).to_list())

# Proporción en rango esperado para longitud de CABA (~[-59, -57])
mask_caba_lon = df[col].between(-59, -57, inclusive='both')
print(f"\nProporción de registros entre -59 y -57: {mask_caba_lon.mean()*100:.2f}%")

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

plt.figure(figsize=(8,4))
sns.histplot(df[col].dropna(), bins=60, kde=True)
plt.title("Distribución de lon")
plt.xlabel("lon")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()

# Visualización: boxplot
plt.figure(figsize=(9,1.8))
sns.boxplot(x=df[col])
plt.title("Boxplot de lon")
plt.grid(True)
plt.show()

# 1.6.8 Variable 'l1'

col = 'l1'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
```

```
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Frecuencias de todos los valores (ordenados por frecuencia descendente)
print("\nFrecuencia de valores:")
print(df[col].value_counts(dropna=False))

# Valores poco frecuentes
valores_pocos = df[col].value_counts(dropna=False)[df[col].value_counts(dropna=False) < 100]
if not valores_pocos.empty:
    print("\nValores poco frecuentes (menos de 100 registros):")
    print(valores_pocos)

# 1.6.9 Variable 'l2'

col = 'l2'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Frecuencias de valores
print("\nFrecuencia de valores:")
print(df[col].value_counts(dropna=False))

# Valores poco frecuentes (menos de 100 registros)
valores_pocos = df[col].value_counts(dropna=False)[df[col].value_counts(dropna=False) < 100]
if not valores_pocos.empty:
    print("\nValores poco frecuentes (menos de 100 registros):")
    print(valores_pocos)

# 1.6.10 Variable 'l3'

col = 'l3'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Frecuencias de valores
print("\nFrecuencia de valores:")
print(df[col].value_counts(dropna=False))

# Valores poco frecuentes (menos de 100 registros)
valores_pocos = df[col].value_counts(dropna=False)[df[col].value_counts(dropna=False) < 100]
if not valores_pocos.empty:
    print("\nValores poco frecuentes (menos de 100 registros):")
```

```
    print(valores_pocos)


# 1.6.11 Variable 'l4'

col = 'l4'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Frecuencias de valores
print("\nFrecuencia de valores:")
print(df[col].value_counts(dropna=False))

# Valores poco frecuentes (menos de 100 registros)
valores_pocos = df[col].value_counts(dropna=False)[df[col].value_counts(dropna=False) < 100]
if not valores_pocos.empty:
    print("\nValores poco frecuentes (menos de 100 registros):")
    print(valores_pocos)

# 1.6.12 Variable 'l5'

col = 'l5'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Frecuencias de valores
print("\nFrecuencia de valores:")
print(df[col].value_counts(dropna=False))

# Valores poco frecuentes (menos de 100 registros)
valores_pocos = df[col].value_counts(dropna=False)[df[col].value_counts(dropna=False) < 100]
if not valores_pocos.empty:
    print("\nValores poco frecuentes (menos de 100 registros):")
    print(valores_pocos)

# 1.6.13 Variable 'l6'

col = 'l6'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")
```

```
# Frecuencias de valores
print("\nFrecuencia de valores:")
print(df[col].value_counts(dropna=False))

# 1.6.14 Variable 'rooms'

col = 'rooms'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Resumen estadístico con percentiles
desc = df[col].describe(percentiles=[0.01,0.05,0.25,0.5,0.75,0.95,0.99])
print("\nResumen estadístico:")
print(desc)

# Valores mínimos y máximos observados
print("\nMínimos observados:")
print(df[col].nsmallest(10).to_list())
print("Máximos observados:")
print(df[col].nlargest(10).to_list())

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

plt.figure(figsize=(8,4))
sns.histplot(df[col].dropna(), bins=50, kde=False)
plt.title("Distribución de rooms")
plt.xlabel("Cantidad de ambientes")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()

# Boxplot para detección visual de outliers
plt.figure(figsize=(9,1.8))
sns.boxplot(x=df[col])
plt.title("Boxplot de rooms")
plt.grid(True)
plt.show()

# 1.6.15 Variable 'bedrooms'

col = 'bedrooms'

print(f"Variable: {col}")
```

```
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Resumen estadístico con percentiles
desc = df[col].describe(percentiles=[0.01,0.05,0.25,0.5,0.75,0.95,0.99])
print("\nResumen estadístico:")
print(desc)

# Valores mínimos y máximos observados
print("\nMínimos observados:")
print(df[col].nsmallest(10).to_list())
print("Máximos observados:")
print(df[col].nlargest(10).to_list())

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

plt.figure(figsize=(8,4))
sns.histplot(df[col].dropna(), bins=50, kde=False)
plt.title("Distribución de bedrooms")
plt.xlabel("Cantidad de dormitorios")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()

# Boxplot para detección visual de outliers
plt.figure(figsize=(9,1.8))
sns.boxplot(x=df[col])
plt.title("Boxplot de bedrooms")
plt.grid(True)
plt.show()


# 1.6.16 Variable 'bathrooms'

col = 'bathrooms'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Resumen estadístico con percentiles
desc = df[col].describe(percentiles=[0.01,0.05,0.25,0.5,0.75,0.95,0.99])
print("\nResumen estadístico:")
print(desc)
```

```
# Valores mínimos y máximos observados
print("\nMínimos observados:")
print(df[col].nsmallest(10).to_list())
print("Máximos observados:")
print(df[col].nlargest(10).to_list())

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

plt.figure(figsize=(8,4))
sns.histplot(df[col].dropna(), bins=50, kde=False)
plt.title("Distribución de bathrooms")
plt.xlabel("Cantidad de baños")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()

# Boxplot para detección visual de outliers
plt.figure(figsize=(9,1.8))
sns.boxplot(x=df[col])
plt.title("Boxplot de bathrooms")
plt.grid(True)
plt.show()

# 1.6.17 Variable 'surface_total'

col = 'surface_total'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Resumen estadístico con percentiles
desc = df[col].describe(percentiles=[0.01,0.05,0.25,0.5,0.75,0.95,0.99])
print("\nResumen estadístico:")
print(desc)

# Valores mínimos y máximos observados
print("\nMínimos observados:")
print(df[col].nsmallest(10).to_list())
print("Máximos observados:")
print(df[col].nlargest(10).to_list())

# Histograma
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.figure(figsize=(8,4))
sns.histplot(df[col].dropna(), bins=50, kde=False)
plt.title("Distribución de surface_total")
plt.xlabel("Superficie total (m²)")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()

# Boxplot para detección visual de outliers
plt.figure(figsize=(9,1.8))
sns.boxplot(x=df[col])
plt.title("Boxplot de surface_total")
plt.grid(True)
plt.show()

# 1.6.18 Variable 'surface_covered'

col = 'surface_covered'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Resumen estadístico con percentiles
desc = df[col].describe(percentiles=[0.01,0.05,0.25,0.5,0.75,0.95,0.99])
print("\nResumen estadístico:")
print(desc)

# Valores mínimos y máximos observados
print("\nMínimos observados:")
print(df[col].nsmallest(10).to_list())
print("Máximos observados:")
print(df[col].nlargest(10).to_list())

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

plt.figure(figsize=(8,4))
sns.histplot(df[col].dropna(), bins=50, kde=False)
plt.title("Distribución de surface_covered")
plt.xlabel("Superficie cubierta (m²)")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()

# Boxplot para detección visual de outliers
plt.figure(figsize=(9,1.8))
```

```
sns.boxplot(x=df[col])
plt.title("Boxplot de surface_covered")
plt.grid(True)
plt.show()

# 1.6.19 Variable 'currency'

col = 'currency'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Frecuencia de valores
freq = df[col].value_counts(dropna=False)
print("\nFrecuencia de valores:")
print(freq)

# Valores poco frecuentes (menos de 100 registros)
print("\nValores poco frecuentes (menos de 100 registros):")
print(freq[freq < 100])

# 1.6.20 Variable 'price_period'

col = 'price_period'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Frecuencia de valores
freq = df[col].value_counts(dropna=False)
print("\nFrecuencia de valores:")
print(freq)

# Valores poco frecuentes (menos de 100 registros)
print("\nValores poco frecuentes (menos de 100 registros):")
print(freq[freq < 100])

# 1.6.21 Variable 'title'

col = 'title'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")
```

```
# Mostrar ejemplos de títulos
print("\nEjemplos de títulos:")
print(df[col].dropna().sample(10, random_state=42))

# Detección de valores duplicados en títulos
duplicados = df[df.duplicated(subset=[col], keep=False)][col]
print(f"\nCantidad de títulos repetidos: {duplicados.nunique()} (sobre {len(duplicados)}
registros)")

# Top 20 títulos más repetidos
titulos_frecuentes = (
    df['title']
    .value_counts()
    .head(20)
)

print("Títulos más repetidos:\n")
print(titulos_frecuentes)

# 1.6.22 Variable 'description'

col = 'description'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Mostrar ejemplos de descripciones
print("\nEjemplos de descripciones:")
print(df[col].dropna().sample(5, random_state=42))

# Conteo de descripciones duplicadas
duplicadas = df[df.duplicated(subset=[col], keep=False)][col]
print(f"\nCantidad de descripciones repetidas: {duplicadas.nunique()} (sobre {len(duplicadas)}
registros)")

# Top descripciones más repetidas
desc_frecuentes = df[col].value_counts().head(20)
print("\nDescripciones más repetidas:")
print(desc_frecuentes)

# Exportar a Excel las descripciones repetidas para análisis posterior
duplicadas_df = df[df[col].isin(desc_frecuentes.index)]
duplicadas_df.to_excel("descripciones_repetidas.xlsx", index=False)
print("Archivo 'descripciones_repetidas.xlsx' exportado con éxito.")

# 1.6.23 Variable property_type
```

```
print("Variable: property_type")
print(f"Tipo de dato: {df['property_type'].dtype}")
print(f"Nulos: {df['property_type'].isna().sum()}
({df['property_type'].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df['property_type'].nunique(dropna=False)}\n")

# Frecuencia de valores
print("Frecuencia de valores:")
print(df['property_type'].value_counts(dropna=False))

# Valores poco frecuentes (menos de 100 registros)
poco_frecuentes_ptype = df['property_type'].value_counts()
poco_frecuentes_ptype = poco_frecuentes_ptype[poco_frecuentes_ptype < 100]
if not poco_frecuentes_ptype.empty:
    print("\nValores poco frecuentes (menos de 100 registros):")
    print(poco_frecuentes_ptype)

# Ejemplos de valores
print("\nEjemplos de valores:")
print(df['property_type'].dropna().unique()[:10])

# 1.6.24 Variable operation_type

col = 'operation_type'

print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()} ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}\n")

# Frecuencia de valores
print("Frecuencia de valores:")
print(df[col].value_counts(dropna=False))

# Valores poco frecuentes (menos de 100 registros)
poco_frecuentes = df[col].value_counts()
poco_frecuentes = poco_frecuentes[poco_frecuentes < 100]
if not poco_frecuentes.empty:
    print("\nValores poco frecuentes (menos de 100 registros):")
    print(poco_frecuentes)

# Ejemplos de valores
print("\nEjemplos de valores:")
print(df[col].dropna().unique()[:10])

# 1.6.25 Variable price

col = 'price'
```

```
print(f"Variable: {col}")
print(f"Tipo de dato: {df[col].dtype}")
print(f"Nulos: {df[col].isna().sum()}  ({df[col].isna().mean()*100:.2f}%)")
print(f"Valores únicos (incluye NaN): {df[col].nunique(dropna=False)}")

# Resumen estadístico con percentiles informativos
desc = df[col].describe(percentiles=[0.01,0.05,0.25,0.5,0.75,0.95,0.99])
print("\nResumen estadístico:")
print(desc)

# Mínimos y máximos observados (sin NaN)
print("\nMínimos observados:")
print(df[col].nsmallest(10).to_list())
print("Máximos observados:")
print(df[col].nlargest(10).to_list())

# Distribución por histograma
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,4))
sns.histplot(df[col].dropna(), bins=60, kde=False)
plt.title("Distribución de price (moneda original)")
plt.xlabel("price")
plt.ylabel("Frecuencia")
plt.grid(True)
plt.show()

# Boxplot para outliers
plt.figure(figsize=(9,1.8))
sns.boxplot(x=df[col])
plt.title("Boxplot de price")
plt.grid(True)
plt.show()

# 1.7 Chequeo de consistencia start date y end date (diferencia)

# Conversión temporal a datetime para hacer la comparación
df_temp = df.copy()
df_temp['start_date_dt'] = pd.to_datetime(df_temp['start_date'], errors='coerce')
df_temp['end_date_dt'] = pd.to_datetime(df_temp['end_date'], errors='coerce')

# Detección de casos en los que start_date sea mayor que end_date y end_date no sea 9999-12-31
errores_fecha = df_temp[
    (df_temp['start_date_dt'] > df_temp['end_date_dt']) &
    (df_temp['end_date'] != '9999-12-31')
]
```

```
print(f"Cantidad de registros con inconsistencia temporal: {len(errores_fecha)}")
display(errores_fecha.head())

# 1.8 Estadísticas descriptivas generales (sólo para variables numéricas)
df.describe()

# 1.9 Distribución por rangos para variables numéricas

# Filtrar columnas numéricas con al menos 2 valores distintos no nulos
numeric_cols = [
    col for col in df.select_dtypes(include=['number']).columns
    if df[col].dropna().nunique() > 1
]

# Aplicar pd.cut() sólo a columnas con variabilidad
for col in numeric_cols:
    try:
        print(f"\n▶ Distribución de '{col}':")
        print(pd.cut(df[col], bins=10).value_counts())
    except Exception as e:
        print(f"No se pudo procesar '{col}': {e}")

# 1.10 Histogramas de variables clave

import seaborn as sns
import matplotlib.pyplot as plt

columnas_numericas = ['price', 'surface_total', 'surface_covered', 'rooms', 'bathrooms']

for col in columnas_numericas:
    if col in df.columns and df[col].dropna().nunique() > 1:
        plt.figure(figsize=(8, 4))
        sns.histplot(df[col].dropna(), kde=True, bins=50)
        plt.title(f"Distribución de {col}")
        plt.xlabel(col)
        plt.ylabel("Frecuencia")
        plt.grid(True)
        plt.show()

# 1.11 Outliers según z-score

from scipy.stats import zscore
import numpy as np

# Creo subconjunto sin nulos para columnas válidas
columnas_validas = [col for col in columnas_numericas if df[col].dropna().nunique() > 1]
df_z = df[columnas_validas].dropna()

# Calculo z-score absoluto
```

```
z_scores = np.abs(zscore(df_z))

# Cuento valores con z > 3 (potenciales outliers)
outliers = (z_scores > 3).sum(axis=0)
print("\n▶ Cantidad de outliers detectados (z-score > 3):")
print(pd.Series(outliers, index=df_z.columns))

# 1.12 Asimetría y curtosis

print("\n▶ Asimetría de variables numéricas:")
print(df.skew(numeric_only=True))

print("\n▶ Curtosis de variables numéricas:")
print(df.kurtosis(numeric_only=True))

# 1.13 Correlación entre variables numéricas (excluyendo 'id')

import seaborn as sns
import matplotlib.pyplot as plt

# selecciono columnas numéricas
numeric_cols = df.select_dtypes(include=['number']).columns.tolist()

# excluyo la columna 'id' solo para este análisis
numeric_cols = [c for c in numeric_cols if c != "id"]

# calculo la matriz de correlación
corr_matrix = df[numeric_cols].corr()

print("Matriz de correlación (sin 'id'):")
print(corr_matrix)

# visualización con mapa de calor
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", center=0)
plt.title("Matriz de correlación (sin 'id')")
plt.show()

## 2. Preparación y limpieza de datos

# 2.1 Intercambio latitud y longitud

df["lat"], df["lon"] = df["lon"], df["lat"]

print("\n▶ Coordenadas luego del intercambio:")
display(df[["lat", "lon"]].head())

print("\n▶ Rango esperado para coordenadas de CABA:")
print("Latitud:", df["lat"].min(), "→", df["lat"].max())
```

```
print("Longitud:", df["lon"].min(), "→", df["lon"].max())
# 2.2 Filtrado l1 = país Argentina

dftp = df[df['l1'] == 'Argentina']

print("Cantidad de registros en Argentina:", len(dftp))
dftp[['l1','l2']].head()

# 2.3 Filtrado l2 = Capital Federal

dftp = dftp[dftp['l2'] == 'Capital Federal']

print("Cantidad de registros en Capital Federal:", len(dftp))
dftp.head()

# 2.4 Filtrado: cantidad de ambientes, tipo de propiedad y operación venta

filtro = (dftp['rooms'] >= 1) & \
         (dftp['rooms'] <= 3) & \
         (dftp['property_type'] == 'Departamento') & \
         (dftp['operation_type'] == 'Venta')

dftp1 = dftp[filtro]

print("Cantidad de registros tras aplicar filtro de ambientes, tipo y operación:", len(dftp1))
dftp1.head()

# guardado intermedio
dftp1.to_csv('base_tp.csv', index=False)

dftp1.to_csv('base_tp.csv', index=False)

# 2.5 Filtrado: fechas parseadas y creación de variable de vigencia

import pandas as pd
import numpy as np
import re

for c in ["start_date","end_date","created_on"]:
    dftp1[c] = pd.to_datetime(dftp1[c], errors="coerce")

dftp1["activo"] = dftp1["end_date"].dt.date.astype(str).eq("9999-12-31")
dftp1.loc[dftp1["activo"], "end_date"] = pd.NaT

dftp1[["start_date","end_date","created_on","activo"]].head()

# 2.6 Filtrado: validación de coordenadas con bounding box de CABA

LAT_MIN, LAT_MAX = -34.73, -34.49
```

```
LON_MIN, LON_MAX = -58.55, -58.32

dftp1 = dftp1[
    dftp1["lat"].between(LAT_MIN, LAT_MAX) &
    dftp1["lon"].between(LON_MIN, LON_MAX)
]

print("Registros dentro de CABA (bbox):", len(dftp1))

# Validación visual: comparación de lat/lon corregido vs invertido

import folium
from IPython.display import HTML, display

# Subconjunto con coordenadas válidas
tmp = df.dropna(subset=['lat', 'lon'])[['lat', 'lon']].copy()

# Muestreo para no sobrecargar el mapa
N = min(3000, len(tmp))
tmp_sample = tmp.sample(N, random_state=42).reset_index(drop=True)

# Mapa A: coordenadas corregidas (lat, lon)
m_corrected = folium.Map(
    location=[tmp_sample['lat'].mean(), tmp_sample['lon'].mean()],
    zoom_start=10,
    control_scale=True
)
for lat, lon in zip(tmp_sample['lat'], tmp_sample['lon']):
    folium.CircleMarker([lat, lon], radius=2, fill=True, fill_opacity=0.6).add_to(m_corrected)

# Mapa B: simula coordenadas sin corregir (lon, lat)
m_invert = folium.Map(
    location=[tmp_sample['lon'].mean(), tmp_sample['lat'].mean()],
    zoom_start=10,
    control_scale=True
)
for lat, lon in zip(tmp_sample['lat'], tmp_sample['lon']):
    folium.CircleMarker([lon, lat], radius=2, fill=True, fill_opacity=0.6).add_to(m_invert)

# Mostrar ambos mapas lado a lado
html_corrected = m_corrected._repr_html_()
html_invert = m_invert._repr_html_()

display(HTML(f"""
<div style="display:flex; gap:16px;">
  <div style="flex:1; border:1px solid #ddd; border-radius:8px; overflow:hidden;">
    <div style="padding:8px; font-weight:600;">Mapa A · coordenadas corregidas (lat, lon)</div>
    {html_corrected}
  </div>
```

```
  <div style="flex:1; border:1px solid #ddd; border-radius:8px; overflow:hidden;">
    <div style="padding:8px; font-weight:600;">Mapa B · coordenadas invertidas (lon, lat)</div>
    {html_invert}
  </div>
</div>
"""))

# 2.7 Limpieza: estandarización de texto para títulos

def norm_txt(s):
    if pd.isna(s):
        return s
    s = s.lower()
    s = re.sub(r"\s+", " ", s)
    s = re.sub(r"[^a-z0-9áéíóúñü° ]", " ", s)
    return s.strip()

dftp1["title_norm"] = dftp1["title"].astype(str).map(norm_txt)
dftp1[["title","title_norm"]].head()

# 2.8 Limpieza: corrección de superficies, baños, dormitorios y precio

dftp1.loc[(dftp1["surface_total"]<=12) | (dftp1["surface_total"]>=300), "surface_total"] =
np.nan
dftp1.loc[(dftp1["surface_covered"]<0) | (dftp1["surface_covered"]>dftp1["surface_total"]),
"surface_covered"] = np.nan
dftp1.loc[~dftp1["bathrooms"].between(1,5, inclusive="both"), "bathrooms"] = np.nan
dftp1.loc[~dftp1["bedrooms"].between(0,5, inclusive="both"), "bedrooms"] = np.nan
dftp1.loc[~dftp1["price"].between(1, 2_000_000, inclusive="both"), "price"] = np.nan

dftp1[["surface_total","surface_covered","bathrooms","bedrooms","price"]].describe()

# 2.9 Limpieza: unificación de moneda y conversión a USD con tipo de cambio histórico

import pandas as pd
import numpy as np
from google.colab import files

uploaded = files.upload()

filename = list(uploaded.keys())[0]

# leer según extensión
if filename.endswith(".xlsx"):
    fx = pd.read_excel(filename)
else:
    fx = pd.read_csv(filename)

fx["Fecha"] = pd.to_datetime(fx["Fecha"], errors="coerce")
```

```
fx["Valor"] = pd.to_numeric(fx["Valor"], errors="coerce")
fx = fx.sort_values("Fecha").dropna(subset=["Fecha"]).reset_index(drop=True)
fx["Valor"] = fx["Valor"].ffill()

# preparar dataset principal
dftp1["currency"] = dftp1["currency"].str.upper().str.strip()
dftp1["fecha_tc"] = pd.to_datetime(
    dftp1["start_date"].fillna(dftp1["created_on"]),
    errors="coerce"
)
dftp1 = dftp1.sort_values("fecha_tc")

# unir con merge_asof
dftp1 = pd.merge_asof(
    dftp1, fx[["Fecha","Valor"]].sort_values("Fecha"),
    left_on="fecha_tc", right_on="Fecha", direction="backward"
)

# cálculo de price_usd
def compute_price_usd(row):
    p = row.get("price", np.nan)
    c = row.get("currency", None)
    r = row.get("Valor", np.nan)
    if pd.isna(p) or pd.isna(c) or pd.isna(r):
        return np.nan
    if c == "USD":
        return p
    if c == "ARS":
        return np.nan if r == 0 else p / r  # dividir porque Valor = ARS por USD
    return np.nan

dftp1["price_usd"] = dftp1.apply(compute_price_usd, axis=1)

# 2.10 Validación: análisis de la nueva columna 'price_usd'

# primeras filas para revisar
dftp1[["price","currency","Valor","price_usd"]].head()

# estadísticos descriptivos
print("Estadísticos descriptivos de price_usd:")
print(dftp1["price_usd"].describe())

# cantidad y porcentaje de nulos
nulos_price_usd = dftp1["price_usd"].isna().sum()
pct_nulos = (nulos_price_usd / len(dftp1)) * 100
print(f"\nCantidad de nulos en price_usd: {nulos_price_usd} ({pct_nulos:.2f}%)")

# histograma en escala lineal
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(8,4))
dftp1["price_usd"].dropna().hist(bins=50)
plt.xlabel("Precio en USD")
plt.ylabel("Frecuencia")
plt.title("Distribución de price_usd (lineal)")
plt.show()

# histograma en escala logarítmica
plt.figure(figsize=(8,4))
dftp1["price_usd"].dropna().apply(np.log1p).hist(bins=50)
plt.xlabel("log(price_usd + 1)")
plt.ylabel("Frecuencia")
plt.title("Distribución log-transformada de price_usd")
plt.show()

# valores más frecuentes
print("\nValores de price_usd más frecuentes:")
print(dftp1["price_usd"].value_counts().head(10))

# 2.11 Limpieza: eliminación de registros con price_usd nulo

antes = len(dftp1)
dftp1 = dftp1[dftp1["price_usd"].notna()].copy()
despues = len(dftp1)

print(f"Registros eliminados por price_usd nulo: {antes - despues}")
print(f"Cantidad final de registros: {despues}")

# 2.12 Limpieza: creación de ratio superficie cubierta / superficie total

# cálculo del ratio
dftp1["ratio_sc"] = dftp1["surface_covered"] / dftp1["surface_total"]

# filtrado de ratios imposibles
dftp1.loc[~dftp1["ratio_sc"].between(0.3, 1.0, inclusive="both"), "ratio_sc"] = np.nan

# chequeo de nulos
print("Cantidad de nulos en ratio_sc:", dftp1["ratio_sc"].isna().sum())

# primeras filas
dftp1[["surface_total","surface_covered","ratio_sc"]].head()

# 2.13 Limpieza: imputación de valores faltantes en 'bedrooms'

# criterio: usar superficie cubierta o superficie total como referencia
# reglas aproximadas:
# - si surface_covered < 40 → 0 dormitorios
# - si surface_covered < 65 → 1 dormitorio
# - si surface_covered < 90 → 2 dormitorios
```

```
# - si surface_covered < 120 → 3 dormitorios
# - si surface_covered < 160 → 4 dormitorios
# - si es mayor → 5 dormitorios

# función de imputación con surface_covered
def imputar_bedrooms_sc(row):
    if pd.isna(row['bedrooms']):
        sc = row['surface_covered']
        if pd.isna(sc):
            return row['bedrooms']  # no se puede imputar
        elif sc < 40:
            return 0
        elif sc < 65:
            return 1
        elif sc < 90:
            return 2
        elif sc < 120:
            return 3
        elif sc < 160:
            return 4
        else:
            return 5
    else:
        return row['bedrooms']

dftp1['bedrooms'] = dftp1.apply(imputar_bedrooms_sc, axis=1)

# para los que sigan nulos, usar surface_total como referencia
def imputar_bedrooms_st(row):
    if pd.isna(row['bedrooms']):
        st = row['surface_total']
        if pd.isna(st):
            return row['bedrooms']
        elif st < 40:
            return 0
        elif st < 65:
            return 1
        elif st < 90:
            return 2
        elif st < 120:
            return 3
        elif st < 160:
            return 4
        else:
            return 5
    else:
        return row['bedrooms']

dftp1['bedrooms'] = dftp1.apply(imputar_bedrooms_st, axis=1)
```

```
# para los que aún queden en NaN, imputar usando rooms (si rooms>=1 entonces bedrooms=1)
dftp1['bedrooms'] = dftp1.apply(
    lambda row: 1 if pd.isna(row['bedrooms']) and row['rooms'] >= 1 else row['bedrooms'],
    axis=1
)

# redondear y convertir a entero manteniendo nulos como <NA>
dftp1['bedrooms'] = dftp1.round(decimals=0)['bedrooms'].astype('Int64')

# ver cantidad de nulos restantes en bedrooms
print("Cantidad de nulos en bedrooms:", dftp1['bedrooms'].isna().sum())
dftp1['bedrooms'].value_counts().head()

# 2.14 Limpieza: imputación de 'bathrooms' con regla simple por bedrooms

# regla:
# - bedrooms 0–2 -> 1 baño
# - bedrooms 3–4 -> 2 baños
# - bedrooms >=5 -> 3 baños
def imputar_bathrooms(row):
    b = row["bathrooms"]
    bd = row["bedrooms"]
    if pd.isna(b):
        if bd <= 2:   return 1
        if bd <= 4:   return 2
        return 3
    return b

dftp1["bathrooms"] = dftp1.apply(imputar_bathrooms, axis=1)
dftp1.loc[~dftp1["bathrooms"].between(1,5, inclusive="both"), "bathrooms"] = np.nan
dftp1["bathrooms"] = dftp1["bathrooms"].fillna(1).round().astype("Int64")

print("nulos en bathrooms:", dftp1["bathrooms"].isna().sum())
print(dftp1["bathrooms"].value_counts().sort_index())


# 2.15 Validación: consistencia básica entre bedrooms, bathrooms y surfaces

# casos con bedrooms >=3 y bathrooms == 1 (posible subestimación)
flag_bajo_banio = dftp1[(dftp1["bedrooms"] >= 3) & (dftp1["bathrooms"] == 1)].shape[0]
print("registros con bedrooms>=3 y bathrooms==1:", flag_bajo_banio)

# casos extremos: bedrooms == 0 y bathrooms >=3
flag_alto_banio = dftp1[(dftp1["bedrooms"] == 0) & (dftp1["bathrooms"] >= 3)].shape[0]
print("registros con bedrooms==0 y bathrooms>=3:", flag_alto_banio)

# chequeo de superficies vs dormitorios
print("superficie cubierta promedio por bedrooms:")
```

```
print(dftp1.groupby("bedrooms", dropna=False)["surface_covered"].mean().round(1))

# histograma de bathrooms
import matplotlib.pyplot as plt
plt.figure(figsize=(6,4))
dftp1["bathrooms"].hist(bins=5)
plt.xlabel("bathrooms")
plt.ylabel("frecuencia")
plt.title("Distribución de bathrooms")
plt.show()

# histograma de bedrooms
plt.figure(figsize=(6,4))
dftp1["bedrooms"].hist(bins=6)
plt.xlabel("bedrooms")
plt.ylabel("frecuencia")
plt.title("Distribución de bedrooms")
plt.show()

# 2.16 Limpieza: reglas finales suaves de consistencia

# si bedrooms >= 3 y bathrooms imputado fue 1, subir a 2 (suave)
mask_corr = (dftp1["bedrooms"] >= 3) & (dftp1["bathrooms"] == 1)
dftp1.loc[mask_corr, "bathrooms"] = 2

# recortar posibles valores fuera de rango por errores previos
dftp1.loc[~dftp1["bathrooms"].between(1,5, inclusive="both"), "bathrooms"] = 1
dftp1["bathrooms"] = dftp1["bathrooms"].astype("Int64")

# 2.17 Limpieza: eliminación de columnas con demasiados nulos (umbral 80%)

umbral = 0.80
null_ratio = dftp1.isna().mean().sort_values(ascending=False)

# columnas críticas que NO se eliminan aunque superen el umbral
protegidas = {
    "id","start_date","end_date","created_on","lat","lon",
    "l1","l2","rooms","price","currency","property_type"
}

cols_drop = [c for c, r in null_ratio.items() if (r >= umbral and c not in protegidas)]
print("Columnas candidatas a eliminar por nulos ≥ 80%:", cols_drop)

dftp1 = dftp1.drop(columns=cols_drop, errors="ignore")
print("Columnas restantes:", len(dftp1.columns))

# 2.18 Limpieza: eliminación de columnas constantes (misma categoría/valor en todo el dataset)

# columnas críticas que no se eliminan aunque sean constantes
```

```
protegidas_const = {
    "id","start_date","end_date","created_on","lat","lon",
    "l1","l2","property_type"
}

# detecta columnas con un único valor NO nulo (o todas NaN)
const_cols = [c for c in dftp1.columns if dftp1[c].nunique(dropna=True) <= 1]

# excluye protegidas
const_cols = [c for c in const_cols if c not in protegidas_const]

print("Columnas constantes a eliminar:", const_cols)

dftp1 = dftp1.drop(columns=const_cols, errors="ignore")
print("Columnas restantes:", dftp1.shape[1])

print("Columnas finales en el dataset (total = {}):".format(dftp1.shape[1]))
print(dftp1.columns.tolist())

# 2.19 Limpieza: eliminación de columnas auxiliares del tipo de cambio
drop_aux = ["fecha_tc", "Fecha", "Valor"]
dftp1 = dftp1.drop(columns=drop_aux, errors="ignore")

print("Columnas restantes tras eliminar auxiliares:", dftp1.shape[1])
print(dftp1.columns.tolist())

(dftp['property_type'] == 'Departamento')

print("Valores únicos en property_type:")
print(dftp1["property_type"].value_counts(dropna=False))

if dftp1["property_type"].nunique(dropna=True) == 1:
    dftp1 = dftp1.drop(columns=["property_type"])
    print("Se eliminó property_type por ser constante.")

# 2.20 Limpieza: corrección de superficies inconsistentes y actualización de ratio_sc

# si surface_covered > surface_total, se intercambian (caso de carga invertida)
mask_swap = (dftp1["surface_covered"].notna() & dftp1["surface_total"].notna() &
             (dftp1["surface_covered"] > dftp1["surface_total"]))

print("Registros con surface_covered > surface_total (antes):", mask_swap.sum())

tmp = dftp1.loc[mask_swap, "surface_covered"].copy()
dftp1.loc[mask_swap, "surface_covered"] = dftp1.loc[mask_swap, "surface_total"].values
dftp1.loc[mask_swap, "surface_total"]   = tmp.values

# recalcular ratio_sc y filtrar valores fuera de rango
dftp1["ratio_sc"] = dftp1["surface_covered"] / dftp1["surface_total"]
```

```
dftp1.loc[~dftp1["ratio_sc"].between(0.3, 1.0, inclusive="both"), "ratio_sc"] = np.nan

print("Nulos en ratio_sc (después de corrección):", dftp1["ratio_sc"].isna().sum())

# 2.21 Limpieza: control de precios en USD (cero/negativos y extremos)

antes = len(dftp1)
dftp1 = dftp1[
    dftp1["price_usd"].notna() &
    (dftp1["price_usd"] > 0) &
    (dftp1["price_usd"] <= 2_000_000)
].copy()
despues = len(dftp1)

print(f"Filas eliminadas por price_usd inválido: {antes - despues}")
print(dftp1["price_usd"].describe())

# 2.22 Limpieza: coordenadas y ubicaciones inconsistentes

# eliminación de lat/lon iguales a 0 o nulos (ya hay bbox aplicado, esto es residual)
antes = len(dftp1)
dftp1 = dftp1[
    dftp1["lat"].notna() & dftp1["lon"].notna() &
    ~((dftp1["lat"] == 0) & (dftp1["lon"] == 0))
].copy()
print("Filas eliminadas por coordenadas nulas o (0,0):", antes - len(dftp1))

# opción suave: si existe l3 pero l4 falta, no se elimina; solo informar
faltantes_l4 = dftp1["l4"].isna().sum() if "l4" in dftp1.columns else 0
print("Registros con l4 faltante (informativo):", faltantes_l4)

# 2.23 Limpieza: incongruencias de superficies (covered > total)

# - Si la diferencia es chica (≤20%), se asume swap y se intercambian.
# - Si es grande, se marca surface_covered como NaN.
import numpy as np

# detectar casos covered > total
mask_incoh = (dftp1["surface_covered"].notna() & dftp1["surface_total"].notna() &
              (dftp1["surface_covered"] > dftp1["surface_total"]))
print("Registros con surface_covered > surface_total (antes):", mask_incoh.sum())

df_incoh = dftp1.loc[mask_incoh, ["surface_total","surface_covered"]].copy()

# diferencia relativa respecto de covered
mask_swap = (df_incoh["surface_total"] <= df_incoh["surface_covered"] * 1.2)
mask_big  = ~mask_swap

# SWAP donde parece inversión
```

```
tmp_total = dftp1.loc[df_incoh.index[mask_swap], "surface_total"].copy()
dftp1.loc[df_incoh.index[mask_swap], "surface_total"]  = dftp1.loc[df_incoh.index[mask_swap],
"surface_covered"].values
dftp1.loc[df_incoh.index[mask_swap], "surface_covered"] = tmp_total.values

# BIG diff → marcar covered como NaN
dftp1.loc[df_incoh.index[mask_big], "surface_covered"] = np.nan

print("Intercambios realizados (swap):", mask_swap.sum())
print("Marcados como NaN (diff grande):", mask_big.sum())

# recalcular ratio_sc y limpiar fuera de rango
dftp1["ratio_sc"] = dftp1["surface_covered"] / dftp1["surface_total"]
dftp1.loc[~dftp1["ratio_sc"].between(0.3, 1.0, inclusive="both"), "ratio_sc"] = np.nan
print("Nulos en ratio_sc tras corrección:", dftp1["ratio_sc"].isna().sum())

# 2.24 Limpieza: lógica 'bedrooms' vs 'rooms' y revisión de nulos/ceros clave

# bedrooms no puede exceder rooms-1 para rooms en {1,2,3} del recorte (ej.: monoambiente ->
bedrooms=0)
mask_bedrooms_bad = (
    ((dftp1["rooms"] == 1) & (dftp1["bedrooms"] > 0)) |
    ((dftp1["rooms"] == 2) & (dftp1["bedrooms"] > 1)) |
    ((dftp1["rooms"] == 3) & (dftp1["bedrooms"] > 2))
)
print("Registros con bedrooms > lógico:", mask_bedrooms_bad.sum())

# marcar como NaN para reimputar con tu regla previa
dftp1.loc[mask_bedrooms_bad, "bedrooms"] = np.nan

# reimputación rápida reutilizando tu criterio
def _imp_bed_sc(row):
    bd, sc = row["bedrooms"], row["surface_covered"]
    if pd.isna(bd) and pd.notna(sc):
        if sc < 40: return 0
        if sc < 65: return 1
        if sc < 90: return 2
        if sc < 120: return 3
        if sc < 160: return 4
        return 5
    return bd

def _imp_bed_st(row):
    bd, st = row["bedrooms"], row["surface_total"]
    if pd.isna(bd) and pd.notna(st):
        if st < 40: return 0
        if st < 65: return 1
        if st < 90: return 2
        if st < 120: return 3
```

```
        if st < 160: return 4
        return 5
    return bd

dftp1["bedrooms"] = dftp1.apply(_imp_bed_sc, axis=1)
dftp1["bedrooms"] = dftp1.apply(_imp_bed_st, axis=1)
dftp1["bedrooms"] = dftp1.apply(
    lambda r: 1 if pd.isna(r["bedrooms"]) and r["rooms"] >= 1 else r["bedrooms"], axis=1
).round().astype("Int64")

print("Nulos en bedrooms tras reimputar:", dftp1["bedrooms"].isna().sum())

# Revisión de price/superficies/baños: nulos y <=0
vars_check = ["price", "surface_total", "surface_covered", "bathrooms"]
resumen = []
for col in vars_check:
    nulos = dftp1[col].isna().sum()
    ceros = (dftp1[col] <= 0).sum() if col != "bathrooms" else (dftp1[col] <= 0).sum()
    total = len(dftp1)
    resumen.append({
        "Variable": col, "Nulos": nulos, "Ceros/<=0": ceros,
        "% Nulos": round(nulos/total*100, 2),
        "% Ceros/<=0": round(ceros/total*100, 2)
    })

import pandas as pd
tabla_resumen = pd.DataFrame(resumen)
print(tabla_resumen)

# Acciones: eliminar price <= 0; marcar <=0 en superficies/baños como NaN
before = len(dftp1)
dftp1 = dftp1[dftp1["price"] > 0].copy()
after = len(dftp1)
print(f"Filas eliminadas por price <= 0: {before - after}")

for col in ["surface_total","surface_covered","bathrooms"]:
    dftp1.loc[dftp1[col] <= 0, col] = np.nan

# Resumen corto post-acción
for col in ["surface_total","surface_covered","bathrooms"]:
    nulos = dftp1[col].isna().sum()
    ceros = (dftp1[col] <= 0).sum()
    print(f"{col}: {nulos} nulos; {ceros} <= 0")

# 2.25 Limpieza: estandarización de barrios (l3) y creación de variable 'barrio'

import pandas as pd
import numpy as np
import unicodedata
```

```
# --- Lista oficial de 48 barrios
OFFICIAL_BARRIOS = [
    "Agronomía","Almagro","Balvanera","Barracas","Belgrano","Boedo","Caballito","Chacarita",
    "Coghlan","Colegiales","Constitución","Flores","Floresta","La Boca","La Paternal","Liniers",
    "Mataderos","Monte Castro","Monserrat","Nueva Pompeya","Núñez","Palermo","Parque
Avellaneda",
    "Parque Chacabuco","Parque Chas","Parque Patricios","Puerto Madero","Recoleta","Retiro",
    "Saavedra","San Cristóbal","San Nicolás","San Telmo","Vélez Sarsfield","Versalles",
    "Villa Crespo","Villa del Parque","Villa Devoto","Villa General Mitre","Villa Lugano",
    "Villa Luro","Villa Ortúzar","Villa Pueyrredón","Villa Real","Villa Riachuelo",
    "Villa Santa Rita","Villa Soldati","Villa Urquiza"
]

# helpers de normalización (minúsculas, sin tildes, sin espacios extras)
def _strip_accents(s):
    if pd.isna(s): return np.nan
    s = str(s).strip().lower()
    s = "".join(ch for ch in unicodedata.normalize("NFKD", s) if not unicodedata.combining(ch))
    s = " ".join(s.split())
    return s

# mapa normalizado de oficial -> oficial
OFFICIAL_NORM_MAP = { _strip_accents(b): b for b in OFFICIAL_BARRIOS }

# --- Sinónimos/variantes comunes (lado izquierdo normalizado, valor = oficial correcto) ---
SYNONYMS = {
    # subzonas / alias frecuentes
    "barrio norte": "Recoleta",
    "las canitas": "Palermo",
    "canitas": "Palermo",
    "abasto": "Balvanera",
    "once": "Balvanera",
    "microcentro": "San Nicolás",
    "centro": "San Nicolás",
    "centro / microcentro": "San Nicolás",

    # ortografías/acentos
    "san cristobal": "San Cristóbal",
    "san nicolas": "San Nicolás",
    "montserrat": "Monserrat",
    "monserrat": "Monserrat",   # por las dudas
    "velez sarsfield": "Vélez Sarsfield",
    "velez sarfield": "Vélez Sarsfield",
    "villa ortuzar": "Villa Ortúzar",
    "villa ortuzar.": "Villa Ortúzar",
    "constitucion": "Constitución",
    "nunez": "Núñez",
    "parque chas.": "Parque Chas",
```

```
    # pompeya
    "pompeya": "Nueva Pompeya",
}

# función de estandarización
def estandarizar_barrio(x):
    if pd.isna(x):
        return np.nan
    key = _strip_accents(x)
    # si es un oficial exacto (normalizado)
    if key in OFFICIAL_NORM_MAP:
        return OFFICIAL_NORM_MAP[key]
    # si es un sinónimo
    if key in SYNONYMS:
        return SYNONYMS[key]
    # si trae "barrio" adelante, recorto e intento
    if key.startswith("barrio "):
        key2 = key.replace("barrio ","",1).strip()
        if key2 in OFFICIAL_NORM_MAP:
            return OFFICIAL_NORM_MAP[key2]
        if key2 in SYNONYMS:
            return SYNONYMS[key2]
    # si no matchea, devolvemos NaN (o "Desconocido", si preferís)
    return np.nan

# aplicar al dataset
dftp1["barrio"] = dftp1["l3"].apply(estandarizar_barrio)

print("Barrios no mapeados (muestran algunos ejemplos):")
print(dftp1[dftp1["barrio"].isna()]["l3"].dropna().value_counts().head(15))

print("\nTop 15 barrios (ya estandarizados):")
print(dftp1["barrio"].value_counts().head(15))

# 2.26 (Dummies de 'barrio' con umbral mínimo de frecuencia

# umbral para evitar dummies muy raras
UMBRAL_MIN = 50

freq = dftp1["barrio"].value_counts()
barrios_keep = freq[freq >= UMBRAL_MIN].index.tolist()

# mantengo NaN fuera (no genero dummy para NaN)
dftp1["barrio_f"] = dftp1["barrio"].where(dftp1["barrio"].isin(barrios_keep), other=np.nan)

barrio_dummies = pd.get_dummies(dftp1["barrio_f"], prefix="barrio", dummy_na=False)

print("Dummies creadas:", barrio_dummies.shape[1])
```

```
print(barrio_dummies.columns.tolist()[:10], "...")

# unir al dataset
dftp1 = pd.concat([dftp1, barrio_dummies], axis=1)

# dftp1.drop(columns=["l3"], inplace=True)  # <- descomentar si preferís borrar l3


# 2.27 Validación final y guardado del dataset limpio

# chequeo de nulos clave
chk = {
    "surface_total": dftp1["surface_total"].isna().sum(),
    "surface_covered": dftp1["surface_covered"].isna().sum(),
    "bedrooms": dftp1["bedrooms"].isna().sum() if "bedrooms" in dftp1.columns else "columna no
presente",
    "bathrooms": dftp1["bathrooms"].isna().sum() if "bathrooms" in dftp1.columns else "columna
no presente",
    "price_usd": dftp1["price_usd"].isna().sum(),
    "ratio_sc": dftp1["ratio_sc"].isna().sum()
}
print("Nulos en variables clave:", chk)

# columnas finales
cols_keep = [c for c in [
    "id","start_date","end_date","created_on","activo",
    "lat","lon","l1","l2","l3","l4",
    "rooms","bedrooms","bathrooms",
    "surface_total","surface_covered","ratio_sc",
    "currency","price","price_usd","price_period" if "price_period" in dftp1.columns else None,
    "property_type","operation_type","title","title_norm"
] if c in dftp1.columns]

df_final = dftp1[cols_keep].copy()
print("Filas finales:", len(df_final))
print("Columnas finales:", len(cols_keep))

out_path = "/content/caba_clean_final.csv"
df_final.to_csv(out_path, index=False)
print("Archivo guardado:", out_path)

## 3. Creación y evaluación de modelos

3.1 Random Forest Regressor:

# 3.1.1 Preparación básica para modelado

import numpy as np
import pandas as pd
```

```python
import sklearn as sk

# Tomo la base limpia
dfm = df_final.copy() if 'df_final' in globals() else dftp1.copy()

# Usar price_usd si está disponible; si no, caer a price
target_col = 'price_usd' if 'price_usd' in dfm.columns else 'price'

# Me quedo solo con columnas numéricas
df_num = dfm.select_dtypes(include=['float64','int64','int32','int16','int8','bool']).copy()

# Aseguro que el target exista y no tenga nulos
df_num = df_num[df_num[target_col].notna()].copy()

# Lleno NaN SOLO en features (no en el target)
X = df_num.drop(columns=[target_col]).fillna(0)
y = df_num[target_col].values

# Modelo y CV
n_estimators = 500
max_depth = 10
reg = sk.ensemble.RandomForestRegressor(
    n_estimators=n_estimators, max_depth=max_depth, n_jobs=-1, random_state=42
)

# MSE negativo como en el profe
mse_neg = sk.model_selection.cross_val_score(
    reg, X, y, scoring="neg_mean_squared_error", cv=5, n_jobs=-1
)

# Reporto igual que el profe y además en RMSE
mse = -1 * mse_neg
rmse = np.sqrt(mse)

print(f"MSE CV -- mean={mse.mean():.2f}, std={mse.std():.2f}")
print(f"RMSE CV -- mean={rmse.mean():.2f}, std={rmse.std():.2f}")

# 3.1.2 Evaluar con más métricas (MAE y R²)

from sklearn.model_selection import cross_val_predict
from sklearn.metrics import mean_absolute_error, r2_score

# Predicciones de CV
y_pred_cv = cross_val_predict(reg, X, y, cv=5, n_jobs=-1)

mae = mean_absolute_error(y, y_pred_cv)
r2  = r2_score(y, y_pred_cv)

print(f"MAE CV: {mae:.2f}")
```

```
print(f"R2  CV: {r2:.3f}")

# 3.1.3 Visualización de predicciones vs valores reales

import matplotlib.pyplot as plt

plt.figure(figsize=(6,6))
plt.scatter(y, y_pred_cv, alpha=0.3)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.xlabel("Precio real (USD)")
plt.ylabel("Precio predicho (USD)")
plt.title("Predicción vs Realidad (Validación Cruzada)")
plt.grid(True)
plt.show()

# 3.1.4 Importancia de las variables

# Entreno el modelo completo para obtener importancias
reg.fit(X, y)

importances = pd.Series(reg.feature_importances_, index=X.columns).sort_values(ascending=False)
print(importances.head(15))

# gráfico
plt.figure(figsize=(8,5))
importances.head(15).plot(kind="barh")
plt.title("Importancia de las principales variables (Random Forest)")
plt.xlabel("Importancia relativa")
plt.gca().invert_yaxis()
plt.show()

3.2 Decision Tree Regressor:

# 3.2.1 Preparación de datos (Decision Tree con 80/20)

import pandas as pd
import numpy as np
import sklearn as sk
from sklearn import tree
import matplotlib.pyplot as plt

# Partimos del dataset limpio
dfm = df_final.copy() if 'df_final' in globals() else dftp1.copy()

# Target = price_usd
target_col = "price_usd"

# Usar solo numéricas
df_num = dfm.select_dtypes(include=['float64','int64','int32','int16','int8','bool']).copy()
```

```
df_num = df_num[df_num[target_col].notna()]

X = df_num.drop(columns=[target_col])
y = df_num[target_col].values

# Split 80/20
X_train, X_test, y_train, y_test = sk.model_selection.train_test_split(
    X, y, test_size=0.2, random_state=42
)

# 3.2.2 Entrenamiento básico del árbol de regresión

from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Árbol simple con profundidad limitada
dt = DecisionTreeRegressor(max_depth=5, random_state=42)
dt.fit(X_train, y_train)

# Predicciones
y_pred = dt.predict(X_test)

# Métricas
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae  = mean_absolute_error(y_test, y_pred)
r2   = r2_score(y_test, y_pred)

print(f"RMSE test: {rmse:.2f}")
print(f"MAE  test: {mae:.2f}")
print(f"R²   test: {r2:.3f}")

# 3.3.3 Visualización del árbol entrenado (máx. 3 niveles para claridad)

plt.figure(figsize=(20,10))
tree.plot_tree(
    dt,
    feature_names=X.columns,
    filled=True,
    rounded=True,
    max_depth=3,    # muestro solo los primeros 3 niveles
    fontsize=10
)
plt.show()

# 3.2.4 Validación cruzada 10 folds (RMSE promedio)

from sklearn.model_selection import cross_val_score

dt_cv = DecisionTreeRegressor(max_depth=5, random_state=42)
```

```
mse_neg = cross_val_score(
    dt_cv, X, y, scoring="neg_mean_squared_error", cv=10, n_jobs=-1
)
mse = -mse_neg
rmse = np.sqrt(mse)

print(f"RMSE por fold: {rmse}")
print(f"RMSE promedio: {rmse.mean():.2f}, std: {rmse.std():.2f}")

# 3.2.5 Optimización de hiperparámetros con GridSearchCV

from sklearn.model_selection import GridSearchCV

param_grid = {
    "max_depth": [3, 5, 7, 10, None],
    "min_samples_split": [2, 5, 10, 20],
    "min_samples_leaf": [1, 5, 10, 20]
}

grid_dt = GridSearchCV(
    DecisionTreeRegressor(random_state=42),
    param_grid,
    scoring="neg_mean_squared_error",
    cv=10,
    n_jobs=-1,
    verbose=1
)

grid_dt.fit(X, y)

print("Mejores hiperparámetros:", grid_dt.best_params_)
rmse_best = np.sqrt(-grid_dt.best_score_)
print(f"Mejor RMSE (CV=10): {rmse_best:.2f}")

3.3 K-Nearest Neighbors Regressor:

# 3.3.1 Diagnóstico rapio

import numpy as np
import pandas as pd
import sklearn as sk

dfm = df_final.copy() if 'df_final' in globals() else dftp1.copy()
target_col = "price_usd"

# Solo numéricas
df_num = dfm.select_dtypes(include=['float64','int64','int32','int16','int8','bool']).copy()
```

```python
# Asegurar target válido
df_num = df_num[df_num[target_col].notna()].copy()

# Chequeos
print("Cols num:", len(df_num.columns))
print("NaNs totales en X(incl target):", df_num.isna().sum().sum())
print("NaNs por columna TOP:")
print(df_num.isna().sum().sort_values(ascending=False).head(10))

# 3.3.2 Imputación y normalización

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Separar X/y
X = df_num.drop(columns=[target_col])
y = df_num[target_col].astype(float).values  # asegurar float

# Pipeline: imputar medianas -> escalar 0-1 -> KNN
pipe_knn = Pipeline(steps=[
    ("imp",    SimpleImputer(strategy="median")),
    ("scale",  MinMaxScaler()),
    ("model",  KNeighborsRegressor(n_neighbors=5, weights="distance", p=2))
])

# Split 80/20
X_train, X_test, y_train, y_test = sk.model_selection.train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Fit + pred
pipe_knn.fit(X_train, y_train)
y_pred = pipe_knn.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae  = mean_absolute_error(y_test, y_pred)
r2   = r2_score(y_test, y_pred)

print(f"RMSE test: {rmse:.2f}")
print(f"MAE  test: {mae:.2f}")
print(f"R²   test: {r2:.3f}")

# 3.3.3 CV-10 + optimización

from sklearn.model_selection import GridSearchCV
```

```
param_grid = {
    "model__n_neighbors": [3,5,7,10,15,20,30],
    "model__weights": ["uniform","distance"],
    "model__p": [1,2]
}

grid_knn = GridSearchCV(
    estimator=Pipeline([
        ("imp",    SimpleImputer(strategy="median")),
        ("scale", MinMaxScaler()),
        ("model", KNeighborsRegressor())
    ]),
    param_grid=param_grid,
    scoring="neg_mean_squared_error",
    cv=10,
    n_jobs=-1,
    verbose=1
)

grid_knn.fit(X, y)
best_rmse = np.sqrt(-grid_knn.best_score_)
print("Mejores hiperparámetros:", grid_knn.best_params_)
print(f"Mejor RMSE (CV=10): {best_rmse:.2f}")


3.4 Gradient Boosting:

# 3.4.1 Preparación (numéricas + imputación)

import numpy as np
import pandas as pd
import sklearn as sk
from sklearn.impute import SimpleImputer

dfm = df_final.copy() if 'df_final' in globals() else dftp1.copy()
target_col = "price_usd"

df_num = dfm.select_dtypes(include=['float64','int64','int32','int16','int8','bool']).copy()
df_num = df_num[df_num[target_col].notna()].copy()

X = df_num.drop(columns=[target_col])
y = df_num[target_col].astype(float).values

imp = SimpleImputer(strategy="median")
X_imp = imp.fit_transform(X)  # árboles no necesitan escalado

# 3.4.2 Baseline 80/20 con HistGradientBoosting

from sklearn.experimental import enable_hist_gradient_boosting  # noqa: F401
```

```
from sklearn.ensemble import HistGradientBoostingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

X_train, X_test, y_train, y_test = sk.model_selection.train_test_split(
    X_imp, y, test_size=0.2, random_state=42
)

hgb = HistGradientBoostingRegressor(
    max_depth=None,
    max_iter=300,
    learning_rate=0.06,
    l2_regularization=0.0,
    early_stopping=True,
    validation_fraction=0.1,
    random_state=42
)

hgb.fit(X_train, y_train)
pred = hgb.predict(X_test)

rmse = np.sqrt(mean_squared_error(y_test, pred))
mae  = mean_absolute_error(y_test, pred)
r2   = r2_score(y_test, pred)
print(f"[HGB 80/20] RMSE: {rmse:.2f} | MAE: {mae:.2f} | R²: {r2:.3f}")

# 3.4.3 CV-10 + RandomizedSearch para HistGradientBoosting

from sklearn.model_selection import RandomizedSearchCV

param_dist = {
    "learning_rate":   [0.03, 0.05, 0.06, 0.08, 0.1],
    "max_depth":       [None, 4, 6, 8, 10],
    "max_leaf_nodes":  [15, 31, 63, 127],
    "min_samples_leaf":[10, 20, 50, 100],
    "l2_regularization":[0.0, 1e-4, 1e-3, 1e-2]
}

hgb_base = HistGradientBoostingRegressor(
    max_iter=400,
    early_stopping=True,
    validation_fraction=0.1,
    random_state=42
)

rs_hgb = RandomizedSearchCV(
    estimator=hgb_base,
    param_distributions=param_dist,
    n_iter=20,
    scoring="neg_mean_squared_error",
```

```
    cv=10,
    n_jobs=-1,
    verbose=1,
    random_state=42
)

rs_hgb.fit(X_imp, y)
best_rmse = np.sqrt(-rs_hgb.best_score_)
print("Mejores hiperparámetros (HGB):", rs_hgb.best_params_)
print(f"[HGB CV=10] Mejor RMSE: {best_rmse:.2f}")

# 3.4.4 Importancia de atributos

best_hgb = rs_hgb.best_estimator_ if 'rs_hgb' in globals() else hgb
importances = getattr(best_hgb, "feature_importances_", None)

if importances is not None:
    imp = pd.Series(importances, index=X.columns).sort_values(ascending=False)
    print(imp.head(15))
else:

    from sklearn.inspection import permutation_importance
    r = permutation_importance(best_hgb, X_imp, y, n_repeats=5, random_state=42, n_jobs=-1)
    imp = pd.Series(r.importances_mean, index=X.columns).sort_values(ascending=False)
    print(imp.head(15))

3.5 MLPRegressor:

# 3.5.1 Preparación
import numpy as np, pandas as pd, sklearn as sk
from sklearn.model_selection import train_test_split

# tomo df_final si existe; si no, dftp1
dfm = df_final.copy() if 'df_final' in globals() else dftp1.copy()
target_col = "price_usd"

# quedarme con columnas numéricas
df_num = dfm.select_dtypes(include=['number','bool']).copy()

# remover filas con target nulo
df_num = df_num[df_num[target_col].notna()].copy()

# evitar fuga: si existe 'price' (original), quitarla de X
leaks = [c for c in ['price'] if c in df_num.columns and c != target_col]

X = df_num.drop(columns=[target_col] + leaks).astype('float32')
y = df_num[target_col].astype('float32').values

print("Shape X, y:", X.shape, y.shape)
```

```
print("Nulos en X:", int(X.isna().sum().sum()), "| Nulos en y:", int(np.isnan(y).sum()))

# 3.5.2 MLP baseline 80/20 (imputación + escalado + MLP) — rápido
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import warnings; warnings.filterwarnings("ignore")

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42
)

pipe_mlp = Pipeline(steps=[
    ("imp", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler()),
    ("model", MLPRegressor(
        hidden_layer_sizes=(64, 32),
        activation="relu",
        solver="adam",
        alpha=1e-4,
        learning_rate="adaptive",
        learning_rate_init=1e-3,
        early_stopping=True,
        validation_fraction=0.10,
        n_iter_no_change=10,
        max_iter=200,
        random_state=42
    ))
])

pipe_mlp.fit(X_train, y_train)
pred = pipe_mlp.predict(X_test)

mse  = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
mae  = mean_absolute_error(y_test, pred)
r2   = r2_score(y_test, pred)

print(f"[MLP baseline 80/20] RMSE: {rmse:,.2f} | MAE: {mae:,.2f} | R²: {r2:.3f}")

# 3.5.3 Búsqueda rápida (RandomizedSearchCV, 20 combos, CV=5)
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform

grid_pipe = Pipeline(steps=[
    ("imp", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler()),
```

```
    ("model", MLPRegressor(
        early_stopping=True, validation_fraction=0.10,
        n_iter_no_change=12, max_iter=300, random_state=42
    ))
])

param_dist = {
    "model__hidden_layer_sizes": [(64,), (128,), (64,32), (128,64)],
    "model__activation": ["relu", "tanh"],
    "model__alpha": loguniform(1e-5, 1e-2),         # L2
    "model__learning_rate_init": loguniform(5e-4, 5e-3),
    "model__solver": ["adam", "lbfgs"]
}

rs = RandomizedSearchCV(
    estimator=grid_pipe,
    param_distributions=param_dist,
    n_iter=10,
    scoring="neg_mean_squared_error",
    cv=5,
    n_jobs=-1,
    random_state=42,
    verbose=1
)

idx = X.sample(frac=0.5, random_state=42).index
X_search, y_search = X.loc[idx], y[idx]
rs.fit(X_search, y_search)

rs.fit(X, y)
best_rmse_cv = np.sqrt(-rs.best_score_)
print("Mejores hiperparámetros (rápido):", rs.best_params_)
print(f"[MLP CV=5] Mejor RMSE: {best_rmse_cv:,.2f}")

# 3.5.4 Entrenar el mejor MLP en 80/20 y reportar métricas finales
best_mlp = rs.best_estimator_
best_mlp.fit(X_train, y_train)
pred_b = best_mlp.predict(X_test)

rmse_b = np.sqrt(mean_squared_error(y_test, pred_b))
mae_b  = mean_absolute_error(y_test, pred_b)
r2_b   = r2_score(y_test, pred_b)

print(f"[Best MLP 80/20] RMSE: {rmse_b:,.2f} | MAE: {mae_b:,.2f} | R²: {r2_b:.3f}")

## 4. Minería de Texto

# 4.1 Preparación del entorno
!pip -q install unidecode wordcloud nltk
```

```
import re, numpy as np, pandas as pd
from unidecode import unidecode
from wordcloud import WordCloud
import matplotlib.pyplot as plt

# stopwords en español
try:
    import nltk
    from nltk.corpus import stopwords
    try:
        _ = stopwords.words('spanish')
    except LookupError:
        nltk.download('stopwords')
    STOP_ES = set(stopwords.words('spanish'))
except Exception:
    STOP_ES = {
        "de","la","que","el","en","y","a","los","del","se","las","por","un","para",
        "con","no","una","su","al","lo","como","más","pero","sus","le","ya","o",
        "este","sí","porque","esta","entre","cuando","muy","sin","sobre"
    }

# ampliación con términos del dominio
STOP_ES |= {
    "usd","u$s","us$","dolares","dólares","venta","alquiler","caba","capital","federal",
    "propiedad","propiedades","inmueble","inmuebles","ambiente","ambientes","departamento"
}

# base de trabajo
df_txt = df_final.copy() if 'df_final' in globals() else dftp1.copy()
target_col = "price_usd"

# 4.2 Unificación + normalización de dominio

df_txt["title"] = df_txt.get("title", pd.Series("", index=df_txt.index)).fillna("")
df_txt["description"] = df_txt.get("description", pd.Series("", index=df_txt.index)).fillna("")
df_txt["text_raw"] = (df_txt["title"].astype(str) + " " +
df_txt["description"].astype(str)).str.strip()

DOM_MAP = {
    r"\bdepto[s]?\b": "departamento",
    r"\bdpto[s]?\b": "departamento",
    r"\bdto[s]?\b": "departamento",
    r"\bmonoambiente\b": "mono ambiente",
    r"\bamb\b": "ambiente",
    r"\bambientes\b": "ambiente",
    r"\bmts2\b|\bm2\b|\bmetros2\b": "metros cuadrados",
    r"\bmts\b|\bmetros\b": "metro",
    r"\bgarage\b|\bparking\b": "cochera",
```

```
    r"\bcochera[s]?\b": "cochera",
    r"\ba estrenar\b": "a_estrenar",
    r"\bamenities\b": "amenity",
    r"\b24hs\b|\b24 hs\b|\b24h\b": "seguridad_24hs",
    r"\bbalcon\b": "balcon",
    r"\bpileta\b|\bpiscina\b": "piscina",
    r"\bquincho\b": "quincho",
    r"\bsum\b": "sum"
}

def normalizar_dominio(texto: str) -> str:
    t = str(texto)
    for pat, rep in DOM_MAP.items():
        t = re.sub(pat, rep, t, flags=re.IGNORECASE)
    return t

def limpiar_texto(s: str) -> str:
    if pd.isna(s) or not str(s).strip():
        return ""
    s = normalizar_dominio(s)
    s = unidecode(s.lower().strip())              # sin tildes
    s = re.sub(r"[^a-z0-9\s_]", " ", s)           # mantengo '_' p/ tokens como a_estrenar
    s = re.sub(r"\b\d+\b", " ", s)                # saco números sueltos
    s = re.sub(r"\s+", " ", s).strip()
    return s

df_txt["text_clean"] = df_txt["text_raw"].map(limpiar_texto)
df_txt[["text_raw","text_clean"]].head(3)

# 4.3 TF-IDF (1–2-gramas) + top términos
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np, pandas as pd

token_pattern = r"(?u)\b[a-z0-9_]{3,}\b"

tfidf = TfidfVectorizer(
    preprocessor=lambda s: s,
    tokenizer=str.split,       # simple split por espacio
    token_pattern=None,        # desactivo el default porque uso tokenizer
    stop_words=list(STOP_ES),
    ngram_range=(1,2),
    min_df=10,
    max_features=40000
)

X_tfidf = tfidf.fit_transform(df_txt["text_clean"])
vocab = np.array(tfidf.get_feature_names_out())

mean_w = np.asarray(X_tfidf.mean(axis=0)).ravel()
```

```
top_idx = mean_w.argsort()[::-1][:30]
top_terms = pd.DataFrame({"termino": vocab[top_idx], "tfidf_prom": mean_w[top_idx]})
print("Top términos (TF-IDF):")
display(top_terms)
print("Shape TF-IDF:", X_tfidf.shape)

# 4.4 CountVectorizer (frecuencias) + bigramas
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(
    preprocessor=lambda s: s,
    tokenizer=str.split,
    token_pattern=None,
    stop_words=list(STOP_ES),
    ngram_range=(1,2),
    min_df=10,
    max_features=40000
)
X_cv = cv.fit_transform(df_txt["text_clean"])
vocab_cv = np.array(cv.get_feature_names_out())

mean_cv = np.asarray(X_cv.mean(axis=0)).ravel()
top_idx_cv = mean_cv.argsort()[::-1][:30]
top_terms_cv = pd.DataFrame({"termino": vocab_cv[top_idx_cv], "freq_prom": mean_cv[top_idx_cv]})
print("Top términos (frecuencia):")
display(top_terms_cv)
print("Shape CountVEC:", X_cv.shape)

# 4.5 Random Forest con texto (fix de indexado)

from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score
import numpy as np
import pandas as pd

mask_ok = df_txt[target_col].notna().to_numpy()
idx = np.flatnonzero(mask_ok)
y_text = df_txt.loc[mask_ok, target_col].astype(float).values
X_text = X_cv[idx, :]

rf_text = RandomForestRegressor(n_estimators=300, random_state=42, n_jobs=1)

mse_neg = cross_val_score(
    rf_text, X_text, y_text,
    scoring="neg_mean_squared_error", cv=5, n_jobs=1   # <- sin paralelismo
)
rmse_cv = np.sqrt(-mse_neg)
print(f"[RF-Texto] RMSE CV5 mean={rmse_cv.mean():.2f} | std={rmse_cv.std():.2f}")
```

```
rf_text.fit(X_text, y_text)
imp = pd.Series(rf_text.feature_importances_,
index=vocab_cv).sort_values(ascending=False).head(30)
display(imp.to_frame("importance"))

# 4.6 TF-IDF -> SVD y merge para modelado conjunto

from sklearn.decomposition import TruncatedSVD

k = 50  # 20-100 suele andar bien
svd = TruncatedSVD(n_components=k, random_state=42)
tfidf_svd = svd.fit_transform(X_tfidf)

cols_svd = [f"txt_svd_{i+1}" for i in range(k)]
df_svd = pd.DataFrame(tfidf_svd, columns=cols_svd, index=df_txt.index)

# unimos a la base de modelado
df_model = (df_final.copy() if 'df_final' in globals() else dftp1.copy()).join(df_svd)
print("df_model con texto (SVD) ->", df_model.shape)
df_model[cols_svd[:5]].head()
```