Proyecto final de Inteligencia Artificial

Reto de estimar el valor por metro cuadrado de una construcción.

Trabajo realizado por: Jereminth Muñoz y Guillermo Hernandez

Proceso para obtener resultados de regresión.

Se realizó la estimación utilizando los datasets entregados por el DANE, titulados *Censo de Edificaciones - CEED - 2007 - 2024 - IV Trimestre*. Este conjunto incluye cuatro datasets, de los cuales se seleccionaron tres: los correspondientes a los trimestres II, III y IV, ya que comparten todas sus variables (un total de 80). Cada dataset contiene un promedio de 3 millones de filas, lo que da un total aproximado de 9 millones de registros entre los tres, con un peso cercano a 4 GB.

Para llevar a cabo el proceso, se emplearon las siguientes herramientas: Python 3.13, Jupyter Notebook y librerías como *polars*, *sklearn*, *numpy*, *scipy*, entre otras. Estas permitieron abordar todas las fases del proyecto, desde la limpieza, el preprocesamiento y el acondicionamiento de los datos, hasta el desarrollo de modelos de inteligencia artificial de tipo regresión.

1. Limpieza:

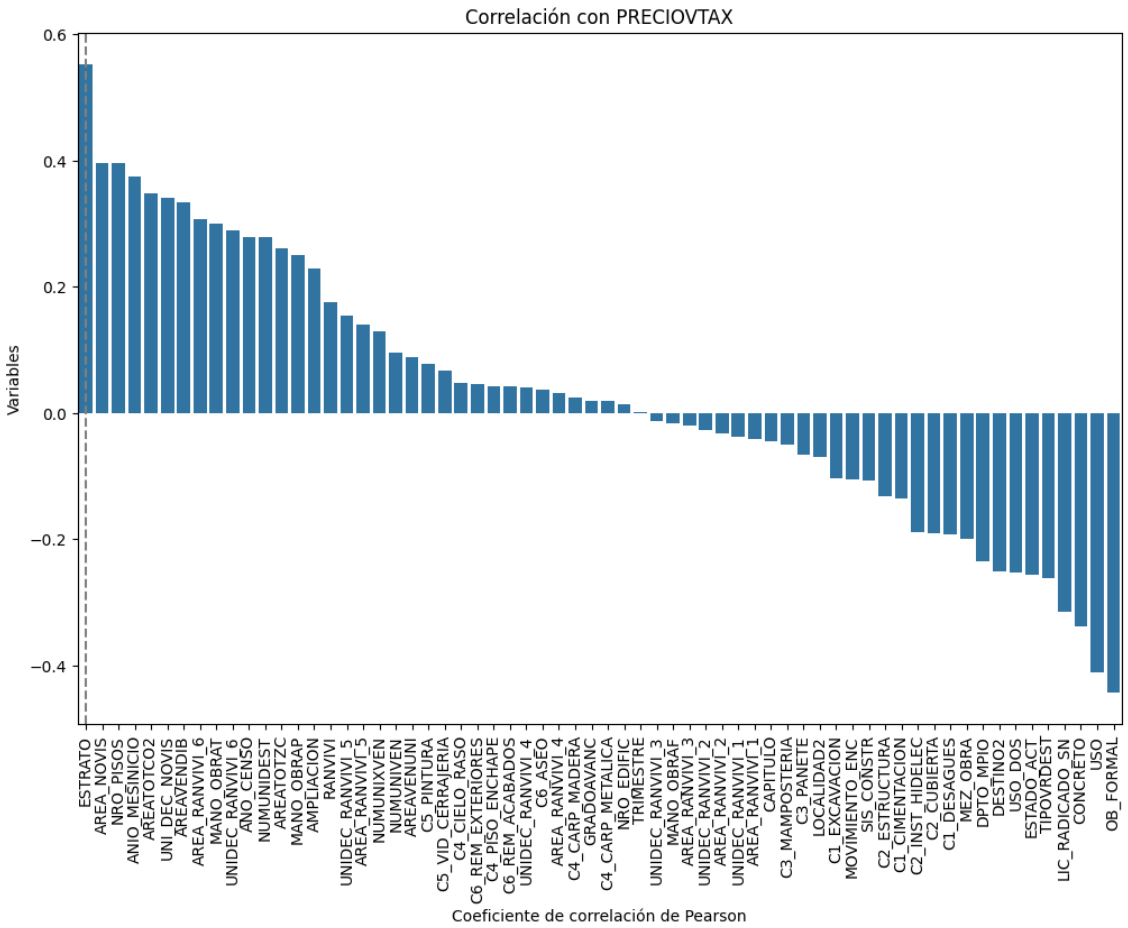
Para esta etapa nos enfrentamos a lo más complicado, lo que conlleva el 80% del tiempo y las decisiones necesarias para obtener un resultado esperado.

Lo primero que se hace es crear un notebook con el fin de ejecutar código por partes. Luego, se extrae una muestra para realizar la limpieza y estandarización de los datos, con el objetivo de aplicar posteriormente el mismo proceso a todo el conjunto. Esto se hace bajo la suposición de que dicha muestra aleatoria se comportará de manera similar al total de los datos. Para este caso, se eligieron 100.000 muestras por cada trimestre de forma aleatoria. Luego, se guarda ese dataset de prueba y se inicia el proceso de limpieza con la muestra.

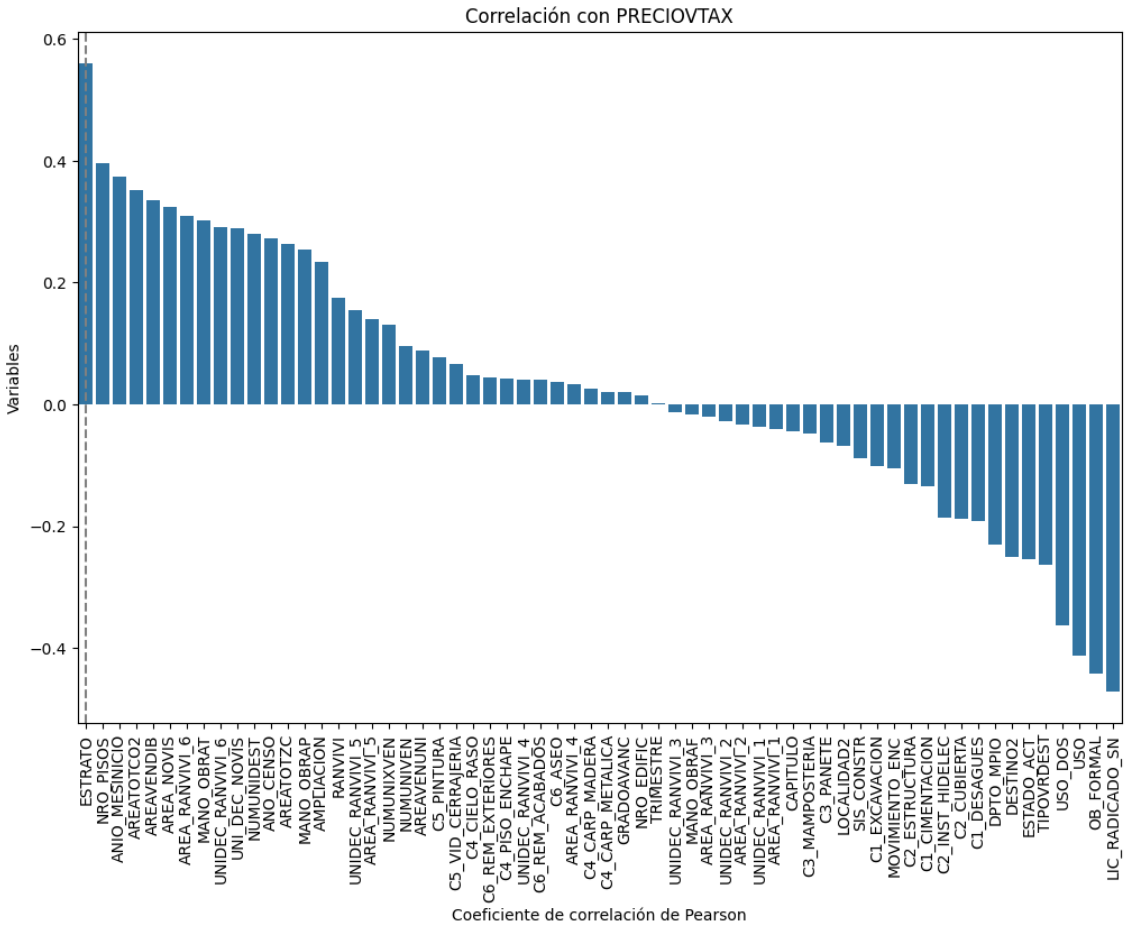
El primer paso consiste en observar las columnas con mayor cantidad de nulos o datos inválidos, con el fin de eliminar aquellas que tienen más del 70% de valores faltantes. Estas representan variables poco aprovechadas en el censo. Para las columnas con menos del 10% de nulos, se opta por eliminar directamente las filas

que los contienen, lo cual es un sacrificio aceptable. Posteriormente, se realiza la imputación de los valores faltantes bajo los siguientes criterios: a todas las variables que representan porcentajes (valores entre 0 y 100) se les asigna el promedio; para las variables categóricas, al analizar el censo se observa que sus valores van del 1 al número de categorías, por lo tanto, se utiliza el 0 para representar la categoría "No aplica". Finalmente, las variables tipo *string* se convierten en categóricas utilizando *LabelEncoder* y se eliminan las columnas que contienen identificadores únicos, ya que no aportan relevancia al entrenamiento del modelo.

Un proceso transversal fue revisar el comportamiento de la variable objetivo “PRECIOVTAX” para detectar posibles cambios significativos durante la limpieza. Se adjuntan las matrices de correlación antes y después de todo el proceso de depuración.



Primera grafica de correlación frente a la variable objetivo.



Segunda grafica de correlación frente a la variable objetivo después de limpiar.

Esto nos arroja que las variables con mayor correlación respecto al precio del metro cuadrado son: *Estrato* con 0.559, *Nro_pisos* con 0.395, *Lic_radicado_sn* con -0.471 y *Ob_formal* con -0.442. Esto indica que, a mayor estrato y número de pisos, el precio del metro cuadrado tiende a incrementarse. En cambio, la radicación de licencia y el licenciamiento penalizan dicho precio.

Con esto se concluye el proceso de limpieza y se procede a guardar el dataset limpio.

2. Tranformacion y acondicionamiento.

Aunque el dataset esté limpio y las variables estén alineadas para el entrenamiento, es necesario realizar un preprocesamiento adicional para escalar y normalizar los datos, así como mejorar su caracterización. Para ello, se utilizan herramientas como *StandardScaler*, *OneHotEncoder* y *pct_transform*, que permiten, por ejemplo, llevar las variables de porcentaje a un rango entre 0 y 1, escalar magnitudes como el área construida, o aplicar *OneHotEncoder* a columnas categóricas como el ID del municipio, estrato o destino.

En esta etapa es fundamental guardar las transformaciones por dos razones: para poder reutilizarlas en futuros entrenamientos con los mismos parámetros y para realizar predicciones en producción de forma consistente.

Estos archivos fueron guardados como y_scaler.joblib y feature_pipeline.joblib.

3. Entrenamiento de modelos de regresión

Se utilizaron cuatro modelos de regresión para la estimación:

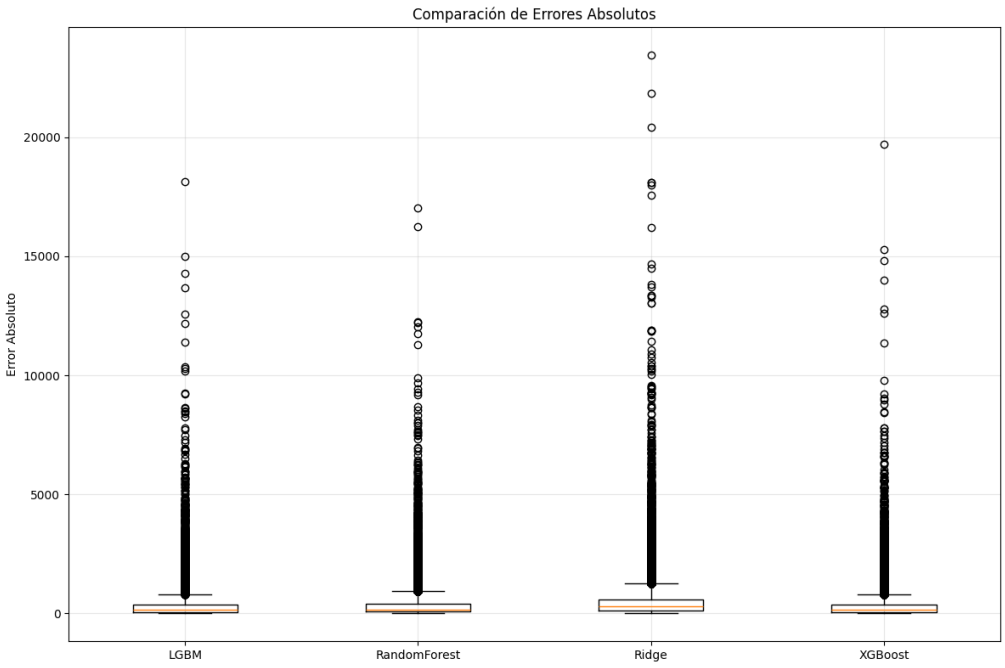
- **LGBM (LightGBM):** es un algoritmo de *gradient boosting* para problemas de regresión y clasificación. Se basa en construir una serie de árboles de decisión de forma secuencial, donde cada nuevo árbol corrige los errores del anterior. Se distingue por su velocidad y eficiencia, especialmente con grandes conjuntos de datos, gracias a técnicas como *Gradient-based One-Side Sampling* (GOSS) y *Exclusive Feature Bundling* (EFB). Uno de sus principales beneficios es la alta velocidad de entrenamiento.
- **RandomForest (Random Forest Regressor):** es un algoritmo de *ensemble learning* que construye múltiples árboles de decisión durante el entrenamiento y produce la media (para regresión) o la moda (para clasificación) de las predicciones de los árboles individuales. La aleatoriedad proviene de la selección aleatoria de subconjuntos de datos y características para cada árbol, lo que ayuda a reducir el sobreajuste. Es muy preciso, pero puede ser lento con grandes volúmenes de datos.
- **Ridge (Ridge Regression):** es una extensión de la regresión lineal que busca resolver el problema de la multicolinealidad (cuando las variables predictoras están altamente correlacionadas entre sí) y el sobreajuste. Para

ello, añade un término de penalización (regularización L2) a la función de costo de la regresión lineal, lo que contrae los coeficientes hacia cero sin eliminarlos completamente. Su principal ventaja es la reducción del sobreajuste.

- **XGBoost (Extreme Gradient Boosting):** es otro algoritmo de *gradient boosting* muy popular y de alto rendimiento, similar en concepto a LGBM. También construye árboles de decisión de forma secuencial, corrigiendo los errores de los árboles anteriores. Se destaca por su eficiencia, escalabilidad y por incluir regularización para evitar el sobreajuste. Es ampliamente utilizado en competiciones de ciencia de datos por su precisión y rendimiento.

Estos cuatro modelos fueron entrenados y evaluados utilizando las métricas MAE, RMSE y R^2. Adicionalmente, para analizar la estabilidad de los modelos, se aplicó una técnica de *cross-validation*, que permite dividir y cruzar los datos para validar y observar la consistencia de los resultados.

A partir de esta validación cruzada, se calcularon también las métricas **CV_MAE**, **CV_RMSE** y **CV_R^2**, que representan los promedios obtenidos al aplicar la validación cruzada con **5 particiones** (K=5), es decir, el proceso se repitió cinco veces con diferentes divisiones de los datos.



Tercer grafica evaluando qué tan grandes son las diferencias entre los valores predichos por cada modelo y los valores reales.

LGBM, **RandomForest** y **XGBoost** presentan un comportamiento muy similar en cuanto a la distribución de sus errores, incluidos los valores atípicos. Esto sugiere que, para este conjunto de datos específico y considerando la métrica de error absoluto, sus rendimientos son bastante comparables.

Modelo	MAE	RMSE	R^2	CV_MAE	CV_RMSE	CV_R^2
LGBM	305.92	569.81	0.805	304.46	560.80	0.807
RandomForest	330.38	602.25	0.782	329.26	596.41	0.781
Ridge	467.44	809.41	0.607	462.56	781.26	0.625
XGBoost	302.44	565.42	0.808	301.86	553.08	0.812

Con estos resultados, se puede observar que los modelos con mejor desempeño fueron **LGBM** y **XGBoost**. A estos dos se les aplicó la técnica **HalvingRandomSearchCV** con el objetivo de encontrar sus mejores hiperparámetros.

Lo que se obtuvo como resultado.

models = {

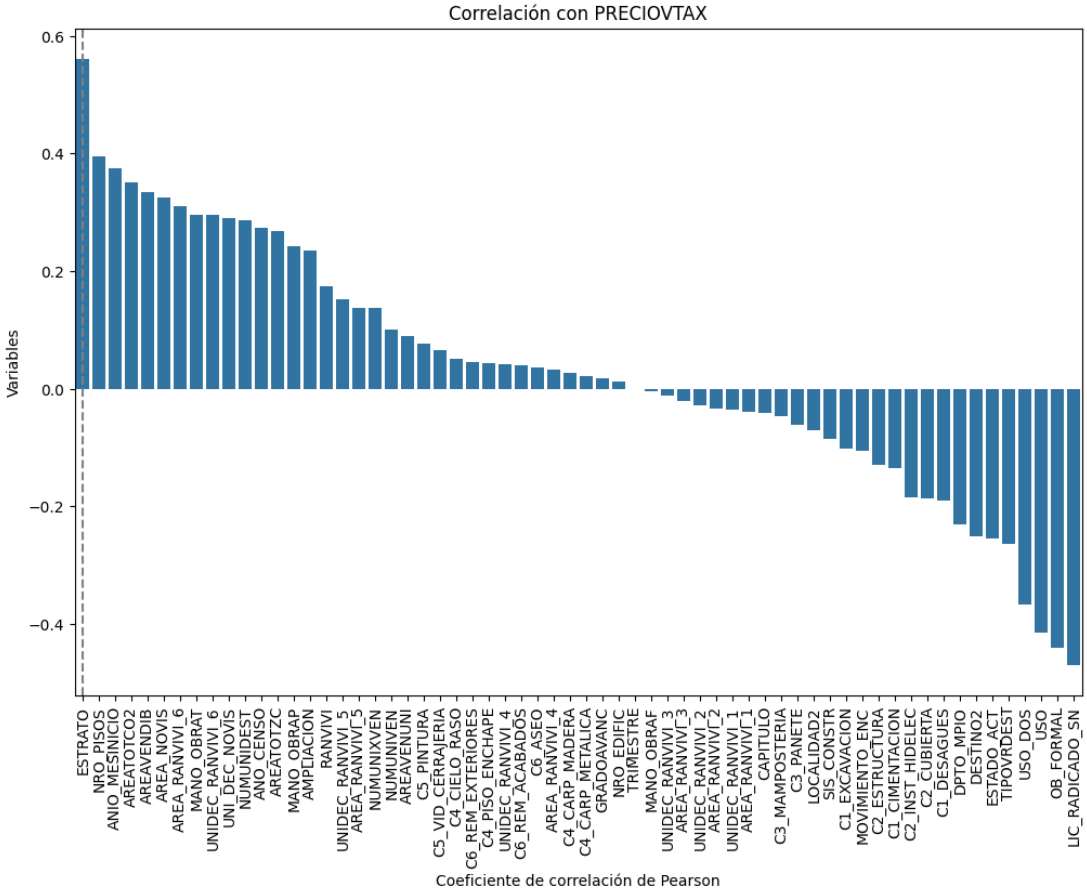
"LGBM": LGBMRegressor(n_estimators=1200, learning_rate=0.286, min_child_samples=12, max_depth=9, num_leaves=219, subsample=0.56, colsample_bytree=0.8, random_state=42),

"XGBoost": XGBRegressor(n_estimators=1200, learning_rate=0.098, min_child_weight=9, max_depth=8, subsample=0.54, colsample_bytree=0.8, random_state=42, n_jobs=-1)

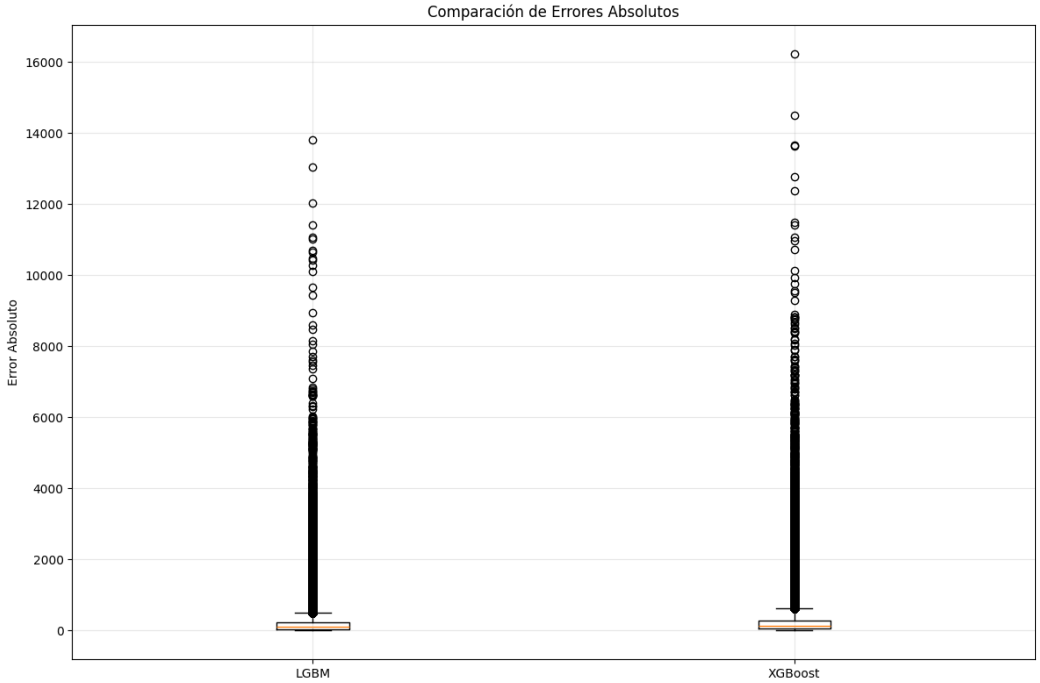
}

Cada modelo fue guardado correctamente para su uso posterior. Como último paso, se entrenó un modelo con el dataset completo para evaluar su comportamiento. Dado el volumen de datos, se optó por entrenar únicamente los dos modelos con mejor rendimiento, con la expectativa de obtener mejoras significativas.

Es importante aclarar que las transformaciones previamente guardadas se aplicaron al dataset completo para asegurar coherencia en los resultados y mantener un comportamiento consistente. Asimismo, se aplicó nuevamente todo el proceso de limpieza para obtener un dataset final completamente limpio.



Cuarta grafica comportamiento de la correlación con todo el dataset (9 millones).



Quinta grafica evaluando qué tan grandes son las diferencias entre los valores predichos por cada modelo y los valores reales.

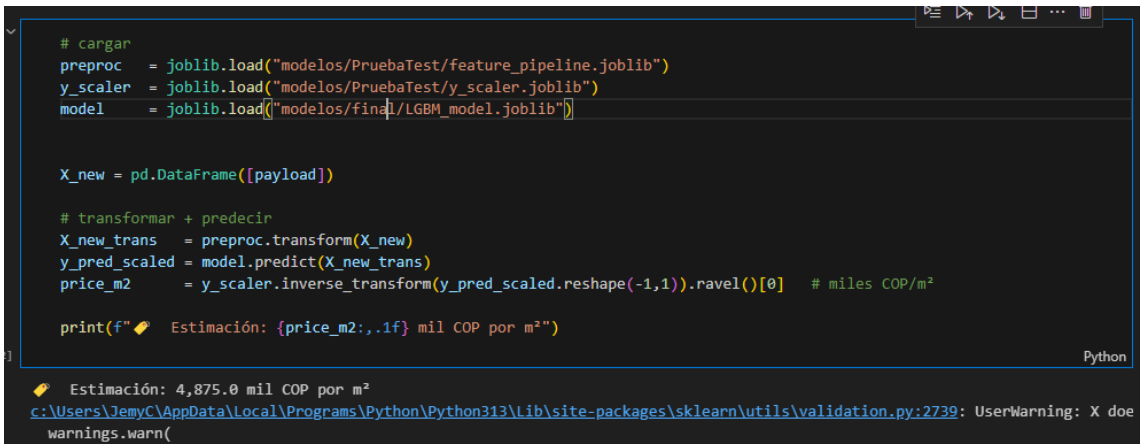
Por cuestiones de memoria, se optó por no realizar validación cruzada. Por lo tanto, únicamente se reportarán las tres métricas principales de los modelos: MAE, RMSE y R^2.

Modelo	MAE	RMSE	R^2
LGBM	191.92	350.05	0.925
XGBoost	240.65	425.07	0.889

Esto declara como ganador al modelo **LGBM**, con un **R^2 de 0.92**.

Como punto final, se concluye que la estrategia de tomar una muestra para realizar todo el proceso inicial y verificar si los resultados son prometedores permite, posteriormente, entrenar un modelo final con todo el conjunto de datos. Aunque el alto volumen de información limita al equipo a entrenar únicamente el modelo sin aplicar validación cruzada, se evidencia una mejora significativa: pasar de un R^2 de 0.80 en el dataset de prueba a 0.92 con el dataset completo representa un avance relevante.

Al momento de utilizar el modelo en producción, es fundamental aplicar correctamente las transformaciones y los escalados previamente guardados, para asegurar resultados coherentes.



Sexta grafica Probando el modelo.

Nota. La RAM Y CPU casi explota