



Predicción del precio óptimo de compra de sacos de papel en la industria cementera: Un enfoque basado en modelos SARIMAX, ARIMA y red neuronal LSTM.

ANDRÉS FELIPE MORALES MARTÍNEZ

Tesis

Asesor

Diego Fernando Fonseca Valero

UNIVERSIDAD EAFIT
ESCUELA DE CIENCIAS APLICADAS E INGENIERÍA
MAESTRÍA EN CIENCIA DE DATOS Y ANALÍTICA
MEDELLÍN
2025

CONTENIDO

0	INTRODUCCIÓN	6
1	PLANTEAMIENTO DEL PROBLEMA	7
	1.1. Antecedentes y contexto	7
	1.2. Formulación del problema	7
	1.3. Pregunta de investigación	8
	1.4. Alcance y viabilidad	8
2	Justificación	9
3	Objetivos	10
	3.1. Objetivo general	10
	3.2. Objetivos específicos	10
4	Marco teórico y Metodología	11
	4.1. Gestión de compras bajo incertidumbre de precios	11
	4.2. Modelos de series temporales para predicción de precios	12
	4.2.1. Modelos lineales: ARIMA y SARIMAX	12
	4.2.2. Modelos no lineales de aprendizaje profundo: redes LSTM	13
	4.2.3. Modelos de <i>gradient boosting</i> en series temporales: LightGBM	13
	4.2.4. Enfoques híbridos y metodologías de <i>ensemble</i>	14
	4.3. Evaluación rigurosa de modelos de pronóstico	15
	4.3.1. Métricas de error y capacidad explicativa	15
	4.3.2. Protocolos de validación y comparación entre modelos	15
	4.4. Brecha identificada y aportes esperados	16
	4.5. Síntesis comparativa de la literatura	16
	4.6. Metodología	17
	4.7. Enfoque general y diseño experimental	17
	4.8. Glosario de abreviaturas	18
	4.9. Datos y fuentes de información	19
	4.9.1. Datos internos: serie objetivo	19
	4.9.2. Datos externos: variables exógenas	20
	4.10. Preprocesamiento y control de fuga de información	21
	4.10.1. Reconstrucción de serie diaria	21
	4.10.2. Partición temporal y transformaciones	21
	4.11. Ingeniería de características	22
	4.11.1. Lags y suavizadores del objetivo	22

4.11.2. Retornos y volatilidad de variables exógenas	22
4.11.3. Variables de calendario	23
4.12. Modelos y especificaciones	23
4.12.1. Modelos ARIMA y SARIMAX	23
4.12.2. Modelo LSTM	23
4.12.3. Modelo LightGBM	23
4.13. Esquema de validación temporal	24
4.14. Métricas de evaluación	25
4.15. Reproducibilidad y trazabilidad	26
5 Resultados	27
5.1. Análisis exploratorio de datos	27
5.2. Fase I: modelos base	28
5.3. Fase II: LSTM con variables exógenas	29
5.4. Fase III: transformación de datos y Gradient Boosting	30
5.5. Fase IV: LSTM con serie diaria transformada	31
5.6. Comparación global	32
6 Conclusiones	34
6.1. Conclusiones generales	34
6.2. Contribuciones principales	35
6.3. Implicaciones para la gestión de compras	36
6.4. Limitaciones del estudio	37
6.5. Líneas de trabajo futuro	38
REFERENCIAS	39

LISTA DE TABLAS

Tabla 1	Resumen comparativo de familias de modelos consideradas en la tesis. . .	14
Tabla 2	Tabla resumen comparativa de modelos predictivos en la literatura. Resultados reportados en diferentes dominios de aplicación para modelos ARIMA/SARIMAX, LSTM y enfoques híbridos; las métricas específicas varían según el estudio original.	17
Tabla 3	Comparación de desempeño (modelos base y LSTM con exógenas, conjunto de prueba).	29
Tabla 4	Comparación de desempeño en conjunto de prueba (10 % más reciente).	33

LISTA DE FIGURAS

Figura 1	Esquema de ventaneo para el modelo LSTM: para cada ventana de entrada se genera un pronóstico un día adelante, respetando el orden temporal.	24
Figura 2	Esquema de partición temporal 90/10 y validación cruzada con ventanas crecientes.	25
Figura 3	Serie histórica del precio neto de compra de sacos de papel kraft (2014–2023).	28
Figura 4	Comportamiento del modelo ARIMA en el conjunto de prueba: valores observados vs. pronosticados.	29
Figura 5	Desempeño de LightGBM en el conjunto de prueba (10 % más reciente): valores observados vs. pronosticados.	31
Figura 6	Desempeño del LSTM con serie diaria transformada en el conjunto de prueba: valores observados vs. pronosticados.	32

Resumen

En esta tesis, existe un problema importante relacionado con la predicción del precio de compra de sacos de papel kraft en un entorno de alta incertidumbre y con ausencia de herramientas de cobertura financiera. Después de extraer conocimiento transaccional del ERP (2014–2023) y variables exógenas públicas (TRM y precios de acciones de proveedores globales), construimos un conjunto de datos diario, escalonado con el esquema LOCF, y creamos una canalización de características diseñada para evitar la fuga de información. Usando esto como plataforma, se eligieron cuatro modelos basados en series temporales: ARIMA, SARIMAX, red neuronal LSTM y LightGBM como método de aumento de gradiente en árboles de decisión, de acuerdo con un protocolo de validación temporal del 90/10 junto con la evaluación en un conjunto de datos reciente de retención. El resultado indica que los modelos lineales y el LSTM básico en la serie original tienen errores absolutos y relativos muy altos, por lo que no pueden apoyar decisiones de compra precisas. LightGBM aplicado a la serie diaria transformada, por otro lado, reduce los errores relativos a alrededor del 1 %, y el LSTM entrenado con esa misma representación ofrece el mejor rendimiento general, en términos de MAE aproximadamente dos veces más bajo que el de LightGBM en el último período. Estos resultados muestran que una buena representación de la serie, junto con un modelo no lineal y variables externas significativas, también aseguran que las previsiones puedan ser operativamente útiles, allanando el camino hacia una gestión de compras más basada en datos para este insumo clave.

Palabras clave: Pronóstico de precios, series de tiempo, LSTM, LightGBM, gestión de compras, sacos de papel kraft.

0. INTRODUCCIÓN

La bolsa de papel kraft de 50 kg es un factor importante en la estructura de costos del producto terminado en la industria del cemento. El precio de su compra en pesos colombianos es resultado de fuerzas externas, por ejemplo, los tipos de cambio y el estado del comercio internacional de celulosa y papel, sin un instrumento formal de cobertura (por ejemplo, mercado de futuros) para este insumo. Como consecuencia, las decisiones de compra se toman con gran incertidumbre respecto al movimiento futuro de precios, lo que reduce el poder de negociación y la eficiencia de la planificación. En los últimos años, los cambios recientes en los métodos de pronóstico de series temporales han permitido una evolución de modelos lineales más tradicionales, incluidos ARIMA y SARIMAX, a modelos de series temporales basados en aprendizaje profundo y algoritmos de aumento de gradiente que tienen en cuenta interrelaciones complejas entre múltiples variables y series temporales.

Sin embargo, el uso sistemático de estos modelos dentro de América Latina para el control de ciertas compras de insumos, particularmente cuando se combina datos transaccionales internos con variables exógenas replicables (y públicas), es limitado. Nuestro trabajo aplica el pronóstico del precio de compra de bolsas de papel kraft para empresas cementeras colombianas utilizando datos históricos del sistema ERP y series exógenas públicas (TRM y precios de acciones de proveedores globales relevantes) para el período 2014–2023. El objetivo principal es construir, validar y comparar modelos ARIMA, SARIMAX, LSTM y LightGBM utilizando un protocolo de validación temporal sólido y compararlos entre sí, para evaluar un conjunto de soluciones (procesamiento de nivel intermedio) capaces de generar señales de compra accionables que mitiguen dicha incertidumbre en el proceso de toma de decisiones de compra.

Este documento está organizado de la siguiente manera: La Sección 1 formula el problema de investigación, incluyendo antecedentes, formulación declarativa, pregunta central y alcance. La Sección 2 está dedicada a las justificaciones prácticas y académicas del estudio. El objetivo general y los objetivos específicos se presentan en la Sección 3. La Sección 4 integra el marco teórico y la metodología en un solo documento que explica tanto los fundamentos conceptuales como el diseño experimental, los datos, el preprocesamiento, los modelos, las métricas y los criterios de evaluación. Luego, la Sección 5 presenta y discute los resultados obtenidos en las diferentes fases de modelado. Finalmente, la Sección 6 resume las principales conclusiones, las contribuciones del estudio, sus limitaciones y las líneas de trabajo futuro.

1. PLANTEAMIENTO DEL PROBLEMA

1.1. Antecedentes y contexto

La empresa Cementos Argos ha empleado bolsas de papel kraft de 50 kg como el principal insumo para el embalaje de cemento. Tradicionalmente, el precio de compra se decidía después de un número limitado de discusiones con uno o algunos proveedores clave, junto con una investigación descriptiva de las tendencias recientes de precios y pronósticos cualitativos de la condición económica. La compañía tiene un historial de precios de compra transaccionales de 2014 a 2023, junto con información pública sobre el tipo de cambio y los precios de las acciones de las empresas del sector papelerero, pero esta información no se ha incorporado en un sistema formal de pronóstico. En la práctica, las decisiones sobre el volumen de compra y el momento se toman en respuesta a la experiencia de los equipos de compras y a las señales parciales del mercado sin un marco cuantitativo para estimar escenarios de precios y evaluar alternativas. Este escenario también sugiere que, en el contexto de una devaluación de la moneda o choques en los mercados internacionales de papel, la empresa corre un riesgo considerable de pagar precios altos sin la información para prepararse para tal impacto y evitarlo.

1.2. Formulación del problema

En la situación actual, la empresa:

- Toma decisiones de compra de sacos de papel kraft sin contar con pronósticos cuantitativos sistemáticos del precio de compra.
- No dispone de un protocolo estandarizado para comparar el desempeño de diferentes modelos de pronóstico ni para traducir los resultados en reglas operativas de compra.
- Aprovecha de manera parcial la información disponible en su ERP y en fuentes públicas externas.

Como consecuencia, las decisiones de negociación y programación de compras se realizan con alta incertidumbre, lo que puede traducirse en mayores costos y en una menor capacidad de aprovechar ventanas de oportunidad cuando el precio del insumo presenta caídas temporales.

El problema que aborda este trabajo se puede formular de manera declarativa como sigue:

La empresa carece de un sistema cuantitativo de pronóstico de precios que integre datos internos y señales exógenas, y que permita evaluar de manera comparativa distintos modelos de series de tiempo, con el fin de apoyar las decisiones de compra de sacos de papel kraft y reducir la incertidumbre en la negociación.

1.3. Pregunta de investigación

A partir de la formulación anterior, la pregunta central de investigación es:

¿Qué modelo de pronóstico basado en series de tiempo (ARIMA, SARIMAX, LSTM o LightGBM), entrenado con datos transaccionales internos y series exógenas públicas, ofrece el mejor desempeño fuera de muestra para anticipar el precio de compra de sacos de papel kraft en una empresa cementera colombiana y cómo puede traducirse ese pronóstico en señales operativas de compra?

1.4. Alcance y viabilidad

El estudio se limita a:

- Una empresa cementera específica, con datos internos de precios de sacos de papel kraft correspondientes al periodo 2014–2023.
- Un conjunto de variables exógenas públicas seleccionadas por su relevancia económica y disponibilidad: tipo de cambio representativo del mercado (TRM) y precios accionarios de fabricantes globales de papel y empaques.
- El horizonte de pronóstico de corto plazo requerido para la planificación de compras y negociaciones periódicas, trabajando con un esquema de validación temporal de tipo 90/10 y evaluación en un bloque de prueba holdout.

El objetivo no es desarrollar un contrato de cobertura o una cartera de los mejores proveedores, sino evaluar el potencial de diferentes modelos de series temporales para mejorar la información disponible en el momento de la toma de decisiones. La viabilidad del estudio se facilita mediante datos históricos, modelado estadístico y herramientas de aprendizaje automático; además, la replicabilidad del protocolo metodológico en otros insumos críticos de la cadena de suministro de la empresa.

2. Justificación

El precio de las bolsas de papel kraft es, desde un punto de vista práctico, una fracción considerable del costo total del cemento empaquetado. Pequeñas diferencias en el precio de compra, acumuladas a lo largo del volumen anual, pueden tener efectos financieros considerables. Cuando existen pronósticos confiables del precio de este insumo, se permite identificar cuándo existen ventanas de oportunidad para negociar o preacordar compras, minimiza la exposición a períodos de altos precios y proporciona verificación numérica de las decisiones de suministro. Desde la perspectiva del análisis de datos, este problema requiere convertir el rastro transaccional del ERP y las series exógenas públicas en un sistema de pronóstico que sea auditable, reproducible y reutilizable dentro de la organización. Esto implica transformar fuentes históricas heterogéneas en un flujo estructurado que permita generar, validar y comparar modelos bajo criterios metodológicos consistentes, de manera que los pronósticos puedan incorporarse de forma confiable en los procesos de planeación y negociación de compras. Producir un sistema de pronóstico replicable, basado en evidencia y científicamente correcto ayuda a convertir este conocimiento en un activo analítico que complementa la estrategia de gestión de riesgos y la continuidad del negocio. En el ámbito académico, el presente trabajo responde a un vacío particular en la literatura aplicada de América Latina: llenar la escasez de estudios sobre las series de precios de insumos industriales, a saber, bolsas de papel kraft, basándose en datos transaccionales empresariales en combinación con variables exógenas públicas y evaluando formalmente modelos lineales y no lineales utilizando un método de validación temporal. A través del detalle del diseño y verificación del conjunto de datos y la configuración de los modelos ARIMA, SARIMAX, LSTM y LightGBM en condiciones realistas, la tesis ofrece la oportunidad de generar un estudio de investigación replicable en la comunidad de análisis de datos de la cadena de suministro y ciencia de la decisión. Finalmente, este trabajo se alinea con el objetivo de la organización de mejorar su capacidad analítica sofisticada en áreas centrales como la adquisición y gestión de proveedores, y juega un papel en proporcionar prueba interna de la importancia de combinar métodos de series temporales y aprendizaje automático en la toma de decisiones operativas.

3. Objetivos

3.1. Objetivo general

Desarrollar, validar y comparar modelos de pronóstico basados en series de tiempo (ARIMA, SARIMAX, LSTM y LightGBM), entrenados con datos transaccionales internos y variables exógenas públicas, para anticipar el precio de compra de sacos de papel kraft en una empresa cementera colombiana y evaluar su utilidad como soporte a las decisiones operativas de compra.

3.2. Objetivos específicos

1. Construir un conjunto de datos limpio y reproducible que integre el historial de precios de compra de sacos de papel kraft con series exógenas públicas relevantes, documentando criterios de depuración, imputación y transformación temporal.
2. Desarrollar y entrenar modelos ARIMA, SARIMAX, LSTM y LightGBM bajo un esquema de validación temporal, cuidando la ausencia de fuga de información y la comparabilidad entre enfoques.
3. Evaluar comparativamente el desempeño de los modelos mediante métricas de error fuera de muestra (por ejemplo, MAE, RMSE, sMAPE, WAPE y R^2), en relación con modelos de referencia ingenuos.
4. Analizar el potencial de los modelos con mejor desempeño para generar señales operativas de compra (por ejemplo, ventanas de oportunidad asociadas a caídas pronosticadas de precio) y discutir su impacto esperado en el costo del insumo.
5. Documentar el protocolo metodológico y los artefactos desarrollados (scripts, configuraciones, transformaciones) de manera que faciliten su replicación y adaptación a otros insumos y contextos.

4. Marco teórico y Metodología

La predicción de precios en cadenas de suministro constituye un desafío metodológico que combina la teoría económica con técnicas avanzadas de análisis de series de tiempo [1, 2]. En el contexto de insumos industriales como los sacos de papel kraft, la empresa compradora enfrenta decisiones de abastecimiento bajo alta incertidumbre de precios, exposición cambiaria y restricciones contractuales [3]. Este marco teórico presenta los fundamentos conceptuales que sustentan la investigación, organizados en tres ejes principales: (i) la gestión de compras bajo incertidumbre de precios en cadenas de suministro, (ii) los enfoques metodológicos para la predicción de series temporales aplicados a precios y costos, y (iii) las métricas y protocolos de evaluación rigurosa de modelos de pronóstico. Finalmente, se sintetiza la brecha específica que aborda esta tesis.

4.1. Gestión de compras bajo incertidumbre de precios

La gestión de compras en cadenas de suministro busca asegurar el abastecimiento oportuno de insumos críticos al menor costo posible, garantizando al mismo tiempo la continuidad operativa y el cumplimiento de estándares de calidad [3]. En mercados donde existen instrumentos de cobertura bien desarrollados, las empresas pueden mitigar parte del riesgo de precio mediante estrategias financieras; sin embargo, en muchos insumos especializados, como los empaques industriales de papel kraft, no existen mercados de derivados líquidos, por lo que la exposición al riesgo de precio se gestiona principalmente mediante decisiones operativas y negociaciones con proveedores [4].

La incertidumbre se manifiesta en la variabilidad de costos de materias primas, la volatilidad del tipo de cambio y las condiciones de oferta de los proveedores. La literatura en compras y gestión de costos resalta la importancia de ir más allá del precio unitario y considerar el costo total de propiedad del insumo, es decir, todos los costos asociados a la adquisición, uso y disposición del bien [5]. Cuando no se dispone de herramientas cuantitativas de pronóstico, las decisiones de compra tienden a basarse en promedios históricos recientes o en la intuición de los responsables de compras, lo que puede conducir a compras en momentos desfavorables y a una utilización ineficiente del poder de negociación.

Diversos autores han enfatizado que la integración de información histórica y modelos de pronóstico en procesos de compras permite identificar ventanas de oportunidad, estimar rangos plausibles de evolución de precios y respaldar con evidencia cuantitativa la negociación

con proveedores [2, 6]. Para que estos modelos sean útiles, deben construirse sobre datos confiables, evaluarse rigurosamente y traducirse en indicadores comprensibles para los gestores de la cadena de suministro.

4.2. Modelos de series temporales para predicción de precios

La predicción de precios a partir de series temporales ha sido abordada tradicionalmente mediante modelos estadísticos lineales, a los que se han sumado en las últimas décadas enfoques no lineales de aprendizaje profundo y métodos de *gradient boosting* [6]. Cada familia de modelos presenta supuestos, fortalezas y limitaciones que resultan relevantes al analizar series de precios con comportamiento escalonado, presencia de *shocks* y posible influencia de variables exógenas.

4.2.1. Modelos lineales: ARIMA y SARIMAX Los modelos autorregresivos integrados de media móvil (ARIMA) constituyen uno de los enfoques clásicos para el pronóstico de series de tiempo. Estos modelos suponen que el valor futuro de la serie puede representarse como una combinación lineal de sus propios rezagos y de términos de error pasados, después de aplicar, si es necesario, transformaciones de diferenciación para lograr estacionariedad [1, 2]. La familia ARIMA ofrece un marco bien establecido, con criterios de selección de órdenes, diagnóstico de residuales y herramientas de inferencia. Siguiendo la formulación clásica de los modelos ARIMA propuesta por Box et al. [1], se asume que el valor futuro de la serie puede escribirse como una combinación lineal de rezagos de la propia serie y de errores pasados, después de aplicar las transformaciones necesarias para lograr estacionariedad.

Cuando se dispone de variables exógenas que pueden contener información relevante sobre la evolución futura de la serie objetivo, los modelos se extienden a la clase SARIMAX, que incorpora regresores externos y, cuando corresponde, componentes estacionales [2]. En el contexto de precios de insumos industriales, variables como el tipo de cambio o indicadores de mercado de proveedores pueden tratarse como regresores que desplazan el nivel esperado de la serie, manteniendo al mismo tiempo la estructura autorregresiva.

La principal ventaja de ARIMA y SARIMAX radica en su interpretabilidad y en la amplia experiencia acumulada en su uso. Sin embargo, su capacidad para capturar dinámicas fuertemente no lineales, cambios de régimen y comportamientos escalonados puede ser limitada, especialmente cuando la relación entre la serie objetivo y las exógenas es compleja o cuando los ajustes de precio se producen en forma de saltos discretos en lugar de variaciones

graduales [6].

4.2.2. Modelos no lineales de aprendizaje profundo: redes LSTM Las redes neuronales recurrentes (RNN) y, en particular, las arquitecturas de memoria de largo corto plazo (LSTM, por sus siglas en inglés), han sido ampliamente adoptadas para modelar series de tiempo con dependencias temporales complejas. Las LSTM introducen celdas de memoria y mecanismos de compuertas que permiten retener información relevante durante horizontes largos y mitigar problemas de desvanecimiento del gradiente [7].

En el ámbito de la predicción de precios, las LSTM se han utilizado para capturar relaciones no lineales entre la serie objetivo y un conjunto de variables de entrada que incluyen rezagos del propio precio, indicadores técnicos, variables macroeconómicas o de mercado, y otros factores exógenos [6]. A diferencia de los modelos lineales, las LSTM no imponen una estructura funcional predeterminada, sino que aprenden patrones a partir de los datos mediante entrenamiento supervisado.

Esta flexibilidad viene acompañada de desafíos: la necesidad de contar con conjuntos de datos suficientemente ricos, la sensibilidad a la selección de hiperparámetros (tamaño de la red, longitud de ventana, tasas de aprendizaje, regularización) y el riesgo de sobreajuste si no se emplean esquemas de validación adecuados [2]. Además, la interpretabilidad suele ser menor que en modelos lineales, lo que exige recursos adicionales para explicar el comportamiento del modelo a tomadores de decisión no expertos en aprendizaje profundo.

4.2.3. Modelos de *gradient boosting* en series temporales: LightGBM Los métodos de *gradient boosting* basados en árboles de decisión, como LightGBM, han demostrado un alto desempeño en tareas de regresión y clasificación con datos tabulares [8, 9]. Estos modelos construyen un ensamble de árboles poco profundos que se ajustan de manera secuencial para corregir los errores de las iteraciones previas, optimizando una función de pérdida específica.

Aplicados a series temporales, estos métodos requieren que la información temporal se represente explícitamente mediante características (por ejemplo, rezagos de la serie objetivo, retornos y volatilidades de variables exógenas, variables de calendario). Una vez construida esta representación, el modelo de *boosting* puede capturar interacciones no lineales y efectos umbral sin necesidad de especificar una forma funcional explícita [8].

Entre las ventajas de enfoques como LightGBM se encuentran su capacidad para manejar relaciones complejas, su eficiencia computacional en conjuntos de datos de tamaño moderado

y la posibilidad de obtener medidas de importancia de variables que apoyan la interpretación [9]. No obstante, al no modelar la dependencia temporal de manera explícita como un proceso estocástico, la calidad del pronóstico depende en gran medida de la ingeniería de características y del cuidado en el diseño del esquema de validación temporal.

Modelo	Supuestos principales	Fortalezas	Limitaciones
ARIMA/SARIMAX	Linealidad, estructura ARMA, estacionariedad (tras transformación)	Interpretables, marco bien establecido	Dificultad para capturar no linealidades fuertes y cambios de régimen
LSTM	Relación no lineal entre entradas y objetivo, dependencias temporales complejas	Capacidad para capturar dependencias de largo plazo	Requiere más datos, calibración compleja y menor interpretabilidad
LightGBM	Relación no lineal, datos tabulares con características explícitas	Maneja interacciones complejas, eficiente en conjuntos de datos tabulares	Depende fuertemente de la ingeniería de características y del esquema de validación

Tabla 1. Resumen comparativo de familias de modelos consideradas en la tesis.

La Tabla 1 resume de forma sintética los supuestos, fortalezas y limitaciones de las familias de modelos consideradas en este estudio.

4.2.4. Enfoques híbridos y metodologías de *ensemble* La literatura reciente reporta enfoques híbridos que combinan modelos lineales (por ejemplo, ARIMA) con modelos no lineales (como redes LSTM), con la intención de aprovechar la capacidad de los primeros para capturar estructura lineal y la de los segundos para modelar patrones residuales no lineales [6]. Una estrategia común consiste en ajustar un modelo ARIMA a la serie original, modelar los residuales mediante una red LSTM y combinar las predicciones de ambos componentes.

Estos enfoques híbridos pueden ofrecer mejoras de desempeño en ciertos contextos, pero también incrementan la complejidad del sistema, tanto en términos de calibración como de mantenimiento. Además, requieren un cuidado especial en el diseño del esquema de validación para evitar sobreestimar su capacidad predictiva. En el contexto de esta tesis, los modelos híbridos se consideran como parte del panorama conceptual de la literatura, pero el análisis empírico se centra en la comparación de ARIMA, SARIMAX, LSTM y LightGBM, que son los modelos incluidos formalmente en los objetivos del estudio.

4.3. Evaluación rigurosa de modelos de pronóstico

El valor práctico de un modelo de pronóstico no depende únicamente de su sofisticación técnica, sino de la solidez con la que se evalúa su desempeño y de la forma en que sus resultados se traducen en decisiones de negocio. La literatura enfatiza dos componentes centrales en esta evaluación: (i) la selección e interpretación de métricas de error y (ii) el uso de protocolos temporales consistentes para validar y comparar modelos fuera de muestra [2], [6].

4.3.1. Métricas de error y capacidad explicativa Las métricas de error cuantifican la discrepancia entre los valores observados y los pronosticados. Entre las más utilizadas en problemas de precios se encuentran el error absoluto medio (MAE), el error cuadrático medio (RMSE), el porcentaje absoluto medio simétrico (sMAPE) y el porcentaje absoluto medio ponderado (WAPE) [2]. El MAE y el RMSE expresan el error en unidades de la variable objetivo, siendo el RMSE más sensible a errores grandes; las métricas porcentuales, por su parte, facilitan la interpretación relativa del error y permiten comparaciones entre periodos o productos con magnitudes distintas.

Como complemento, en algunos casos se reporta el coeficiente de determinación (R^2) como indicador global de capacidad explicativa. Sin embargo, su interpretación en series temporales debe hacerse con cautela, especialmente cuando existen saltos discretos o cambios de régimen [6]. Por ello, se recomienda analizar varias métricas de forma conjunta y considerar no solo el error promedio, sino también su estabilidad temporal y relevancia operativa.

4.3.2. Protocolos de validación y comparación entre modelos En series de tiempo la dependencia temporal impide el uso de esquemas de validación que mezclen aleatoriamente observaciones de distintos momentos [2]. En consecuencia, se emplean protocolos específicos que respetan el orden cronológico, tales como particiones entrenamiento–prueba, validación cruzada con ventanas crecientes (expanding window) y esquemas walk-forward sobre bloques recientes. El objetivo es aproximar el escenario real de uso en producción: entrenar con datos pasados y evaluar sobre periodos futuros no vistos, reduciendo el riesgo de sobreajuste, en particular en modelos altamente flexibles como LSTM o métodos de gradient boosting [6].

La comparación entre modelos puede abordarse de forma descriptiva contrastando métricas fuera de muestra bajo el mismo protocolo temporal, o mediante pruebas estadísticas de precisión predictiva. Una referencia clásica es la prueba de Diebold–Mariano, que evalúa si las diferencias en error entre dos modelos son estadísticamente significativas a partir de sus series

de pérdidas [10]. Independientemente de la prueba utilizada, la literatura converge en tres buenas prácticas: (i) definir benchmarks ingenuos, (ii) aplicar exactamente el mismo esquema temporal a todos los modelos y (iii) reportar de forma transparente métricas y decisiones derivadas de ellas, garantizando comparaciones justas y auditables.

En este estudio, basado en estas directrices, se realiza un análisis de medidas fuera de muestra según un protocolo temporal consistente, pero no se utilizan pruebas estadísticas formales como la prueba de Diebold-Mariano. La prueba se menciona solo como una referencia metodológica en la literatura, y se propone explícitamente como una línea de investigación futura, ya que la implementación de este método implicaría un análisis más extenso de las series de pérdidas y un marco comparativo más amplio.

4.4. Brecha identificada y aportes esperados

La revisión de la literatura muestra una amplia aplicación de modelos de series de tiempo a precios de materias primas, tipos de cambio y activos financieros, así como un creciente uso de modelos no lineales y métodos de *ensemble* en problemas de pronóstico [6]. Sin embargo, se observa una menor presencia de estudios que aborden insumos industriales específicos, como los sacos de papel kraft utilizados en la industria cementera, combinando datos transaccionales extraídos de sistemas ERP con variables exógenas públicas y comparando de forma sistemática modelos lineales y no lineales bajo protocolos de validación temporal realistas.

Esta brecha es particularmente notoria en el contexto latinoamericano, donde las empresas enfrentan alta volatilidad cambiaria y limitaciones de cobertura financiera, pero disponen cada vez más de datos históricos detallados [4]. La tesis se posiciona en este espacio, al integrar conceptos de gestión de compras bajo incertidumbre, técnicas avanzadas de series de tiempo y buenas prácticas de evaluación de modelos, con el objetivo de aportar evidencia aplicada sobre el uso de ARIMA, SARIMAX, LSTM y LightGBM para el pronóstico de precios de un insumo crítico en una cadena de suministro real.

4.5. Síntesis comparativa de la literatura

La Tabla 2 presenta una síntesis de estudios representativos que comparan modelos ARIMA/SARIMAX, LSTM e híbridos en diversos dominios. Los resultados son mixtos: ARIMA supera a LSTM en series con estructura predominantemente lineal (epidemiología, precios de materias primas estables), mientras que LSTM y modelos híbridos son superiores en series

con alta no linealidad y volatilidad (series agrícolas, materias primas con choques).

Estudio	Dominio	ARIMA/SARIMAX	LSTM	Híbrido	Mejor
Zhang (2022) [11]	Epidemiología	MAE 12.3	MAE 15.7	–	ARIMA
Mbah (2021) [12]	Piedra caliza	Acc 95.7 %	Acc 91.8 %	–	ARIMA
Xia (2024) [13]	Agrícola	RMSE 45.2	RMSE 13.5	–	LSTM
Liu (2023) [14]	Materias primas	0.082	0.074	0.065	Híbrido
Ray et al. (2023) [15]	Agrícola	R^2 0.82	R^2 0.89	0.936	Híbrido

Tabla 2. **Tabla resumen comparativa de modelos predictivos en la literatura.**

Resultados reportados en diferentes dominios de aplicación para modelos ARIMA/SARIMAX, LSTM y enfoques híbridos; las métricas específicas varían según el estudio original.

4.6. Metodología

Esta sección describe el diseño metodológico seguido para construir el conjunto de datos, especificar los modelos de pronóstico y evaluar su desempeño en un contexto de decisión de compras. Se integra en un solo bloque todo lo relacionado con datos, preprocesamiento, modelos, validación, métricas y reproducibilidad, evitando duplicaciones y manteniendo coherencia con los objetivos planteados y con el marco teórico [2, 6].

4.7. Enfoque general y diseño experimental

El estudio adopta un enfoque cuantitativo, observacional y retrospectivo, basado en datos históricos transaccionales y series exógenas públicas. El objetivo es comparar el desempeño fuera de muestra de cuatro familias de modelos de series de tiempo —ARIMA, SARIMAX, redes LSTM y LightGBM— para el pronóstico del precio de compra de sacos de papel kraft, bajo un protocolo de validación temporal riguroso [1, 7, 9].

Para articular la narrativa de resultados, el diseño experimental se organiza en cuatro fases, que corresponden a escenarios progresivos de complejidad metodológica:

- **Fase I: Modelos base sobre la serie original.** Evaluación inicial de ARIMA, SARIMAX y un LSTM base utilizando la serie de precios en su forma original, con irregularidad en el calendario y duplicados fecha–producto.

- **Fase II: LSTM con variables exógenas.** Extensión del LSTM base incorporando como entradas la TRM y precios accionarios de proveedores globales, manteniendo aún la estructura original de la serie.
- **Fase III: Transformación de datos y Gradient Boosting.** Reconstrucción de una serie diaria equiespaciada mediante LOCF, ingeniería de características sobre el objetivo y las exógenas, e implementación de LightGBM como modelo de *gradient boosting* [8, 9].
- **Fase IV: LSTM sobre la serie diaria transformada.** Entrenamiento de un LSTM utilizando la misma representación diaria enriquecida, para contrastar un modelo de aprendizaje profundo secuencial con el enfoque tabular de boosting.

Todas las fases comparten el mismo horizonte de pronóstico (un día adelante, $h = 1$) y el mismo esquema de partición temporal (90 % entrenamiento, 10 % prueba), de modo que las comparaciones entre modelos sean consistentes y libres de fuga de información [2].

4.8. Glosario de abreviaturas

Para facilitar la lectura y evitar ambigüedades, en esta subsección se presentan las principales siglas utilizadas a lo largo del documento:

- **ERP:** *Enterprise Resource Planning*. Sistema de planificación de recursos empresariales (por ejemplo, SAP MM).
- **TRM:** *Tasa Representativa del Mercado*. Tasa de cambio oficial COP/USD publicada por el Banco de la República.
- **LSTM:** *Long Short-Term Memory*. Arquitectura de redes neuronales recurrentes diseñada para modelar dependencias de largo plazo en series de tiempo.
- **ARIMA:** *Autoregressive Integrated Moving Average*. Modelo clásico de series de tiempo basado en componentes autorregresivos y de promedios móviles.
- **SARIMAX:** *Seasonal ARIMA with eXogenous regressors*. Extensión de ARIMA que incorpora estacionalidad y variables exógenas.
- **GBM / Gradient Boosting:** Método de ensamble basado en la construcción secuencial de árboles para minimizar el error de predicción.

- **LightGBM**: Implementación eficiente del método de *gradient boosting* basada en histogramas y optimizada para conjuntos de datos tabulares.
- **MAE**: *Mean Absolute Error*. Error absoluto medio.
- **RMSE**: *Root Mean Squared Error*. Raíz del error cuadrático medio.
- **sMAPE**: *Symmetric Mean Absolute Percentage Error*. Porcentaje absoluto medio simétrico.
- **WAPE**: *Weighted Absolute Percentage Error*. Error porcentual absoluto ponderado.
- **IQR**: *Interquartile Range*. Rango intercuartílico (Q3–Q1).
- **LOCF**: *Last Observation Carried Forward*. Técnica que rellena valores faltantes con la última observación disponible.
- **DM**: *Diebold–Mariano*. Prueba estadística para comparar precisión predictiva entre dos modelos.

4.9. Datos y fuentes de información

4.9.1. Datos internos: serie objetivo La variable objetivo y_t corresponde al precio neto de compra en COP de sacos de papel kraft de 50 kg, extraído del sistema ERP corporativo para el periodo 2014–2023. A partir del módulo de compras se consolidaron todas las órdenes asociadas al ítem de interés, identificando para cada día con transacción el precio unitario efectivo después de descuentos y ajustes.

La serie presenta un comportamiento característico de escalones: el precio se mantiene constante entre fechas de compra y se ajusta únicamente cuando se renegocian condiciones, lo que refleja la lógica contractual del insumo. Este patrón, habitual en insumos industriales negociados por lotes, condiciona la elección de transformaciones y modelos apropiados [2, 6].

En la depuración se eliminaron registros duplicados fecha–producto, correspondientes a múltiples líneas con el mismo ítem y día, que distorsionaban la señal temporal. Aproximadamente la mitad de los registros originales fueron identificados como duplicados y consolidados en un único precio representativo por fecha, manteniendo la trazabilidad de la fuente ERP.

Cuantitativamente, el conjunto de datos original contenía alrededor de 8,705 registros de compras de bolsas de papel kraft desde 2014 hasta 2023, obtenidos del módulo de compras del

sistema ERP (como el módulo SAP MM). Después de eliminar duplicados de fecha-producto, filtrar inconsistencias y descartar registros con información incompleta, se obtuvo un conjunto refinado de 3,567 observaciones válidas. Los datos diarios utilizados en diferentes partes del análisis se basan en esta serie agregada.

4.9.2. Datos externos: variables exógenas Como señales relacionadas con el frente de oferta se incorporaron series exógenas públicas:

- **TRM COP/USD diaria.** La tasa de cambio afecta directamente el costo de insumos importados o indexados a mercados internacionales, por lo que es un determinante clave del nivel de precios en COP.
- **Precios accionarios de proveedores globales de empaques/papel.** Se utilizaron los precios diarios en USD de compañías representativas del sector, como Klabin, Mondi y Smurfit Kappa, como *proxies* de la dinámica de oferta y de los choques en la cadena de valor del papel kraft.

La selección de estos factores se debe al hecho de que insumos como el papel kraft forman un costo de producción, hasta cierto punto, que está determinado por elementos globales, como los precios de la pulpa, la energía y la logística, y por la situación financiera de los principales actores en la cadena de suministro [4]. Específicamente, el tipo de cambio COP/USD captura el elemento de intercambio que acentúa o mitiga estos choques (en moneda local) y sus diferencias se transmiten, aunque con cierto retraso, al precio en pesos de los sacos de papel kraft. De manera similar, los precios de las acciones de las empresas Klabin, Mondi y Smurfit indican las expectativas agregadas del mercado con respecto a los márgenes de producción, la capacidad instalada, los precios de las materias primas y los posibles cuellos de botella logísticos, traducidos así como *proxies* para la presión de costos de suministro y aguas arriba, y por lo tanto pueden usarse para predecir ajustes en las condiciones de la negociación sobre el insumo estudiado.

Las series de TRM diaria y los precios accionarios se obtuvieron del Banco de la República de Colombia, utilizando los registros oficiales publicados en su módulo estadístico para el periodo 2014–2023 [16]. Todas las series exógenas se alinearon con el periodo de análisis 2014–2023, de forma que cada observación diaria del precio del saco pudiera asociarse de manera unívoca con el valor contemporáneo de las variables externas.

4.10. Preprocesamiento y control de fuga de información

El preprocesamiento tuvo dos objetivos principales: (i) construir una representación temporal coherente con la lógica de negocio (serie diaria por escalones) y (ii) garantizar que ninguna transformación introdujera información del futuro en el entrenamiento de los modelos.

4.10.1. **Reconstrucción de serie diaria** A partir de las fechas de compra se generó una serie diaria equiespaciada para todo el periodo de estudio. En días sin transacción, el precio se imputó mediante el esquema *last observation carried forward* (LOCF), manteniendo constante el último precio negociado. Esta estrategia respeta la morfología escalonada del proceso de fijación de precios: la ausencia de compras no implica un cambio en el precio contractual, sino la continuidad de las condiciones vigentes [6].

4.10.2. **Partición temporal y transformaciones** Antes de aplicar cualquier transformación que pudiera aprovechar regularidades temporales, la serie se dividió en dos bloques en orden cronológico:

- 90 % de las observaciones iniciales para calibración y validación interna.
- 10 % más reciente como conjunto de prueba (*holdout*) exclusivo para evaluación final.

Todas las transformaciones que dependen del tiempo (rezagos, promedios móviles, retornos y volatilidades) se calcularon utilizando estrictamente información disponible hasta $t - 1$. En particular:

- Los rezagos del objetivo y de las exógenas se definieron como $y_{t-\ell}$ y $x_{t-\ell}$, con ℓ en un conjunto finito de retardos económicamente plausibles.
- Los promedios móviles históricos $y_{t,7}$ y $y_{t,14}$ se calcularon sin incluir la observación de t , de forma que resumen el nivel reciente sin romper la estructura por tramos.
- Los retornos logarítmicos y medidas de volatilidad de las exógenas se construyeron a partir de ventanas puramente históricas.

Los escaladores utilizados para normalizar variables (en los modelos que lo requieren, como LSTM) se ajustaron únicamente con el subconjunto de entrenamiento y se aplicaron

después al conjunto de prueba, evitando cualquier fuga de información [2]. En conjunto, este *pipeline* garantiza que cada etapa del procesamiento respete el orden temporal y que los modelos se evalúen en condiciones realistas de uso.

4.11. Ingeniería de características

La ingeniería de características se diseñó para capturar tanto la estructura de escalones del precio como las señales de choque y tendencia provenientes de las exógenas, manteniendo siempre la coherencia temporal [6].

4.11.1. Lags y suavizadores del objetivo Para la serie objetivo y_t se incluyeron:

- Rezagos $y_{t-\ell}$ con $\ell \in \{1, 5, 7, 14, 22, 30\}$, que aportan contexto de corto y mediano plazo.
- Promedios móviles históricos $y_{t,7}$ y $y_{t,14}$, calculados sin usar la observación de t , que resumen el nivel reciente sin romper la estructura por tramos.

No se aplicaron retornos al objetivo debido a su naturaleza escalonada: diferencias o retornos generarían largas rachas de ceros y picos esporádicos poco informativos para los modelos basados en árboles.

4.11.2. Retornos y volatilidad de variables exógenas Las exógenas se representaron mediante:

- Retornos logarítmicos a distintos horizontes $r_{j,t,h} = \log \left(\frac{x_{j,t}}{x_{j,t-h}} \right)$ para $h \in \{1, 5\}$, donde j indexa cada serie (TRM, Klabin, Mondi, Smurfit). Estos retornos son insensibles a la escala y capturan choques relativos recientes.
- Volatilidad reciente, definida como la desviación estándar de los retornos diarios en una ventana de 10 días, $\sigma_{j,t,10}$, que resume el nivel de incertidumbre en cada exógena.

La intuición económica es que aumentos persistentes o alta volatilidad en TRM o en los *proxies* de oferta elevan la probabilidad de renegociación y ajustes de precio en los sacos de papel.

4.11.3. Variables de calendario Finalmente, se incluyeron variables de calendario (mes, trimestre y dummies de año) para capturar potenciales efectos de estacionalidad contractual o patrones asociados a revisiones periódicas de precios, siguiendo prácticas habituales en pronóstico de negocios [2].

4.12. Modelos y especificaciones

Se implementaron cuatro familias de modelos bajo un protocolo común.

4.12.1. Modelos ARIMA y SARIMAX Los modelos $ARIMA(p, d, q)$ y $SARIMAX(p, d, q)(P, D, Q)_s$ se ajustaron sobre la serie original irregular en las Fases I–II y sobre la serie diaria transformada en los experimentos posteriores, con y sin variables exógenas según correspondía, utilizando criterios de información (AIC/BIC) y diagnóstico de residuales para seleccionar órdenes apropiados [1, 2]. En SARIMAX, las exógenas se incluyeron de forma contemporánea y con rezagos económicamente plausibles, evitando configuraciones que implicaran usar información del futuro.

4.12.2. Modelo LSTM Las redes LSTM se entrenaron sobre ventanas de entrada de longitud entre 30 y 90 días, que incluyen el historial reciente del precio y las exógenas [7]. La arquitectura empleó:

- Una o dos capas LSTM con un número moderado de unidades.
- Regularización mediante *dropout* y *early stopping* sobre la pérdida de validación.
- Normalización previa de las entradas con estadísticas calculadas solo en el conjunto de entrenamiento.

La Figura 1 ilustra el esquema de ventaneo: para cada ventana de entrada se genera un pronóstico un día adelante, respetando el orden temporal.

4.12.3. Modelo LightGBM El modelo de *gradient boosting* se implementó con LightGBM, utilizando la representación tabular construida a partir de rezagos, retornos, volatilidades y variables de calendario [8, 9]. Se exploraron hiperparámetros como profundidad máxima de los árboles, tasa de aprendizaje, número de estimadores y mínimos de datos por hoja,

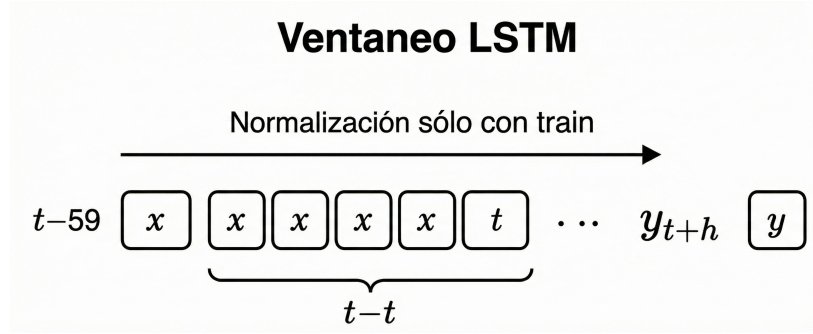


Fig. 1: Esquema de ventaneo para el modelo LSTM: para cada ventana de entrada se genera un pronóstico un día adelante, respetando el orden temporal.

mediante búsqueda con validación cruzada temporal (*TimeSeriesSplit*) sobre el bloque de calibración.

LightGBM no requiere escalado de variables y es capaz de capturar relaciones no lineales y efectos umbral, lo que resulta adecuado para series con saltos discretos y señales exógenas complejas.

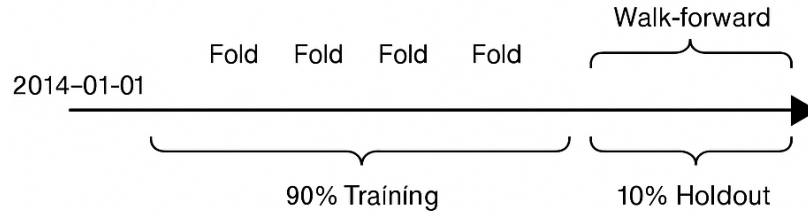
4.13. Esquema de validación temporal

El esquema de validación se diseñó para aproximar el uso real de los modelos en producción y asegurar comparaciones justas entre enfoques [2, 6].

- **Corte 90/10.** Se reservó el 10 % más reciente de la serie como conjunto *holdout* para evaluación final fuera de muestra.
- **Validación cruzada creciente en el 90 %.** En el bloque de calibración se aplicó una validación cruzada con ventanas crecientes (folds acumulativos), donde cada fold añade un tramo adicional de datos en el tiempo.
- **Esquema *walk-forward* en el *holdout*.** En el 10 % final se utilizó un esquema *walk-forward*: el modelo se recalibra secuencialmente a medida que se avanza en el tiempo, generando pronósticos un día adelante con la información disponible hasta cada punto.

La Figura 2 resume gráficamente la partición temporal y el flujo de entrenamiento-validación-prueba. Este esquema se aplicó de forma consistente a todos los modelos, de modo que las diferencias de desempeño puedan atribuirse al modelo y no al procedimiento

Esquema de partición temporal



Esquema de partición temporal

Fig. 2: Esquema de partición temporal 90/10 y validación cruzada con ventanas crecientes.

de evaluación. En esta versión del trabajo no se aplicaron pruebas formales de comparación como la de Diebold–Mariano; la comparación entre modelos se realiza mediante la magnitud de las métricas de error en el conjunto *holdout* y el análisis gráfico de residuales y pronósticos, siguiendo recomendaciones generales de la literatura [6].

4.14. Métricas de evaluación

La evaluación de desempeño se realizó con un conjunto de métricas complementarias, recomendadas en la literatura de pronóstico de series de tiempo [2, 6]:

- **Error absoluto medio (MAE).** Mide el error promedio en las mismas unidades del precio (COP), interpretado como desviación típica esperada.
- **Error cuadrático medio (RMSE).** Penaliza más los errores grandes y es útil para detectar modelos que ocasionalmente cometen errores extremos.
- **Porcentaje absoluto medio simétrico (sMAPE).** Facilita la comparación relativa de errores en distintos niveles de precio.
- **Porcentaje absoluto medio ponderado (WAPE).** Resume la proporción del error absoluto respecto a la suma de los valores observados, útil para comunicar el impacto agregado sobre el costo total de compras.
- **Coefficiente de determinación (R^2).** Se reporta únicamente para algunos modelos como indicador de capacidad explicativa global, interpretándolo con cautela en el contexto de series de tiempo.

Adicionalmente, se comparó el desempeño de cada modelo frente a líneas base ingenuas (persistencia y promedios móviles simples), con el objetivo de verificar que los modelos propuestos aportan valor predictivo más allá de reglas triviales [6].

Establecer estas referencias de comparación en el análisis se complementó con la referencia a las buenas prácticas de reporte recomendadas por la literatura para (i) asegurar la transparencia en la definición de indicadores, (ii) garantizar que todos los modelos fueran evaluados bajo el mismo esquema de validación temporal, y (iii) apoyar visualmente el resultado con gráficos de valores observados vs. predichos y un análisis residual básico. La sección de Resultados implementa estas prácticas para mostrar para cada modelo tanto los valores numéricos de las métricas como las figuras que ilustran su comportamiento en el conjunto de prueba.

4.15. Reproducibilidad y trazabilidad

Para garantizar la reproducibilidad del estudio se adoptaron las siguientes prácticas, en línea con recomendaciones recientes en ciencia de datos aplicada al pronóstico [6]:

- Versionamiento del código y de los scripts de preprocesamiento, entrenamiento y evaluación.
- Fijación de semillas aleatorias en los componentes estocásticos (inicialización de redes neuronales, particiones internas de validación).
- Registro sistemático de hiperparámetros, métricas y artefactos generados (modelos entrenados, escaladores, conjuntos de características).
- Documentación de las versiones de librerías utilizadas (*p. ej.*, `pandas`, `statsmodels`, `scikit-learn`, `lightgbm`, `tensorflow/keras`).

Estas medidas permiten auditar el proceso de punta a punta y facilitan la replicación del análisis por parte de la organización o de evaluadores externos.

5. Resultados

De acuerdo con el diseño experimental proporcionado en la Sección 4, esta sección presenta los resultados organizados en cuatro fases sucesivas que siguen exactamente el flujo metodológico planteado. En primer lugar, se reportan el diagnóstico exploratorio de la serie y el desempeño de los modelos base sobre la serie original (Fase I y Fase II). Posteriormente, se analiza el impacto de reconstruir una serie diaria equiespaciada y de aplicar ingeniería de características para entrenar el modelo de Gradient Boosting (Fase III). Finalmente, se presenta el desempeño del modelo LSTM entrenado sobre esta representación diaria transformada (Fase IV) y se realiza una comparación global entre todos los enfoques evaluados.

5.1. Análisis exploratorio de datos

Una serie de precios netos de compra exhibe un comportamiento claramente escalonado: el precio solo cambia en los días de transacción y permanece constante entre compras debido a la lógica contractual de la entrada. Desde el período 2014-2023, cuando el nivel de precios aumentó para los períodos observados en la Figura 3, hay (i) períodos de picos aislados al principio, (ii) períodos más largos de poca variación, y (iii) una tendencia reciente al alza. Esta heterogeneidad apunta a una dinámica de saltos discretos y períodos de relativa estabilidad.

El precio diario durante el período 2014-2023 osciló aproximadamente entre 457 y 1,353 COP en términos descriptivos, con una mediana cercana a 652 COP y un rango intercuartil de 139 COP. De la serie reconstruida, se detectaron aproximadamente 69 cambios de nivel, con un intervalo mediano de 16 días entre renegociaciones, lo que respalda la estabilidad a largo plazo observada con esta entrada. También hay momentos durante los escalones en los que se limitan a niveles relativamente bajos (especialmente, entre 2014 y 2016), y más recientemente, una fase donde el nivel de precios comienza a subir gradualmente a niveles más altos (específicamente, entre 2021 y 2023). Como se ve en la Figura 3, estas tendencias acentúan la importancia de crear modelos que consideren fluctuaciones discretas y que traten el período más reciente como el más apropiado para elegir comprar algo.

Además, para probar diferentes señales externas al cambio de nivel, se investiga el gráfico de comovimiento entre el precio de los sacos y la serie exógena, así como correlaciones simples. Especialmente, se encontraron correlaciones positivas moderadas entre el nivel de precios y la TRM, y asociaciones positivas regulares entre el precio y los rendimientos acumulados de algunas acciones de proveedores en varios momentos recientes, lo que apoya la interpreta-

ción de estas como señales del lado de la oferta. Estos patrones fomentan la implementación sistemática de estas variables exógenas en las Fases II-IV del estudio. Operativamente, los largos escalones muestran que un precio negociado tiende a mantenerse durante unos días o semanas hasta la próxima compra, mientras que los saltos indican renegociaciones o choques externos relevantes (como cambios en el tipo de cambio o costo internacional). Este diagnóstico exploratorio, combinado con la evidencia previamente descrita y exógena, llevó al rediseño de la representación temporal (serie diaria con LOCF) y también aumentó la presencia de señales exógenas en otras etapas del análisis.

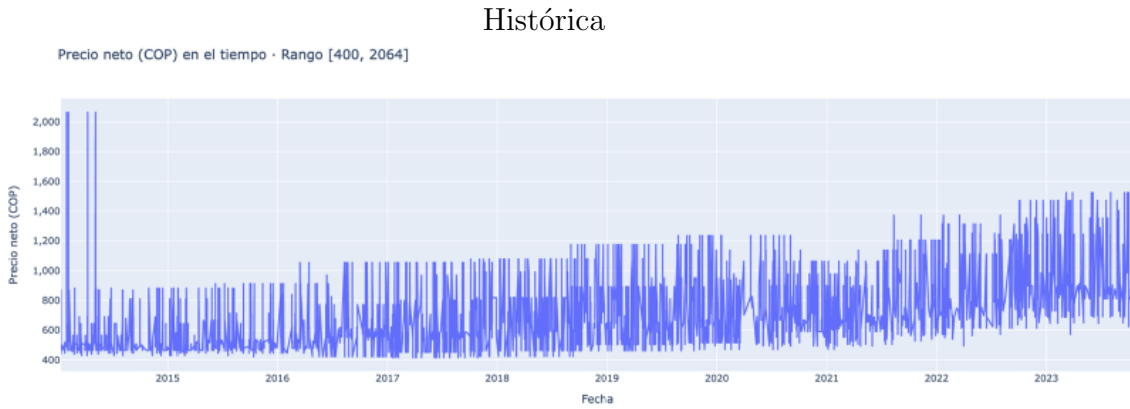


Fig. 3: Serie histórica del precio neto de compra de sacos de papel kraft (2014–2023).

5.2. Fase I: modelos base

En la Fase I se evaluaron tres modelos base sobre la serie original de precios: ARIMA, SARIMAX y un LSTM “simple” sin variables exógenas. La comparación se realizó en el 10 % más reciente de la serie (conjunto de prueba), bajo el protocolo de partición temporal descrito en la Metodología. La Tabla 3 resume los resultados cuantitativos.

ARIMA alcanzó un MAE de 96.70 y un sMAPE de 10.03 %, mientras que SARIMAX obtuvo un MAE de 132.66 y un sMAPE de 13.40 %. Estos valores evidencian errores absolutos y relativos altos para efectos operativos, con predicciones que no capturan adecuadamente los cambios de nivel. La Figura 4 ilustra el comportamiento de ARIMA: el modelo tiende a suavizar la serie, no alinea bien los escalones y subestima sistemáticamente ciertos tramos de precio, confirmando la limitación de enfoques lineales para este tipo de serie escalonada.

El LSTM base, entrenado sobre ventanas de la serie objetivo sin exógenas, mejora parcialmente el desempeño respecto a los modelos lineales: MAE de 81.87 y sMAPE de 9.59 %.

Sin embargo, la magnitud del error absoluto sigue siendo elevada frente a los niveles de precio relevantes para negociación, por lo que el modelo no alcanza aún la precisión operativa deseada. Esta fase sirve como línea base para evaluar el aporte de variables exógenas y transformaciones más ricas en las fases siguientes.

Tabla 3. Comparación de desempeño (modelos base y LSTM con exógenas, conjunto de prueba).

Modelo	MAE	RMSE	sMAPE
ARIMA	96.70	118.16	10.03 %
SARIMAX	132.66	161.35	13.40 %
LSTM (base)	81.87	83.49	9.59 %
LSTM (TRM + Smurfit)	123.99	168.71	16.68 %



Fig. 4: Comportamiento del modelo ARIMA en el conjunto de prueba: valores observados vs. pronosticados.

5.3. Fase II: LSTM con variables exógenas

En la Fase II se incorporaron variables exógenas asociadas al frente de oferta y al entorno macro: la TRM COP/USD y los precios accionarios de referentes/proveedores globales (Klabin, Mondi y Smurfit). Desde la lógica de negocio, la TRM captura la presión cambiaria sobre insumos importados, mientras que las acciones de proveedores reflejan condiciones de mercado y costos en la cadena de papel y empaques. En particular, Smurfit tiene un rol central como proveedor clave, por lo que se exploraron configuraciones que priorizan esta señal.

Se entrenaron varias arquitecturas LSTM con combinaciones de exógenas, manteniendo el mismo protocolo de validación temporal de la Fase I. La mejor variante en términos de MAE fue la que combinó la TRM y el precio de la acción de Smurfit como regresores adicionales. En el conjunto de prueba (10 % más reciente), esta especificación LSTM (TRM + Smurfit) obtuvo un MAE de 123.99, un RMSE de 168.71 y un sMAPE de 16.68 %, tal como se resume en la Tabla 3.

Estos resultados son peores que los del LSTM base e incluso que los de ARIMA en términos de MAE y sMAPE. Es decir, en esta configuración particular la inclusión directa de exógenas no mejoró el pronóstico y, de hecho, degradó el desempeño. Una posible explicación es que, al trabajar sobre la serie original poco transformada, la red tiende a “memorizar” patrones ruidosos en lugar de extraer señales robustas de las variables externas. Esta fase muestra que la introducción de exógenas en LSTM no garantiza mejoras automáticas y que se requieren transformaciones más cuidadosas para aprovecharlas.

5.4. Fase III: transformación de datos y Gradient Boosting

La Fase III transforma el problema en un problema de regresión tabular de la serie diaria transformada junto con el conjunto de características especificadas en la Sección 4. En lugar de tratar con la serie irregular original, el modelo LightGBM recibe como entrada una representación diaria que incluye rezagos del precio objetivo, rendimientos y medidas de volatilidad de las variables exógenas, además de variables de calendario. El objetivo de esta fase es evaluar hasta qué punto este cambio de representación y el uso de un método de *gradient boosting* permiten reducir de manera significativa los errores obtenidos en las fases anteriores.

En el conjunto de prueba (10 % más reciente), LightGBM alcanza un MAE de 8.03 COP, con métricas relativas $sMAPE = 0.92 \%$ y $WAPE = 0.90 \%$. Esto representa una reducción sustancial del error frente a los modelos evaluados en las fases anteriores, cuyos MAE se ubicaban en el orden de 80–130 COP (ARIMA, SARIMAX y LSTM sobre la serie original). En la Figura 5 se observa que el modelo replica con precisión los tramos constantes de la serie y que los errores más visibles se concentran alrededor de algunos cambios bruscos de nivel, donde pequeñas desviaciones en la fecha del salto generan discrepancias puntuales en el holdout. Aun así, la magnitud agregada del error se mantiene en un rango que hace que las predicciones sean operativamente útiles para apoyar decisiones de compra, especialmente si se interpretan como una señal adicional sobre el nivel esperado del precio y no como un valor exacto.

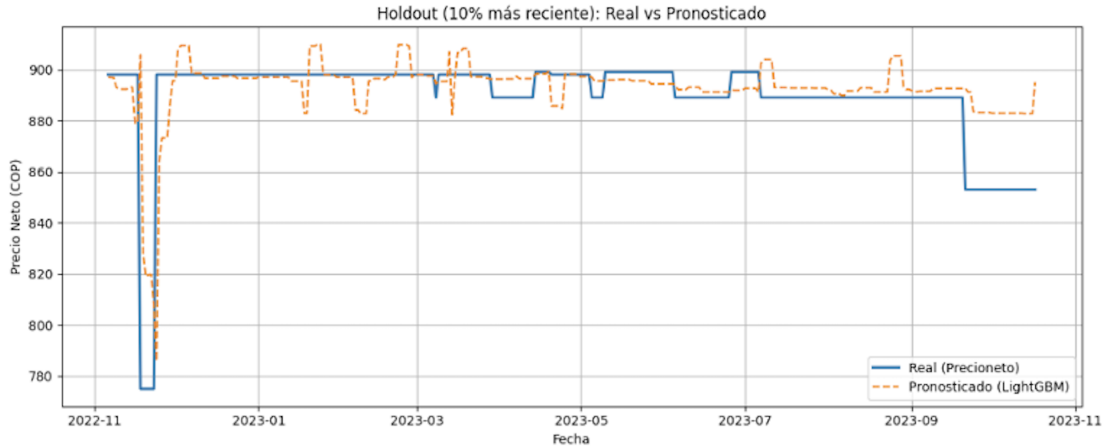


Fig. 5: Desempeño de LightGBM en el conjunto de prueba (10 % más reciente): valores observados vs. pronosticados.

5.5. Fase IV: LSTM con serie diaria transformada

La Fase IV retoma la arquitectura LSTM, pero ahora sobre la serie diaria transformada con LOCF, es decir, sobre una representación que respeta explícitamente los escalones y elimina duplicados de la serie original. El objetivo es evaluar si, al partir de una estructura temporal más limpia y estable, la red recurrente puede capturar dependencias de largo plazo y mejorar aún más la precisión obtenida con Gradient Boosting.

Bajo el mismo esquema de partición 90/10 y validación temporal, el LSTM con serie diaria transformada alcanzó en el conjunto de prueba un MAE de 4.57, un RMSE de 12.40 y un R^2 de 0.96. Estos valores implican una reducción adicional de la mitad del MAE respecto a LightGBM y un ajuste muy alto en términos de varianza explicada. La Figura 6 ilustra el comportamiento del modelo: se observan trayectorias pronosticadas que siguen de forma precisa los tramos constantes y anticipan caídas relevantes de precio con desplazamientos temporales acotados.

En síntesis, la Fase IV confirma que la combinación de (i) una representación temporal coherente con la lógica contractual (LOCF) y (ii) un modelo secuencial capaz de explotar dependencias de mayor longitud permite capturar mejor la dinámica del precio del insumo. No obstante, el costo computacional y la sensibilidad a hiperparámetros son mayores que en el caso de LightGBM, lo que introduce un compromiso entre precisión y simplicidad operativa.

Es importante notar que las Figuras 5 y 6 no cubren exactamente el mismo tramo tem-

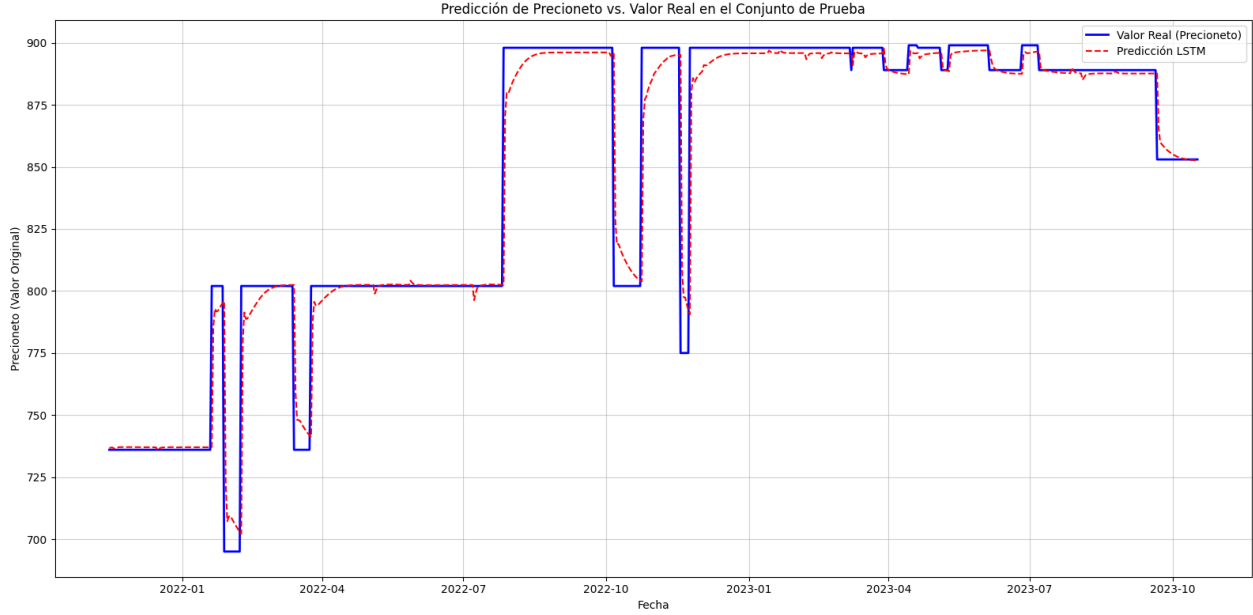


Fig. 6: Desempeño del LSTM con serie diaria transformada en el conjunto de prueba: valores observados vs. pronosticados.

poral. En todos los casos, las métricas de desempeño se calculan sobre el mismo conjunto de prueba (el último 10 % de la serie), pero para LightGBM se grafica la totalidad del periodo de *holdout*, mientras que en el caso del LSTM se presenta un acercamiento al subtramo más reciente del conjunto de prueba, donde la serie exhibe una mayor concentración de escalones y donde las decisiones de compra son operativamente más relevantes. Esta decisión de visualización busca equilibrar legibilidad y foco analítico: el rendimiento global queda cuantificado por las métricas sobre todo el conjunto de prueba, mientras que las figuras ilustran con mayor claridad el comportamiento típico de los modelos en segmentos representativos de dicho periodo.

5.6. Comparación global

La Tabla 4 resume de forma consolidada el desempeño de los principales modelos evaluados en las distintas fases, usando el MAE como métrica comparativa homogénea en el conjunto de prueba (10 % más reciente). Se incluyen los modelos lineales (ARIMA y SARIMAX), el LSTM base, el LSTM con exógenas (TRM + Smurfit), el modelo de Gradient Boosting (LightGBM) y el LSTM con serie diaria transformada.

En términos de MAE, el modelo con mejor desempeño global es el LSTM con serie diaria

transformada ($MAE = 4.57$), seguido por LightGBM ($MAE = 8.03$). Los modelos lineales (ARIMA y SARIMAX) y el LSTM base quedan claramente rezagados, con errores absolutos un orden de magnitud mayores. La variante LSTM con exógenas sobre la serie original presenta el peor desempeño, lo que refuerza la conclusión de que la forma de representar la serie y de transformar las variables externas es tan importante como la elección de la familia de modelos.

Desde la perspectiva de uso en la organización, la tesis considera al LSTM con serie diaria transformada como el modelo de referencia en términos de precisión de pronóstico, y a LightGBM como una alternativa robusta y más simple de implementar y mantener en producción. En consecuencia, las recomendaciones operativas de compra se pueden apoyar en el LSTM como escenario principal y en LightGBM como modelo de respaldo y validación cruzada de señales.

Tabla 4. Comparación de desempeño en conjunto de prueba (10 % más reciente).

Modelo	MAE (COP)
ARIMA	96.70
SARIMAX	132.66
LSTM (base)	81.87
LSTM (TRM + Smurfit)	123.99
LightGBM	8.03
LSTM (serie diaria)	4.57

6. Conclusiones

Este estudio propuso resolver un problema de predicción del costo de la adquisición de sacos de papel kraft en una situación de alta incertidumbre y falta de instrumentos de cobertura financiera. Utilizando datos transaccionales internos (2014-2023) y variables exógenas públicas (TRM y precios de acciones de proveedores globales), se desarrolló una canalización de datos reproducible y se compararon los modelos ARIMA, SARIMAX, LSTM y LightGBM bajo el mismo protocolo temporal. A continuación, se presentan los principales hallazgos del análisis con importantes contribuciones, implicaciones para la gestión de compras, así como las principales limitaciones y direcciones para trabajos futuros.

6.1. Conclusiones generales

La tesis primero demuestra la validez de que la representación de la serie de precios no es menos importante que la elección del modelo. La limpieza de los registros duplicados de fecha-producto y la reconstrucción de la serie diaria con la técnica de Última Observación Llevada Adelante (LOCF) fueron esenciales para expresar la naturaleza escalonada del precio: largos tramos invariables con saltos discretos correspondientes a renegociaciones. Para la serie irregular original, tanto el modelo lineal como el LSTM base tienen altos errores absolutos y dificultades para encontrar los cambios de nivel con precisión; para la serie diaria transformada, los modelos no lineales pueden lograr ganancias sustanciales en la precisión.

En segundo lugar, la comparación metódica de los modelos indica que ARIMA y SARIMAX (modelos lineales) no son apropiados para series de esta naturaleza. A pesar de calibraciones diligentes, sus errores relativos en el conjunto de prueba permanecen en rangos que no ayudan a facilitar decisiones de compra precisas. El simple LSTM estándar mejora estos hallazgos hasta cierto punto, pero aún no logra la precisión esperada.

En tercer lugar, dos métodos no lineales son evidentes dentro de la serie diaria transformada y señales exógenas: (i) un modelo de Gradient Boosting con LightGBM presentado con rezagos de precio, retornos y volatilidad de las variables exógenas y del calendario, que reduce los errores relativos a menos del 1 %; (ii) un LSTM entrenado en la misma serie diaria enriquecida con características resulta en una mejora de los datos de salida que provoca una reducción adicional de aproximadamente la mitad del error absoluto en comparación con LightGBM y un ajuste extremadamente bueno en términos de varianza explicada para el último conjunto.

Finalmente, dado que este documento considera la precisión de las previsiones como un propósito importante, tomamos el LSTM con series diarias transformadas como un modelo de referencia del rendimiento predictivo. LightGBM sigue siendo una alternativa extremadamente poderosa que es más fácil de implementar y mantener en producción, al tiempo que proporciona un punto de contraste transparente. El enfoque práctico recomendado es adoptar el LSTM diario como fuente de previsión y LightGBM como modelo de respaldo y validación cruzada de las señales, compartiendo el mismo flujo de datos y el mismo esquema de validación temporal.

6.2. Contribuciones principales

Desde la perspectiva metodológica y aplicada, el trabajo deja tres contribuciones centrales:

- **Pipeline de datos anti-fuga para series escalonadas de compras.** Se construyó un flujo de datos reproducible, comenzando desde el ERP, sin duplicados, una serie diaria que se alinea con la lógica contractual y definiendo características (retardos, promedios móviles, retornos, volatilidad, calendario) únicamente con información disponible hasta $t - 1$. Este pipeline puede repetirse para otras entradas y también prepara bien el escenario para trabajos posteriores dentro de la organización.
- **Evidencia comparativa de modelos en un caso real de insumo industrial.** La tesis proporciona una comparación sistemática de ARIMA, SARIMAX, LSTM y LightGBM en el contexto de un problema real de compras industriales, demostrando por qué los modelos lineales son menos ventajosos que los métodos no lineales cuando una serie tiene estructura de pasos y se incorporan variables exógenas relevantes. El resultado más relevante no es simplemente identificar un “modelo ganador”, sino más bien demostrar que LOCF, señales de cambio y modelos no lineales juntos reducen el error a grados donde el pronóstico es accionable.
- **Traducción del pronóstico a un lenguaje útil para compras.** Más allá de los indicadores estadísticos, el trabajo describe cómo interpretar métricas como MAE, WAPE y sMAPE en términos de su impacto en el costo total de compra, y ver los pronósticos diarios como indicadores que pueden ayudar al planificar pedidos y negociar con proveedores. Esto sitúa los resultados en el contexto del lenguaje empresarial y facilita la apropiación de la herramienta por parte del departamento de compras.

6.3. Implicaciones para la gestión de compras

Los resultados producidos son de importancia inmediata para la gestión del riesgo de precios en el campo de las compras. En ausencia de instrumentos de cobertura financiera para este insumo, un pronóstico diario con un error relativo bajo puede ser tratado como otra señal cuantitativa que puede emplearse para informar las etapas de planificación y negociación.

Las aplicaciones gerenciales a un primer nivel abarcan la conversión de métricas, como MAE o WAPE, en magnitudes monetarias agregadas, con referencia al volumen total de compra del período. Permite evaluar si el error típico del modelo está dentro de rangos aceptables desde un punto de vista financiero, y cuantificar aproximadamente el costo esperado asociado con decisiones basadas en pronósticos imperfectos. Además, evaluar la estabilidad del error a lo largo del tiempo es crítico para determinar si el ruido producido por las señales del modelo es confiable: un error bajo y altamente inestable impedirá el uso operativo del modelo; un error pequeño y estable permitirá tomar decisiones recurrentes.

Bajo estas consideraciones, las previsiones obtenidas pueden apoyar decisiones operativas tales como:

- Identificar momentos en los que el modelo anticipa estabilidad o disminuciones del precio y, por tanto, informar la conveniencia de posponer compras o fortalecer la negociación para capturar ventanas temporales favorables.
- Detectar tendencias al alza sostenidas que sugieran adelantar pedidos, renegociar condiciones o revisar cláusulas contractuales para mitigar impactos de precio.
- Complementar la experiencia del comprador y las señales cualitativas del mercado con una estimación cuantitativa que integra el efecto de factores externos, como la TRM y el comportamiento de proveedores globales.

Un segundo nivel —propuesto explícitamente como trabajo futuro— sería formalizar reglas de decisión basadas en el pronóstico (por ejemplo, políticas como “comprar/esperar/renegociar”) y evaluar su impacto a través de simulaciones históricas. Este enfoque permitiría pasar de la señal predictiva a un sistema de apoyo a la decisión más automatizado orientado hacia resultados económicos.

En general, la tesis muestra que la función de compras puede avanzar hacia una capacidad más basada en datos, donde las decisiones ya no se basan únicamente en promedios históricos recientes o intuición, sino que incorporan sistemáticamente pronósticos validados bajo

rigurosos protocolos temporales. La contribución radica no solo en el modelo ganador, sino en demostrar la viabilidad de integrar datos internos y externos en un marco cuantitativo que se alinea con las necesidades reales de negociación y planificación.

6.4. Limitaciones del estudio

Para situar adecuadamente el alcance de las conclusiones, es necesario reconocer varias limitaciones:

- En primer lugar, el análisis se basa en una única entrada (sacos de papel kraft de 50 kg) y un horizonte de vista a corto plazo (un día por delante). Si bien su metodología puede aplicarse conceptualmente a otros materiales y horizontes temporales, esta generalización no se verifica empíricamente en la tesis.
- En segundo lugar, los modelos fueron evaluados con métricas de error estimadas en una muestra reciente, así como utilizando un protocolo temporal consistente, aunque no se realizaron pruebas estadísticas formales para comparar la precisión entre los modelos. Las afirmaciones de superioridad relativa están respaldadas por la escala y consistencia de estas métricas, no por pruebas de hipótesis.
- En tercer lugar, el estudio se basó en datos de una empresa específica y en el contexto de una cadena de suministro particular y una política de compras. La extrapolación a otras organizaciones o industrias debe hacerse con la debida precaución y comparación con nuevas investigaciones empíricas cuando sea posible.
- Finalmente, aunque se discuten las posibles aplicaciones operativas de las previsiones (por ejemplo, apoyo a la decisión sobre mover o posponer la adquisición), esta tesis no proporcionó una estructura de decisión integrada, un esquema de prueba de políticas retrospectivas o un sistema de alerta automatizado formalizado para su uso en los procesos de la empresa. Las limitaciones no son contraproducentes para los resultados identificados, sino que más bien limitan el alcance de las afirmaciones y destacan áreas identificables para el desarrollo.

Estas limitaciones no invalidan los resultados, pero sí acotan el alcance de las afirmaciones y señalan espacios claros para continuar el desarrollo.

6.5. Líneas de trabajo futuro

Finalmente, la tesis abre varias líneas de trabajo que pueden potenciar el valor generado:

- **Extensión a otros insumos y horizontes.** Utiliza una lógica de modelo y una canalización de datos similar en otros materiales críticos de la cadena de suministro y prueba largos períodos de pronóstico (por ejemplo, semanas o meses) y compara el equilibrio entre anticipación y precisión.
- **Profundización en la comparación formal de modelos.** En trabajos futuros, en particular, proporciona cálculos estadísticos apropiados para la comparabilidad de errores en modelos y esquemas de backtesting con ventanas móviles para estudiar la estabilidad del rendimiento entre diferentes subperíodos.
- **Diseño e implementación de reglas de decisión y sistema de alertas.** Una línea de trabajo futuro consistiría en implementar estos pronósticos en políticas de gasto reales (reglas del tipo “comprar / esperar / renegociar”) y evaluar sus efectos mediante simulaciones históricas. Sobre esa base, sería posible diseñar e implementar un sistema de alertas que pueda integrarse en las herramientas de trabajo diarias del área de compras.
- **Exploración de variantes de modelos.** Explora otros modelos (por ejemplo, variantes híbridas o métodos de boosting con pérdidas cuantílicas) que permitan no solo obtener un pronóstico puntual, sino también bandas de incertidumbre útiles para utilizar en la planificación de escenarios.

Colectivamente, estas líneas pretenden convertir los resultados de la tesis en una capacidad operacionalizada que esté integrada: a partir del ejercicio de comparación de modelos de un estudio de caso único, para convertirse en una parte arraigada del proceso de planificación de compras y gestión del riesgo de precios en la empresa.

REFERENCIAS

- [1] G. E. P. Box, G. M. Jenkins, G. C. Reinsel y G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5.^a ed. Hoboken, NJ: Wiley, 2015, ISBN: 9781118675021.
- [2] R. J. Hyndman y G. Athanasopoulos, *Forecasting: Principles and Practice*, 2.^a ed. Melbourne: OTexts, 2018. dirección: <https://otexts.com/fpp2/>.
- [3] A. J. Van Weele, *Purchasing and Supply Chain Management: Analysis, Strategy, Planning and Practice*, 5.^a ed. Andover: Cengage Learning, 2010, ISBN: 9781408018965.
- [4] M. Christopher y B. Gaudenzi, «Managing Risks in Sustainable Supply Chains,» *Sinergie Italian Journal of Management*, vol. 33, n.º 96, págs. 65-82, 2015. DOI: 10.7433/s96.2015.04.
- [5] L. M. Ellram, «Total Cost of Ownership: An Analysis Approach for Purchasing,» *International Journal of Physical Distribution & Logistics Management*, vol. 25, n.º 8, págs. 4-23, 1995. DOI: 10.1108/09600039510099928.
- [6] S. Makridakis, E. Spiliotis y V. Assimakopoulos, «Statistical and Machine Learning Forecasting Methods: Concerns and Ways Forward,» *PLOS ONE*, vol. 13, n.º 3, e0194889, 2018. DOI: 10.1371/journal.pone.0194889.
- [7] S. Hochreiter y J. Schmidhuber, «Long Short-Term Memory,» *Neural Computation*, vol. 9, n.º 8, págs. 1735-1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.
- [8] J. H. Friedman, «Greedy Function Approximation: A Gradient Boosting Machine,» *Annals of Statistics*, vol. 29, n.º 5, págs. 1189-1232, 2001. DOI: 10.1214/aos/1013203451.
- [9] G. Ke et al., «LightGBM: A Highly Efficient Gradient Boosting Decision Tree,» en *Advances in Neural Information Processing Systems*, vol. 30, 2017, págs. 3146-3154. dirección: <https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree>.
- [10] F. X. Diebold y R. S. Mariano, «Comparing Predictive Accuracy,» *Journal of Business & Economic Statistics*, vol. 13, n.º 3, págs. 253-263, 1995. DOI: 10.1080/07350015.1995.10524599.
- [11] R. Zhang, H. Song, Q. Chen, Y. Wang, S. Wang e Y. Li, «Comparison of ARIMA and LSTM for prediction of hemorrhagic fever at different time scales in China,» *PLOS ONE*, vol. 17, n.º 1, e0262009, 2022. DOI: 10.1371/journal.pone.0262009.

- [12] T. J. Mbah, H. Ye, J. Zhang y M. Long, «Using LSTM and ARIMA to Simulate and Predict Limestone Price Variations,» *Mining, Metallurgy & Exploration*, vol. 38, n.º 2, págs. 913-926, 2021. DOI: 10.1007/s42461-020-00362-y.
- [13] C. Xia, «Comparative Analysis of ARIMA and LSTM Models for Agricultural Product Price Forecasting,» *Highlights in Science, Engineering and Technology*, vol. 85, págs. 1032-1040, 2024. DOI: 10.54097/8q6nx369.
- [14] S. Liu, «A study of short-term forecasting of raw material prices using a combined LSTM-ARIMA model,» en *Proceedings of the International Conference on Cloud Computing, Performance Computing and Deep Learning (CCPCDL 2023)*, vol. 12712, SPIE, 2023, pág. 1 271 213. DOI: 10.1117/12.2679048.
- [15] S. Ray, A. Lama, P. Mishra, T. Biswas, S. S. Das y B. Gurung, «An ARIMA-LSTM model for predicting volatile agricultural price series with random forest technique,» *Applied Soft Computing*, vol. 149, pág. 110 939, 2023. DOI: 10.1016/j.asoc.2023.110939.
- [16] Banco de la República de Colombia, *TRM diaria y series históricas de precios accionarios – 2014–2023*, Datos históricos 2014–2023. Accedido en 2025, 2023. dirección: <https://www.banrep.gov.co/es/estadisticas>.

Tesis_PrediccionPrecioSacos_AndresMorales.pdf

por ANDRES FELIPE MORALES MARTINEZ

Fecha de entrega: 27-nov-2025 09:03p. m. (UTC-0500)

Identificador de la entrega: 2829226091

Nombre del archivo: Tesis_PrediccionPrecioSacos_AndresMorales.pdf (5.48M)

Total de palabras: 11777

Total de caracteres: 67327



Predicción del precio óptimo de compra de sacos de papel en la industria cementera: Un enfoque basado en modelos SARIMAX, ARIMA y red neuronal LSTM.

ANDRÉS FELIPE MORALES MARTÍNEZ

Tesis

Asesor

Diego Fernando Fonseca Valero

UNIVERSIDAD EAFIT
ESCUELA DE CIENCIAS APLICADAS E INGENIERÍA
MAESTRÍA EN CIENCIA DE DATOS Y ANALÍTICA
MEDELLÍN
2025

CONTENIDO

0	INTRODUCCIÓN	6
1	PLANTEAMIENTO DEL PROBLEMA	7
1.1.	Antecedentes y contexto	7
1.2.	Formulación del problema	7
1.3.	Pregunta de investigación	8
1.4.	Alcance y viabilidad	8
2	Justificación	9
3	Objetivos	10
3.1.	Objetivo general	10
3.2.	Objetivos específicos	10
4	Marco teórico y Metodología	11
4.1.	Gestión de compras bajo incertidumbre de precios	11
4.2.	Modelos de series temporales para predicción de precios	12
4.2.1.	Modelos lineales: ARIMA y SARIMAX	12
4.2.2.	Modelos no lineales de aprendizaje profundo: redes LSTM	13
4.2.3.	Modelos de <i>gradient boosting</i> en series temporales: LightGBM	13
4.2.4.	Enfoques híbridos y metodologías de <i>ensemble</i>	14
4.3.	Evaluación rigurosa de modelos de pronóstico	15
4.3.1.	Métricas de error y capacidad explicativa	15
4.3.2.	Protocolos de validación y comparación entre modelos	15
4.4.	Brecha identificada y aportes esperados	16
4.5.	Síntesis comparativa de la literatura	16
4.6.	Metodología	17
4.7.	Enfoque general y diseño experimental	17
4.8.	Glosario de abreviaturas	18
4.9.	Datos y fuentes de información	19
4.9.1.	Datos internos: serie objetivo	19
4.9.2.	Datos externos: variables exógenas	20
4.10.	Preprocesamiento y control de fuga de información	21
4.10.1.	Reconstrucción de serie diaria	21
4.10.2.	Partición temporal y transformaciones	21
4.11.	Ingeniería de características	22
4.11.1.	Lags y suavizadores del objetivo	22

4.11.2. Retornos y volatilidad de variables exógenas	22
4.11.3. Variables de calendario	23
4.12. Modelos y especificaciones	23
4.12.1. Modelos ARIMA y SARIMAX	23
4.12.2. Modelo LSTM	23
4.12.3. Modelo LightGBM	23
4.13. Esquema de validación temporal	24
4.14. Métricas de evaluación	25
4.15. Reproducibilidad y trazabilidad	26
5 Resultados	27
5.1. Análisis exploratorio de datos	27
5.2. Fase I: modelos base	28
5.3. Fase II: LSTM con variables exógenas	29
5.4. Fase III: transformación de datos y Gradient Boosting	30
5.5. Fase IV: LSTM con serie diaria transformada	31
5.6. Comparación global	32
6 Conclusiones	34
6.1. Conclusiones generales	34
6.2. Contribuciones principales	35
6.3. Implicaciones para la gestión de compras	36
6.4. Limitaciones del estudio	37
6.5. Líneas de trabajo futuro	38
REFERENCIAS	39

LISTA DE TABLAS

Tabla 1	Resumen comparativo de familias de modelos consideradas en la tesis. . .	14
Tabla 2	Tabla resumen comparativa de modelos predictivos en la literatura. Resultados reportados en diferentes dominios de aplicación para modelos ARIMA/SARIMAX, LSTM y enfoques híbridos; las métricas específicas varían según el estudio original.	17
Tabla 3	Comparación de desempeño (modelos base y LSTM con exógenas, conjunto de prueba).	29
Tabla 4	Comparación de desempeño en conjunto de prueba (10 % más reciente).	33

LISTA DE FIGURAS

Figura 1	Esquema de ventaneo para el modelo LSTM: para cada ventana de entrada se genera un pronóstico un día adelante, respetando el orden temporal.	24
Figura 2	Esquema de partición temporal 90/10 y validación cruzada con ventanas crecientes.	25
Figura 3	Serie histórica del precio neto de compra de sacos de papel kraft (2014–2023).	28
Figura 4	Comportamiento del modelo ARIMA en el conjunto de prueba: valores observados vs. pronosticados.	29
Figura 5	Desempeño de LightGBM en el conjunto de prueba (10 % más reciente): valores observados vs. pronosticados.	31
Figura 6	Desempeño del LSTM con serie diaria transformada en el conjunto de prueba: valores observados vs. pronosticados.	32

Resumen

En esta tesis, existe un problema importante relacionado con la predicción del precio de compra de sacos de papel kraft en un entorno de alta incertidumbre y con ausencia de herramientas de cobertura financiera. Después de extraer conocimiento transaccional del ERP (2014–2023) y variables exógenas públicas (TRM y precios de acciones de proveedores globales), construimos un conjunto de datos diario, escalonado con el esquema LOCF, y creamos una canalización de características diseñada para evitar la fuga de información. Usando esto como plataforma, se eligieron cuatro modelos basados en series temporales: ARIMA, SARIMAX, red neuronal LSTM y LightGBM como método de aumento de gradiente en árboles de decisión, de acuerdo con un protocolo de validación temporal del 90/10 junto con la evaluación en un conjunto de datos reciente de retención. El resultado indica que los modelos lineales y el LSTM básico en la serie original tienen errores absolutos y relativos muy altos, por lo que no pueden apoyar decisiones de compra precisas. LightGBM aplicado a la serie diaria transformada, por otro lado, reduce los errores relativos a alrededor del 1 %, y el LSTM entrenado con esa misma representación ofrece el mejor rendimiento general, en términos de MAE aproximadamente dos veces más bajo que el de LightGBM en el último período. Estos resultados muestran que una buena representación de la serie, junto con un modelo no lineal y variables externas significativas, también aseguran que las previsiones puedan ser operativamente útiles, allanando el camino hacia una gestión de compras más basada en datos para este insumo clave.

Palabras clave: *Pronóstico de precios, series de tiempo, LSTM, LightGBM, gestión de compras, sacos de papel kraft.*

0. INTRODUCCIÓN

La bolsa de papel kraft de 50 kg es un factor importante en la estructura de costos del producto terminado en la industria del cemento. El precio de su compra en pesos colombianos es resultado de fuerzas externas, por ejemplo, los tipos de cambio y el estado del comercio internacional de celulosa y papel, sin un instrumento formal de cobertura (por ejemplo, mercado de futuros) para este insumo. Como consecuencia, las decisiones de compra se toman con gran incertidumbre respecto al movimiento futuro de precios, lo que reduce el poder de negociación y la eficiencia de la planificación. En los últimos años, los cambios recientes en los métodos de pronóstico de series temporales han permitido una evolución de modelos lineales más tradicionales, incluidos ARIMA y SARIMAX, a modelos de series temporales basados en aprendizaje profundo y algoritmos de aumento de gradiente que tienen en cuenta interrelaciones complejas entre múltiples variables y series temporales.

Sin embargo, el uso sistemático de estos modelos dentro de América Latina para el control de ciertas compras de insumos, particularmente cuando se combina datos transaccionales internos con variables exógenas replicables (y públicas), es limitado. Nuestro trabajo aplica el pronóstico del precio de compra de bolsas de papel kraft para empresas cementeras colombianas utilizando datos históricos del sistema ERP y series exógenas públicas (TRM y precios de acciones de proveedores globales relevantes) para el período 2014–2023. El objetivo principal es construir, validar y comparar modelos ARIMA, SARIMAX, LSTM y LightGBM utilizando un protocolo de validación temporal sólido y compararlos entre sí, para evaluar un conjunto de soluciones (procesamiento de nivel intermedio) capaces de generar señales de compra accionables que mitiguen dicha incertidumbre en el proceso de toma de decisiones de compra.

Este documento está organizado de la siguiente manera: La Sección 1 formula el problema de investigación, incluyendo antecedentes, formulación declarativa, pregunta central y alcance. La Sección 2 está dedicada a las justificaciones prácticas y académicas del estudio. El objetivo general y los objetivos específicos se presentan en la Sección 3. La Sección 4 integra el marco teórico y la metodología en un solo documento que explica tanto los fundamentos conceptuales como el diseño experimental, los datos, el preprocesamiento, los modelos, las métricas y los criterios de evaluación. Luego, la Sección 5 presenta y discute los resultados obtenidos en las diferentes fases de modelado. Finalmente, la Sección 6 resume las principales conclusiones, las contribuciones del estudio, sus limitaciones y las líneas de trabajo futuro.

1. PLANTEAMIENTO DEL PROBLEMA

1.1. Antecedentes y contexto

La empresa Cementos Argos ha empleado bolsas de papel kraft de 50 kg como el principal insumo para el embalaje de cemento. Tradicionalmente, el precio de compra se decidía después de un número limitado de discusiones con uno o algunos proveedores clave, junto con una investigación descriptiva de las tendencias recientes de precios y pronósticos cualitativos de la condición económica. La compañía tiene un historial de precios de compra transaccionales de 2014 a 2023, junto con información pública sobre el tipo de cambio y ⁵ los precios de las acciones de las empresas del sector papelerero, pero esta información no se ha incorporado en un sistema formal de pronóstico. En la práctica, las decisiones sobre el volumen de compra y el momento se toman en respuesta a la experiencia de los equipos de compras y a las señales parciales del mercado sin un marco cuantitativo para estimar escenarios de precios y evaluar alternativas. Este escenario también sugiere que, en el contexto de una devaluación de la moneda o choques en los mercados internacionales de papel, la empresa corre un riesgo considerable de pagar precios altos sin la información para prepararse para tal impacto y evitarlo.

1.2. Formulación del problema

En la situación actual, la empresa:

- Toma decisiones de compra de sacos de papel kraft sin contar con pronósticos cuantitativos sistemáticos del precio de compra.
- No dispone de un protocolo estandarizado para comparar el desempeño de diferentes modelos de pronóstico ni para traducir los resultados en reglas operativas de compra.
- Aprovecha de manera parcial la información disponible en su ERP y en fuentes públicas externas.

Como consecuencia, las decisiones de negociación y programación de compras se realizan con alta incertidumbre, lo que puede traducirse en mayores costos y en una menor capacidad de aprovechar ventanas de oportunidad cuando el precio del insumo presenta caídas temporales.

El problema que aborda este trabajo se puede formular de manera declarativa como sigue:

La empresa carece de un sistema cuantitativo de pronóstico de precios que integre datos internos y señales exógenas, y que permita evaluar de manera comparativa distintos modelos de series de tiempo, con el fin de apoyar las decisiones de compra de sacos de papel kraft y reducir la incertidumbre en la negociación.

1.3. Pregunta de investigación

A partir de la formulación anterior, la pregunta central de investigación es:

¿Qué modelo de pronóstico basado en series de tiempo (ARIMA, SARIMAX, LSTM o LightGBM), entrenado con datos transaccionales internos y series exógenas públicas, ofrece el mejor desempeño fuera de muestra para anticipar el precio de compra de sacos de papel kraft en una empresa cementera colombiana y cómo puede traducirse ese pronóstico en señales operativas de compra?

1.4. Alcance y viabilidad

El estudio se limita a:

- Una empresa cementera específica, con datos internos de precios de sacos de papel kraft correspondientes al periodo 2014–2023.
- Un conjunto de variables exógenas públicas seleccionadas por su relevancia económica y disponibilidad: tipo de cambio representativo del mercado (TRM) y precios accionarios de fabricantes globales de papel y empaques.
- El horizonte de pronóstico de corto plazo requerido para la planificación de compras y negociaciones periódicas, trabajando con un esquema de validación temporal de tipo 90/10 y evaluación en un bloque de prueba holdout.

El objetivo no es desarrollar un contrato de cobertura o una cartera de los mejores proveedores, sino evaluar el potencial de diferentes modelos de series temporales para mejorar la información disponible en el momento de la toma de decisiones. La viabilidad del estudio se facilita mediante datos históricos, modelado estadístico y herramientas de aprendizaje automático; además, la replicabilidad del protocolo metodológico en otros insumos críticos de la cadena de suministro de la empresa.

2. Justificación

El precio de las bolsas de papel kraft es, desde un punto de vista práctico, una fracción considerable del costo total del cemento empaquetado. Pequeñas diferencias en el precio de compra, acumuladas a lo largo del volumen anual, pueden tener efectos financieros considerables. Cuando existen pronósticos confiables del precio de este insumo, se permite identificar cuándo existen ventanas de oportunidad para negociar o preacordar compras, minimiza la exposición a períodos de altos precios y proporciona verificación numérica de las decisiones de suministro. Desde la perspectiva del análisis de datos, este problema requiere convertir el rastro transaccional del ERP y las series exógenas públicas en un sistema de pronóstico que sea auditable, reproducible y reutilizable dentro de la organización. Esto implica transformar fuentes históricas heterogéneas en un flujo estructurado que permita generar, validar y comparar modelos bajo criterios metodológicos consistentes, de manera que los pronósticos puedan incorporarse de forma confiable en los procesos de planeación y negociación de compras. Producir un sistema de pronóstico replicable, basado en evidencia y científicamente correcto ayuda a convertir este conocimiento en un activo analítico que complementa la estrategia de gestión de riesgos y la continuidad del negocio. En el ámbito académico, el presente trabajo responde a un vacío particular en la literatura aplicada de América Latina: llenar la escasez de estudios sobre las series de precios de insumos industriales, a saber, bolsas de papel kraft, basándose en datos transaccionales empresariales en combinación con variables exógenas públicas y evaluando formalmente modelos lineales y no lineales utilizando un método de validación temporal. A través del detalle del diseño y verificación del conjunto de datos y la configuración de los modelos ARIMA, SARIMAX, LSTM y LightGBM en condiciones realistas, la tesis ofrece la oportunidad de generar un estudio de investigación replicable en la comunidad de análisis de datos de la cadena de suministro y ciencia de la decisión. Finalmente, este trabajo se alinea con el objetivo de la organización de mejorar su capacidad analítica sofisticada en áreas centrales como la adquisición y gestión de proveedores, y juega un papel en proporcionar prueba interna de la importancia de combinar métodos de series temporales y aprendizaje automático en la toma de decisiones operativas.

3. Objetivos

3.1. Objetivo general

Desarrollar, validar y comparar modelos de pronóstico basados en series de tiempo (ARIMA, SARIMAX, LSTM y LightGBM), entrenados con datos transaccionales internos y variables exógenas públicas, para anticipar el precio de compra de sacos de papel kraft en una empresa cementera colombiana y evaluar su utilidad como soporte a las decisiones operativas de compra.

3.2. Objetivos específicos

1. Construir un conjunto de datos limpio y reproducible que integre el historial de precios de compra de sacos de papel kraft con series exógenas públicas relevantes, documentando criterios de depuración, imputación y transformación temporal.
2. Desarrollar y entrenar modelos ARIMA, SARIMAX, LSTM y LightGBM bajo un esquema de validación temporal, cuidando la ausencia de fuga de información y la comparabilidad entre enfoques.
3. Evaluar comparativamente el desempeño de los modelos mediante métricas de error fuera de muestra (por ejemplo, MAE, RMSE, sMAPE, WAPE y R^2), en relación con modelos de referencia ingenuos.
4. Analizar el potencial de los modelos con mejor desempeño para generar señales operativas de compra (por ejemplo, ventanas de oportunidad asociadas a caídas pronosticadas de precio) y discutir su impacto esperado en el costo del insumo.
5. Documentar el protocolo metodológico y los artefactos desarrollados (scripts, configuraciones, transformaciones) de manera que faciliten su replicación y adaptación a otros insumos y contextos.

4. Marco teórico y Metodología

La predicción de precios en cadenas de suministro constituye un desafío metodológico que combina la teoría económica con técnicas avanzadas de análisis de series de tiempo [1][2]. En el contexto de insumos industriales como los sacos de papel kraft, la empresa compradora enfrenta decisiones de abastecimiento bajo alta incertidumbre de precios, exposición cambiaria y restricciones contractuales [3]. Este marco teórico presenta los fundamentos conceptuales que sustentan la investigación, organizados en tres ejes principales: (i) la gestión de compras bajo incertidumbre de precios en cadenas de suministro, (ii) los enfoques metodológicos para la predicción de series temporales aplicados a precios y costos, y (iii) las métricas y protocolos de evaluación rigurosa de modelos de pronóstico. Finalmente, se sintetiza la brecha específica que aborda esta tesis.

4.1. Gestión de compras bajo incertidumbre de precios

La gestión de compras en cadenas de suministro busca asegurar el abastecimiento oportuno de insumos críticos al menor costo posible, garantizando al mismo tiempo la continuidad operativa y el cumplimiento de estándares de calidad [3]. En mercados donde existen instrumentos de cobertura bien desarrollados, las empresas pueden mitigar parte del riesgo de precio mediante estrategias financieras; sin embargo, en muchos insumos especializados, como los empaques industriales de papel kraft, no existen mercados de derivados líquidos, por lo que la exposición al riesgo de precio se gestiona principalmente mediante decisiones operativas y negociaciones con proveedores [4].

La incertidumbre se manifiesta en la variabilidad de costos de materias primas, la volatilidad del tipo de cambio y las condiciones de oferta de los proveedores. La literatura en compras y gestión de costos resalta la importancia de ir más allá del precio unitario y considerar el costo total de propiedad del insumo, es decir, todos los costos asociados a la adquisición, uso y disposición del bien [5]. Cuando no se dispone de herramientas cuantitativas de pronóstico, las decisiones de compra tienden a basarse en promedios históricos recientes o en la intuición de los responsables de compras, lo que puede conducir a compras en momentos desfavorables y a una utilización ineficiente del poder de negociación.

Diversos autores han enfatizado que la integración de información histórica y modelos de pronóstico en procesos de compras permite identificar ventanas de oportunidad, estimar rangos plausibles de evolución de precios y respaldar con evidencia cuantitativa la negociación

con proveedores [2, 6]. Para que estos modelos sean útiles, deben construirse sobre datos confiables, evaluarse rigurosamente y traducirse en indicadores comprensibles para los gestores de la cadena de suministro.

4.2. Modelos de series temporales para predicción de precios

La predicción de precios a partir de series temporales ha sido abordada tradicionalmente mediante modelos estadísticos lineales, a los que se han sumado en las últimas décadas enfoques no lineales de aprendizaje profundo y métodos de *gradient boosting* [6]. Cada familia de modelos presenta supuestos, fortalezas y limitaciones que resultan relevantes al analizar series de precios con comportamiento escalonado, presencia de *shocks* y posible influencia de variables exógenas.

4.2.1. Modelos lineales: ARIMA y SARIMAX Los modelos autorregresivos integrados de media móvil (ARIMA) constituyen uno de los enfoques clásicos para el pronóstico de series de tiempo. Estos modelos suponen que el valor futuro de la serie puede representarse como una combinación lineal de sus propios rezagos y de términos de error pasados, después de aplicar, si es necesario, transformaciones de diferenciación para lograr estacionariedad [1, 2]. La familia ARIMA ofrece un marco bien establecido, con criterios de selección de órdenes, diagnóstico de residuales y herramientas de inferencia. Siguiendo la formulación clásica de los modelos ARIMA propuesta por Box et al. [1], se asume que el valor futuro de la serie puede escribirse como una combinación lineal de rezagos de la propia serie y de errores pasados, después de aplicar las transformaciones necesarias para lograr estacionariedad.

Cuando se dispone de variables exógenas que pueden contener información relevante sobre la evolución futura de la serie objetivo, los modelos se extienden a la clase SARIMAX, que incorpora regresores externos y, cuando corresponde, componentes estacionales [2]. En el contexto de precios de insumos industriales, variables como el tipo de cambio o indicadores de mercado de proveedores pueden tratarse como regresores que desplazan el nivel esperado de la serie, manteniendo al mismo tiempo la estructura autorregresiva.

La principal ventaja de ARIMA y SARIMAX radica en su interpretabilidad y en la amplia experiencia acumulada en su uso. Sin embargo, su capacidad para capturar dinámicas fuertemente no lineales, cambios de régimen y comportamientos escalonados puede ser limitada, especialmente cuando la relación entre la serie objetivo y las exógenas es compleja o cuando los ajustes de precio se producen en forma de saltos discretos en lugar de variaciones

graduales [6].

4.2.2. Modelos no lineales de aprendizaje profundo: redes LSTM Las redes neuronales recurrentes (RNN) y, en particular, las arquitecturas de memoria de largo corto plazo (LSTM, por sus siglas en inglés), han sido ampliamente adoptadas para modelar series de tiempo con dependencias temporales complejas. Las LSTM introducen celdas de memoria y mecanismos de compuertas que permiten retener información relevante durante horizontes largos y mitigar problemas de desvanecimiento del gradiente [7].

En el ámbito de la predicción de precios, las LSTM se han utilizado para capturar relaciones no lineales entre la serie objetivo y un conjunto de variables de entrada que incluyen rezagos del propio precio, indicadores técnicos, variables macroeconómicas o de mercado, y otros factores exógenos [6]. A diferencia de los modelos lineales, las LSTM no imponen una estructura funcional predeterminada, sino que aprenden patrones a partir de los datos mediante entrenamiento supervisado.

Esta flexibilidad viene acompañada de desafíos: la necesidad de contar con conjuntos de datos suficientemente ricos, la sensibilidad a la selección de hiperparámetros (tamaño de la red, longitud de ventana, tasas de aprendizaje, regularización) y el riesgo de sobreajuste si no se emplean esquemas de validación adecuados [2]. Además, la interpretabilidad suele ser menor que en modelos lineales, lo que exige recursos adicionales para explicar el comportamiento del modelo a tomadores de decisión no expertos en aprendizaje profundo.

4.2.3. Modelos de *gradient boosting* en series temporales: LightGBM Los métodos de *gradient boosting* basados en árboles de decisión, como LightGBM, han demostrado un alto desempeño en tareas de regresión y clasificación con datos tabulares [8][9]. Estos modelos construyen un ensamble de árboles poco profundos que se ajustan de manera secuencial para corregir los errores de las iteraciones previas, optimizando una función de pérdida específica.

Aplicados a series temporales, estos métodos requieren que la información temporal se represente explícitamente mediante características (por ejemplo, rezagos de la serie objetivo, retornos y volatilidades de variables exógenas, variables de calendario). Una vez construida esta representación, el modelo de *boosting* puede capturar interacciones no lineales y efectos umbral sin necesidad de especificar una forma funcional explícita [8].

Entre las ventajas de enfoques como LightGBM se encuentran su capacidad para manejar relaciones complejas, su eficiencia computacional en conjuntos de datos de tamaño moderado

y la posibilidad de obtener medidas de importancia de variables que apoyan la interpretación [9]. No obstante, al no modelar la dependencia temporal de manera explícita como un proceso estocástico, la calidad del pronóstico depende en gran medida de la ingeniería de características y del cuidado en el diseño del esquema de validación temporal.

Modelo	Supuestos principales	Fortalezas	Limitaciones
ARIMA/SARIMAX	Linealidad, estructura ARMA, estacionariedad (tras transformación)	Interpretables, marco bien establecido	Dificultad para capturar no linealidades fuertes y cambios de régimen
LSTM	Relación no lineal entre entradas y objetivo, dependencias temporales complejas	Capacidad para capturar dependencias de largo plazo	Requiere más datos, calibración compleja y menor interpretabilidad
LightGBM	Relación no lineal, datos tabulares con características explícitas	Maneja interacciones complejas, eficiente en conjuntos de datos tabulares	Depende fuertemente de la ingeniería de características y del esquema de validación

Tabla 1. Resumen comparativo de familias de modelos consideradas en la tesis.

La Tabla [1] resume de forma sintética los supuestos, fortalezas y limitaciones de las familias de modelos consideradas en este estudio.

4.2.4. Enfoques híbridos y metodologías de *ensemble* La literatura reciente reporta enfoques híbridos que combinan modelos lineales (por ejemplo, ARIMA) con modelos no lineales (como redes LSTM), con la intención de aprovechar la capacidad de los primeros para capturar estructura lineal y la de los segundos para modelar patrones residuales no lineales [6]. Una estrategia común consiste en ajustar un modelo ARIMA a la serie original, modelar los residuales mediante una red LSTM y combinar las predicciones de ambos componentes.

Estos enfoques híbridos pueden ofrecer mejoras de desempeño en ciertos contextos, pero también incrementan la complejidad del sistema, tanto en términos de calibración como de mantenimiento. Además, requieren un cuidado especial en el diseño del esquema de validación para evitar sobreestimar su capacidad predictiva. En el contexto de esta tesis, los modelos híbridos se consideran como parte del panorama conceptual de la literatura, pero el análisis empírico se centra en la comparación de ARIMA, SARIMAX, LSTM y LightGBM, que son los modelos incluidos formalmente en los objetivos del estudio.

4.3. Evaluación rigurosa de modelos de pronóstico

El valor práctico de un modelo de pronóstico no depende únicamente de su sofisticación técnica, sino de la solidez con la que se evalúa su desempeño y de la forma en que sus resultados se traducen en decisiones de negocio. La literatura enfatiza dos componentes centrales en esta evaluación: (i) la selección e interpretación de métricas de error y (ii) el uso de protocolos temporales consistentes para validar y comparar modelos fuera de muestra [2], [6].

4.3.1. Métricas de error y capacidad explicativa Las métricas de error cuantifican la discrepancia entre los valores observados y los pronosticados. Entre las más utilizadas en problemas de precios se encuentran el error absoluto medio (MAE), el error cuadrático medio (RMSE), el porcentaje absoluto medio simétrico (sMAPE) y el porcentaje absoluto medio ponderado (WAPE) [2]. El MAE y el RMSE expresan el error en unidades de la variable objetivo, siendo el RMSE más sensible a errores grandes; las métricas porcentuales, por su parte, facilitan la interpretación relativa del error y permiten comparaciones entre periodos o productos con magnitudes distintas.

Como complemento, en algunos casos se reporta el coeficiente de determinación (R^2) como indicador global de capacidad explicativa. Sin embargo, su interpretación en series temporales debe hacerse con cautela, especialmente cuando existen saltos discretos o cambios de régimen [6]. Por ello, se recomienda analizar varias métricas de forma conjunta y considerar no solo el error promedio, sino también su estabilidad temporal y relevancia operativa.

4.3.2. Protocolos de validación y comparación entre modelos En series de tiempo la dependencia temporal impide el uso de esquemas de validación que mezclen aleatoriamente observaciones de distintos momentos [2]. En consecuencia, se emplean protocolos específicos que respetan el orden cronológico, tales como particiones entrenamiento-prueba, validación cruzada con ventanas crecientes (expanding window) y esquemas walk-forward sobre bloques recientes. El objetivo es aproximar el escenario real de uso en producción: entrenar con datos pasados y evaluar sobre periodos futuros no vistos, reduciendo el riesgo de sobreajuste, en particular en modelos altamente flexibles como LSTM o métodos de gradient boosting [6].

La comparación entre modelos puede abordarse de forma descriptiva contrastando métricas fuera de muestra bajo el mismo protocolo temporal, o mediante pruebas estadísticas de precisión predictiva. Una referencia clásica es la prueba de Diebold–Mariano, que evalúa si las diferencias en error entre dos modelos son estadísticamente significativas a partir de sus series

de pérdidas [10]. Independientemente de la prueba utilizada, la literatura converge en tres buenas prácticas: (i) definir benchmarks ingenuos, (ii) aplicar exactamente el mismo esquema temporal a todos los modelos y (iii) reportar de forma transparente métricas y decisiones derivadas de ellas, garantizando comparaciones justas y auditables.

En este estudio, basado en estas directrices, se realiza un análisis de medidas fuera de muestra según un protocolo temporal consistente, pero no se utilizan pruebas estadísticas formales como la prueba de Diebold-Mariano. La prueba se menciona solo como una referencia metodológica en la literatura, y se propone explícitamente como una línea de investigación futura, ya que la implementación de este método implicaría un análisis más extenso de las series de pérdidas y un marco comparativo más amplio.

4.4. Brecha identificada y aportes esperados

La revisión de la literatura muestra una amplia aplicación de modelos de series de tiempo a precios de materias primas, tipos de cambio y activos financieros, así como un creciente uso de modelos no lineales y métodos de *ensemble* en problemas de pronóstico [6]. Sin embargo, se observa una menor presencia de estudios que aborden insumos industriales específicos, como los sacos de papel kraft utilizados en la industria cementera, combinando datos transaccionales extraídos de sistemas ERP con variables exógenas públicas y comparando de forma sistemática modelos lineales y no lineales bajo protocolos de validación temporal realistas.

Esta brecha es particularmente notoria en el contexto latinoamericano, donde las empresas enfrentan alta volatilidad cambiaria y limitaciones de cobertura financiera, pero disponen cada vez más de datos históricos detallados [4]. La tesis se posiciona en este espacio, al integrar conceptos de gestión de compras bajo incertidumbre, técnicas avanzadas de series de tiempo y buenas prácticas de evaluación de modelos, con el objetivo de aportar evidencia aplicada sobre el uso de ARIMA, SARIMAX, LSTM y LightGBM para el pronóstico de precios de un insumo crítico en una cadena de suministro real.

4.5. Síntesis comparativa de la literatura

La Tabla 2 presenta una síntesis de estudios representativos que comparan modelos ARIMA/SARIMAX, LSTM e híbridos en diversos dominios. Los resultados son mixtos: ARIMA supera a LSTM en series con estructura predominantemente lineal (epidemiología, precios de materias primas estables), mientras que LSTM y modelos híbridos son superiores en series

con alta no linealidad y volatilidad (series agrícolas, materias primas con choques).

Estudio	Dominio	ARIMA/SARIMAX	LSTM	Híbrido	Mejor
Zhang (2022) 11	Epidemiología	MAE 12.3	MAE 15.7	–	ARIMA
Mbah (2021) 12	Piedra caliza	Acc 95.7 %	Acc 91.8 %	–	ARIMA
Xia (2024) 13	Agrícola	RMSE 45.2	RMSE 13.5	–	LSTM
Liu (2023) 14	Materias primas	0.082	0.074	0.065	Híbrido
Ray et al. (2023) 15	Agrícola	R^2 0.82	R^2 0.89	0.936	Híbrido

Tabla 2. **Tabla resumen comparativa de modelos predictivos en la literatura.**

Resultados reportados en diferentes dominios de aplicación para modelos ARIMA/SARIMAX, LSTM y enfoques híbridos; las métricas específicas varían según el estudio original.

4.6. Metodología

Esta sección describe el diseño metodológico seguido para construir el conjunto de datos, especificar los modelos de pronóstico y evaluar su desempeño en un contexto de decisión de compras. Se integra en un solo bloque todo lo relacionado con datos, preprocesamiento, modelos, validación, métricas y reproducibilidad, evitando duplicaciones y manteniendo coherencia con los objetivos planteados y con el marco teórico [2](#) [6](#).

4.7. Enfoque general y diseño experimental

El estudio adopta un enfoque cuantitativo, observacional y retrospectivo, basado en datos históricos transaccionales y series exógenas públicas. El objetivo es comparar el desempeño fuera de muestra de cuatro familias de modelos de series de tiempo —ARIMA, SARIMAX, redes LSTM y LightGBM— para el pronóstico del precio de compra de sacos de papel kraft, bajo un protocolo de validación temporal riguroso [1](#) [7](#) [9](#).

Para articular la narrativa de resultados, el diseño experimental se organiza en cuatro fases, que corresponden a escenarios progresivos de complejidad metodológica:

- **Fase I: Modelos base sobre la serie original.** Evaluación inicial de ARIMA, SARIMAX y un LSTM base utilizando la serie de precios en su forma original, con irregularidad en el calendario y duplicados fecha–producto.

- **Fase II: LSTM con variables exógenas.** Extensión del LSTM base incorporando como entradas la TRM y precios accionarios de proveedores globales, manteniendo aún la estructura original de la serie.
- **Fase III: Transformación de datos y Gradient Boosting.** Reconstrucción de una serie diaria equiespaciada mediante LOCF, ingeniería de características sobre el objetivo y las exógenas, e implementación de LightGBM como modelo de *gradient boosting* [8] [9].
- **Fase IV: LSTM sobre la serie diaria transformada.** Entrenamiento de un LSTM utilizando la misma representación diaria enriquecida, para contrastar un modelo de aprendizaje profundo secuencial con el enfoque tabular de boosting.

Todas las fases comparten el mismo horizonte de pronóstico (un día adelante, $h = 1$) y el mismo esquema de partición temporal (90 % entrenamiento, 10 % prueba), de modo que las comparaciones entre modelos sean consistentes y libres de fuga de información [2].

4.8. Glosario de abreviaturas

Para facilitar la lectura y evitar ambigüedades, en esta subsección se presentan las principales siglas utilizadas a lo largo del documento:

- **ERP:** *Enterprise Resource Planning*. Sistema de planificación de recursos empresariales (por ejemplo, SAP MM).
- **TRM:** *Tasa Representativa del Mercado*. Tasa de cambio oficial COP/USD publicada por el Banco de la República.
- **LSTM:** *Long Short-Term Memory*. Arquitectura de redes neuronales recurrentes diseñada para modelar dependencias de largo plazo en series de tiempo.
- **ARIMA:** *Autoregressive Integrated Moving Average*. Modelo clásico de series de tiempo basado en componentes autorregresivos y de promedios móviles.
- **SARIMAX:** *Seasonal ARIMA with eXogenous regressors*. Extensión de ARIMA que incorpora estacionalidad y variables exógenas.
- **GBM / Gradient Boosting:** Método de ensamble basado en la construcción secuencial de árboles para minimizar el error de predicción.

- **LightGBM**: Implementación eficiente del método de *gradient boosting* basada en histogramas y optimizada para conjuntos de datos tabulares.
- **MAE**: *Mean Absolute Error*. Error absoluto medio.
- **RMSE**: *Root Mean Squared Error*. Raíz del error cuadrático medio.
- **sMAPE**: *Symmetric Mean Absolute Percentage Error*. Porcentaje absoluto medio simétrico.
- **WAPE**: *Weighted Absolute Percentage Error*. Error porcentual absoluto ponderado.
- **IQR**: *Interquartile Range*. Rango intercuartílico (Q3-Q1).
- **LOCF**: *Last Observation Carried Forward*. Técnica que rellena valores faltantes con la última observación disponible.
- **DM**: *Diebold-Mariano*. Prueba estadística para comparar precisión predictiva entre dos modelos.

4.9. Datos y fuentes de información

4.9.1. Datos internos: serie objetivo La variable objetivo y_t corresponde al precio neto de compra en COP de sacos de papel kraft de 50 kg, extraído del sistema ERP corporativo para el periodo 2014–2023. A partir del módulo de compras se consolidaron todas las órdenes asociadas al ítem de interés, identificando para cada día con transacción el precio unitario efectivo después de descuentos y ajustes.

La serie presenta un comportamiento característico de escalones: el precio se mantiene constante entre fechas de compra y se ajusta únicamente cuando se renegocian condiciones, lo que refleja la lógica contractual del insumo. Este patrón, habitual en insumos industriales negociados por lotes, condiciona la elección de transformaciones y modelos apropiados [2][6].

En la depuración se eliminaron registros duplicados fecha-producto, correspondientes a múltiples líneas con el mismo ítem y día, que distorsionaban la señal temporal. Aproximadamente la mitad de los registros originales fueron identificados como duplicados y consolidados en un único precio representativo por fecha, manteniendo la trazabilidad de la fuente ERP.

Cuantitativamente, el conjunto de datos original contenía alrededor de 8,705 registros de compras de bolsas de papel kraft desde 2014 hasta 2023, obtenidos del módulo de compras del

sistema ERP (como el módulo SAP MM). Después de eliminar duplicados de fecha-producto, filtrar inconsistencias y descartar registros con información incompleta, se obtuvo un conjunto refinado de 3,567 observaciones válidas. Los datos diarios utilizados en diferentes partes del análisis se basan en esta serie agregada.

4.9.2. Datos externos: variables exógenas Como señales relacionadas con el frente de oferta se incorporaron series exógenas públicas:

- **TRM COP/USD diaria.** La tasa de cambio afecta directamente el costo de insumos importados o indexados a mercados internacionales, por lo que es un determinante clave del nivel de precios en COP.
- **Precios accionarios de proveedores globales de empaques/papel.** Se utilizaron los precios diarios en USD de compañías representativas del sector, como Klabin, Mondi y Smurfit Kappa, como *proxies* de la dinámica de oferta y de los choques en la cadena de valor del papel kraft.

La selección de estos factores se debe al hecho de que insumos como el papel kraft forman un costo de producción, hasta cierto punto, que está determinado por elementos globales, como los precios de la pulpa, la energía y la logística, y por la situación financiera de los principales actores en la cadena de suministro [4]. Específicamente, el tipo de cambio COP/USD captura el elemento de intercambio que acentúa o mitiga estos choques (en moneda local) y sus diferencias se transmiten, aunque con cierto retraso, al precio en pesos de los sacos de papel kraft. De manera similar, los precios de las acciones de las empresas Klabin, Mondi y Smurfit indican las expectativas agregadas del mercado con respecto a los márgenes de producción, la capacidad instalada, los precios de las materias primas y los posibles cuellos de botella logísticos, traducidos así como *proxies* para la presión de costos de suministro y aguas arriba, y por lo tanto pueden usarse para predecir ajustes en las condiciones de la negociación sobre el insumo estudiado.

Las series de TRM diaria y los precios accionarios se obtuvieron del Banco de la República de Colombia, utilizando los registros oficiales publicados en su módulo estadístico para el periodo 2014–2023 [16]. Todas las series exógenas se alinearon con el periodo de análisis 2014–2023, de forma que cada observación diaria del precio del saco pudiera asociarse de manera unívoca con el valor contemporáneo de las variables externas.

4.10. Preprocesamiento y control de fuga de información

El preprocesamiento tuvo dos objetivos principales: (i) construir una representación temporal coherente con la lógica de negocio (serie diaria por escalones) y (ii) garantizar que ninguna transformación introdujera información del futuro en el entrenamiento de los modelos.

4.10.1. Reconstrucción de serie diaria A partir de las fechas de compra se generó una serie diaria equiespaciada para todo el periodo de estudio. En días sin transacción, el precio se imputó mediante el esquema *last observation carried forward* (LOCF), manteniendo constante el último precio negociado. Esta estrategia respeta la morfología escalonada del proceso de fijación de precios: la ausencia de compras no implica un cambio en el precio contractual, sino la continuidad de las condiciones vigentes [6].

4.10.2. Partición temporal y transformaciones Antes de aplicar cualquier transformación que pudiera aprovechar regularidades temporales, la serie se dividió en dos bloques en orden cronológico:

- 90 % de las observaciones iniciales para calibración y validación interna.
- 10 % más reciente como conjunto de prueba (*holdout*) exclusivo para evaluación final.

Todas las transformaciones que dependen del tiempo (rezagos, promedios móviles, retornos y volatilidades) se calcularon utilizando estrictamente información disponible hasta $t - 1$. En particular:

- Los rezagos del objetivo y de las exógenas se definieron como $y_{t-\ell}$ y $x_{t-\ell}$, con ℓ en un conjunto finito de retardos económicamente plausibles.
- Los promedios móviles históricos $y_{t,7}$ y $y_{t,14}$ se calcularon sin incluir la observación de t , de forma que resumen el nivel reciente sin romper la estructura por tramos.
- Los retornos logarítmicos y medidas de volatilidad de las exógenas se construyeron a partir de ventanas puramente históricas.

Los escaladores utilizados para normalizar variables (en los modelos que lo requieren, como LSTM) se ajustaron únicamente con el subconjunto de entrenamiento y se aplicaron

después al conjunto de prueba, evitando cualquier fuga de información [2]. En conjunto, este *pipeline* garantiza que cada etapa del procesamiento respete el orden temporal y que los modelos se evalúen en condiciones realistas de uso.

4.11. Ingeniería de características

La ingeniería de características se diseñó para capturar tanto la estructura de escalones del precio como las señales de choque y tendencia provenientes de las exógenas, manteniendo siempre la coherencia temporal [6].

4.11.1. Lags y suavizadores del objetivo Para la serie objetivo y_t se incluyeron:

- Rezagos $y_{t-\ell}$ con $\ell \in \{1, 5, 7, 14, 22, 30\}$, que aportan contexto de corto y mediano plazo.
- Promedios móviles históricos $y_{t,7}$ y $y_{t,14}$, calculados sin usar la observación de t , que resumen el nivel reciente sin romper la estructura por tramos.

No se aplicaron retornos al objetivo debido a su naturaleza escalonada: diferencias o retornos generarían largas rachas de ceros y picos esporádicos poco informativos para los modelos basados en árboles.

4.11.2. Retornos y volatilidad de variables exógenas Las exógenas se representaron mediante:

- Retornos logarítmicos a distintos horizontes $r_{j,t,h} = \log\left(\frac{x_{j,t}}{x_{j,t-h}}\right)$ para $h \in \{1, 5\}$, donde j indexa cada serie (TRM, Klabi, Mondi, Smurfit). Estos retornos son insensibles a la escala y capturan choques relativos recientes.
- Volatilidad reciente, definida como la desviación estándar de los retornos diarios en una ventana de 10 días, $\sigma_{j,t,10}$, que resume el nivel de incertidumbre en cada exógena.

La intuición económica es que aumentos persistentes o alta volatilidad en TRM o en los *proxies* de oferta elevan la probabilidad de renegociación y ajustes de precio en los sacos de papel.

4.11.3. Variables de calendario Finalmente, se incluyeron variables de calendario (mes, trimestre y dummies de año) para capturar potenciales efectos de estacionalidad contractual o patrones asociados a revisiones periódicas de precios, siguiendo prácticas habituales en pronóstico de negocios [2].

4.12. Modelos y especificaciones

Se implementaron cuatro familias de modelos bajo un protocolo común.

4.12.1. Modelos ARIMA y SARIMAX Los modelos $ARIMA(p, d, q)$ y $SARIMAX(p, d, q)(P, D, Q)_s$ se ajustaron sobre la serie original irregular en las Fases I–II y sobre la serie diaria transformada en los experimentos posteriores, con y sin variables exógenas según correspondía, utilizando criterios de información (AIC/BIC) y diagnóstico de residuales para seleccionar órdenes apropiados [1, 2]. En SARIMAX, las exógenas se incluyeron de forma contemporánea y con rezagos económicamente plausibles, evitando configuraciones que implicaran usar información del futuro.

4.12.2. Modelo LSTM Las redes LSTM se entrenaron sobre ventanas de entrada de longitud entre 30 y 90 días, que incluyen el historial reciente del precio y las exógenas [7]. La arquitectura empleó:

- Una o dos capas LSTM con un número moderado de unidades.
- Regularización mediante *dropout* y *early stopping* sobre la pérdida de validación.
- Normalización previa de las entradas con estadísticas calculadas solo en el conjunto de entrenamiento.

La Figura 1 ilustra el esquema de ventaneo: para cada ventana de entrada se genera un pronóstico un día adelante, respetando el orden temporal.

4.12.3. Modelo LightGBM El modelo de *gradient boosting* se implementó con LightGBM, utilizando la representación tabular construida a partir de rezagos, retornos, volatilidades y variables de calendario [8, 9]. Se exploraron hiperparámetros como profundidad máxima de los árboles, tasa de aprendizaje, número de estimadores y mínimos de datos por hoja,

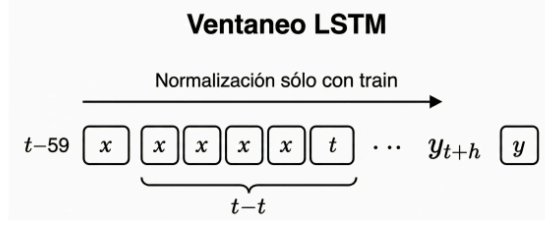


Fig. 1: Esquema de ventaneo para el modelo LSTM: para cada ventana de entrada se genera un pronóstico un día adelante, respetando el orden temporal.

mediante búsqueda con validación cruzada temporal (*TimeSeriesSplit*) sobre el bloque de calibración.

LightGBM no requiere escalado de variables y es capaz de capturar relaciones no lineales y efectos umbral, lo que resulta adecuado para series con saltos discretos y señales exógenas complejas.

4.13. Esquema de validación temporal

El esquema de validación se diseñó para aproximar el uso real de los modelos en producción y asegurar comparaciones justas entre enfoques [2][6].

- **Corte 90/10.** Se reservó el 10 % más reciente de la serie como conjunto *holdout* para evaluación final fuera de muestra.
- **Validación cruzada creciente en el 90 %.** En el bloque de calibración se aplicó una validación cruzada con ventanas crecientes (folds acumulativos), donde cada fold añade un tramo adicional de datos en el tiempo.
- **Esquema *walk-forward* en el *holdout*.** En el 10 % final se utilizó un esquema *walk-forward*: el modelo se recalibra secuencialmente a medida que se avanza en el tiempo, generando pronósticos un día adelante con la información disponible hasta cada punto.

La Figura 2 resume gráficamente la partición temporal y el flujo de entrenamiento–validación–prueba. Este esquema se aplicó de forma consistente a todos los modelos, de modo que las diferencias de desempeño puedan atribuirse al modelo y no al procedimiento

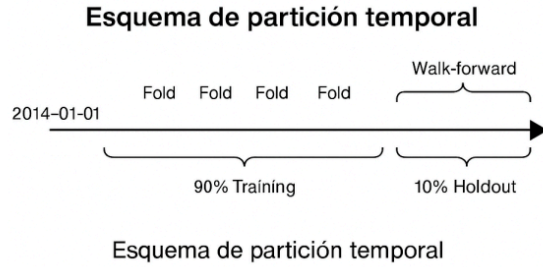


Fig. 2: Esquema de partición temporal 90/10 y validación cruzada con ventanas crecientes.

de evaluación. En esta versión del trabajo no se aplicaron pruebas formales de comparación como la de Diebold–Mariano; la comparación entre modelos se realiza mediante la magnitud de las métricas de error en el conjunto *holdout* y el análisis gráfico de residuales y pronósticos, siguiendo recomendaciones generales de la literatura [6].

4.14. Métricas de evaluación

La evaluación de desempeño se realizó con un conjunto de métricas complementarias, recomendadas en la literatura de pronóstico de series de tiempo [2, 6]:

- **Error absoluto medio (MAE).** Mide el error promedio en las mismas unidades del precio (COP), interpretado como desviación típica esperada.
- **Error cuadrático medio (RMSE).** Penaliza más los errores grandes y es útil para detectar modelos que ocasionalmente cometen errores extremos.
- **Porcentaje absoluto medio simétrico (sMAPE).** Facilita la comparación relativa de errores en distintos niveles de precio.
- **Porcentaje absoluto medio ponderado (WAPE).** Resume la proporción del error absoluto respecto a la suma de los valores observados, útil para comunicar el impacto agregado sobre el costo total de compras.
- **Coefficiente de determinación (R^2).** Se reporta únicamente para algunos modelos como indicador de capacidad explicativa global, interpretándolo con cautela en el contexto de series de tiempo.

Adicionalmente, se comparó el desempeño de cada modelo frente a líneas base ingenuas (persistencia y promedios móviles simples), con el objetivo de verificar que los modelos propuestos aportan valor predictivo más allá de reglas triviales [6].

Establecer estas referencias de comparación en el análisis se complementó con la referencia a las buenas prácticas de reporte recomendadas por la literatura para (i) asegurar la transparencia en la definición de indicadores, (ii) garantizar que todos los modelos fueran evaluados bajo el mismo esquema de validación temporal, y (iii) apoyar visualmente el resultado con gráficos de valores observados vs. predichos y un análisis residual básico. La sección de Resultados implementa estas prácticas para mostrar para cada modelo tanto los valores numéricos de las métricas como las figuras que ilustran su comportamiento en el conjunto de prueba.

4.15. Reproducibilidad y trazabilidad

Para garantizar la reproducibilidad del estudio se adoptaron las siguientes prácticas, en línea con recomendaciones recientes en ciencia de datos aplicada al pronóstico [6]:

- Versionamiento del código y de los scripts de preprocesamiento, entrenamiento y evaluación.
- Fijación de semillas aleatorias en los componentes estocásticos (inicialización de redes neuronales, particiones internas de validación).
- Registro sistemático de hiperparámetros, métricas y artefactos generados (modelos entrenados, escaladores, conjuntos de características).
- Documentación de las versiones de librerías utilizadas (*p. ej.*, `pandas`, `statsmodels`, `scikit-learn`, `lightgbm`, `tensorflow/keras`).

Estas medidas permiten auditar el proceso de punta a punta y facilitan la replicación del análisis por parte de la organización o de evaluadores externos.

5. Resultados

De acuerdo con el diseño experimental proporcionado en la Sección 4 esta sección presenta los resultados organizados en cuatro fases sucesivas que siguen exactamente el flujo metodológico planteado. En primer lugar, se reportan el diagnóstico exploratorio de la serie y el desempeño de los modelos base sobre la serie original (Fase I y Fase II). Posteriormente, se analiza el impacto de reconstruir una serie diaria equiespaciada y de aplicar ingeniería de características para entrenar el modelo de Gradient Boosting (Fase III). Finalmente, se presenta el desempeño del modelo LSTM entrenado sobre esta representación diaria transformada (Fase IV) y se realiza una comparación global entre todos los enfoques evaluados.

5.1. Análisis exploratorio de datos

Una serie de precios netos de compra exhibe un comportamiento claramente escalonado: el precio solo cambia en los días de transacción y permanece constante entre compras debido a la lógica contractual de la entrada. Desde el período 2014-2023, cuando el nivel de precios aumentó para los períodos observados en la Figura 3 hay (i) períodos de picos aislados al principio, (ii) períodos más largos de poca variación, y (iii) una tendencia reciente al alza. Esta heterogeneidad apunta a una dinámica de saltos discretos y períodos de relativa estabilidad.

El precio diario durante el período 2014-2023 osciló aproximadamente entre 457 y 1,353 COP en términos descriptivos, con una mediana cercana a 652 COP y un rango intercuartil de 139 COP. De la serie reconstruida, se detectaron aproximadamente 69 cambios de nivel, con un intervalo mediano de 16 días entre renegociaciones, lo que respalda la estabilidad a largo plazo observada con esta entrada. También hay momentos durante los escalones en los que se limitan a niveles relativamente bajos (especialmente, entre 2014 y 2016), y más recientemente, una fase donde el nivel de precios comienza a subir gradualmente a niveles más altos (específicamente, entre 2021 y 2023). Como se ve en la Figura 3 estas tendencias acentúan la importancia de crear modelos que consideren fluctuaciones discretas y que traten el período más reciente como el más apropiado para elegir comprar algo.

Además, para probar diferentes señales externas al cambio de nivel, se investiga el gráfico de comovimiento entre el precio de los sacos y la serie exógena, así como correlaciones simples. Especialmente, se encontraron correlaciones positivas moderadas entre el nivel de precios y la TRM, y asociaciones positivas regulares entre el precio y los rendimientos acumulados de algunas acciones de proveedores en varios momentos recientes, lo que apoya la interpreta-

ción de estas como señales del lado de la oferta. Estos patrones fomentan la implementación sistemática de estas variables exógenas en las Fases II-IV del estudio. Operativamente, los largos escalones muestran que un precio negociado tiende a mantenerse durante unos días o semanas hasta la próxima compra, mientras que los saltos indican renegociaciones o choques externos relevantes (como cambios en el tipo de cambio o costo internacional). Este diagnóstico exploratorio, combinado con la evidencia previamente descrita y exógena, llevó al rediseño de la representación temporal (serie diaria con LOCF) y también aumentó la presencia de señales exógenas en otras etapas del análisis.



Fig. 3: Serie histórica del precio neto de compra de sacos de papel kraft (2014–2023).

5.2. Fase I: modelos base

En la Fase I se evaluaron tres modelos base sobre la serie original de precios: ARIMA, SARIMAX y un LSTM “simple” sin variables exógenas. La comparación se realizó en el 10 % más reciente de la serie (conjunto de prueba), bajo el protocolo de partición temporal descrito en la Metodología. La Tabla 3 resume los resultados cuantitativos.

ARIMA alcanzó un MAE de 96.70 y un sMAPE de 10.03 %, mientras que SARIMAX obtuvo un MAE de 132.66 y un sMAPE de 13.40 %. Estos valores evidencian errores absolutos y relativos altos para efectos operativos, con predicciones que no capturan adecuadamente los cambios de nivel. La Figura 4 ilustra el comportamiento de ARIMA: el modelo tiende a suavizar la serie, no alinea bien los escalones y subestima sistemáticamente ciertos tramos de precio, confirmando la limitación de enfoques lineales para este tipo de serie escalonada.

El LSTM base, entrenado sobre ventanas de la serie objetivo sin exógenas, mejora parcialmente el desempeño respecto a los modelos lineales: MAE de 81.87 y sMAPE de 9.59 %.

Sin embargo, la magnitud del error absoluto sigue siendo elevada frente a los niveles de precio relevantes para negociación, por lo que el modelo no alcanza aún la precisión operativa deseada. Esta fase sirve como línea base para evaluar el aporte de variables exógenas y transformaciones más ricas en las fases siguientes.

Tabla 3. Comparación de desempeño (modelos base y LSTM con exógenas, conjunto de prueba).

Modelo	MAE	RMSE	sMAPE
ARIMA	96.70	118.16	10.03 %
SARIMAX	132.66	161.35	13.40 %
LSTM (base)	81.87	83.49	9.59 %
LSTM (TRM + Smurfit)	123.99	168.71	16.68 %



Fig. 4: Comportamiento del modelo ARIMA en el conjunto de prueba: valores observados vs. pronosticados.

5.3. Fase II: LSTM con variables exógenas

En la Fase II se incorporaron variables exógenas asociadas al frente de oferta y al entorno macro: la TRM COP/USD y los precios accionarios de referentes/proveedores globales (Klabin, Mondi y Smurfit). Desde la lógica de negocio, la TRM captura la presión cambiaria sobre insumos importados, mientras que las acciones de proveedores reflejan condiciones de mercado y costos en la cadena de papel y empaques. En particular, Smurfit tiene un rol central como proveedor clave, por lo que se exploraron configuraciones que priorizan esta señal.

Se entrenaron varias arquitecturas LSTM con combinaciones de exógenas, manteniendo el mismo protocolo de validación temporal de la Fase I. La mejor variante en términos de MAE fue la que combinó la TRM y el precio de la acción de Smurfit como regresores adicionales. En el conjunto de prueba (10 % más reciente), esta especificación LSTM (TRM + Smurfit) obtuvo un MAE de 123.99, un RMSE de 168.71 y un sMAPE de 16.68 %, tal como se resume en la Tabla 3.

Estos resultados son peores que los del LSTM base e incluso que los de ARIMA en términos de MAE y sMAPE. Es decir, en esta configuración particular la inclusión directa de exógenas no mejoró el pronóstico y, de hecho, degradó el desempeño. Una posible explicación es que, al trabajar sobre la serie original poco transformada, la red tiende a “memorizar” patrones ruidosos en lugar de extraer señales robustas de las variables externas. Esta fase muestra que la introducción de exógenas en LSTM no garantiza mejoras automáticas y que se requieren transformaciones más cuidadosas para aprovecharlas.

5.4. Fase III: transformación de datos y Gradient Boosting

La Fase III transforma el problema en un problema de regresión tabular de la serie diaria transformada junto con el conjunto de características especificadas en la Sección 4. En lugar de tratar con la serie irregular original, el modelo LightGBM recibe como entrada una representación diaria que incluye rezagos del precio objetivo, rendimientos y medidas de volatilidad de las variables exógenas, además de variables de calendario. El objetivo de esta fase es evaluar hasta qué punto este cambio de representación y el uso de un método de *gradient boosting* permiten reducir de manera significativa los errores obtenidos en las fases anteriores.

En el conjunto de prueba (10 % más reciente), LightGBM alcanza un MAE de 8.03 COP, con métricas relativas sMAPE = 0.92 % y WAPE = 0.90 %. Esto representa una reducción sustancial del error frente a los modelos evaluados en las fases anteriores, cuyos MAE se ubicaban en el orden de 80–130 COP (ARIMA, SARIMAX y LSTM sobre la serie original). En la Figura 5 se observa que el modelo replica con precisión los tramos constantes de la serie y que los errores más visibles se concentran alrededor de algunos cambios bruscos de nivel, donde pequeñas desviaciones en la fecha del salto generan discrepancias puntuales en el holdout. Aun así, la magnitud agregada del error se mantiene en un rango que hace que las predicciones sean operativamente útiles para apoyar decisiones de compra, especialmente si se interpretan como una señal adicional sobre el nivel esperado del precio y no como un valor exacto.

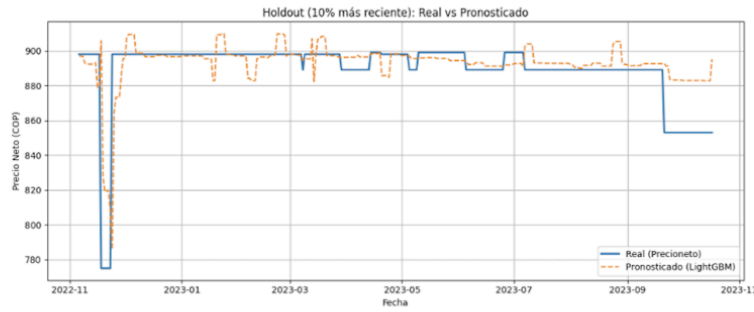


Fig. 5: Desempeño de LightGBM en el conjunto de prueba (10 % más reciente): valores observados vs. pronosticados.

5.5. Fase IV: LSTM con serie diaria transformada

La Fase IV retoma la arquitectura LSTM, pero ahora sobre la serie diaria transformada con LOCF, es decir, sobre una representación que respeta explícitamente los escalones y elimina duplicados de la serie original. El objetivo es evaluar si, al partir de una estructura temporal más limpia y estable, la red recurrente puede capturar dependencias de largo plazo y mejorar aún más la precisión obtenida con Gradient Boosting.

Bajo el mismo esquema de partición 90/10 y validación temporal, el LSTM con serie diaria transformada alcanzó en el conjunto de prueba un MAE de 4.57, un RMSE de 12.40 y un R^2 de 0.96. Estos valores implican una reducción adicional de la mitad del MAE respecto a LightGBM y un ajuste muy alto en términos de varianza explicada. La Figura 6 ilustra el comportamiento del modelo: se observan trayectorias pronosticadas que siguen de forma precisa los tramos constantes y anticipan caídas relevantes de precio con desplazamientos temporales acotados.

En síntesis, la Fase IV confirma que la combinación de (i) una representación temporal coherente con la lógica contractual (LOCF) y (ii) un modelo secuencial capaz de explotar dependencias de mayor longitud permite capturar mejor la dinámica del precio del insumo. No obstante, el costo computacional y la sensibilidad a hiperparámetros son mayores que en el caso de LightGBM, lo que introduce un compromiso entre precisión y simplicidad operativa.

Es importante notar que las Figuras 5 y 6 no cubren exactamente el mismo tramo tem-

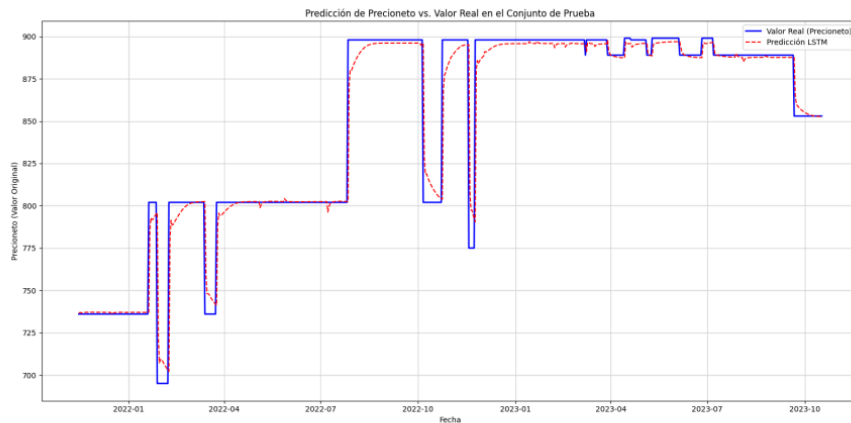


Fig. 6: Desempeño del LSTM con serie diaria transformada en el conjunto de prueba: valores observados vs. pronosticados.

poral. En todos los casos, las métricas de desempeño se calculan sobre el mismo conjunto de prueba (el último 10 % de la serie), pero para LightGBM se grafica la totalidad del periodo de *holdout*, mientras que en el caso del LSTM se presenta un acercamiento al subtramo más reciente del conjunto de prueba, donde la serie exhibe una mayor concentración de escalones y donde las decisiones de compra son operativamente más relevantes. Esta decisión de visualización busca equilibrar legibilidad y foco analítico: el rendimiento global queda cuantificado por las métricas sobre todo el conjunto de prueba, mientras que las figuras ilustran con mayor claridad el comportamiento típico de los modelos en segmentos representativos de dicho periodo.

5.6. Comparación global

La Tabla 4 resume de forma consolidada el desempeño de los principales modelos evaluados en las distintas fases, usando el MAE como métrica comparativa homogénea en el conjunto de prueba (10 % más reciente). Se incluyen los modelos lineales (ARIMA y SARIMAX), el LSTM base, el LSTM con exógenas (TRM + Smurfit), el modelo de Gradient Boosting (LightGBM) y el LSTM con serie diaria transformada.

En términos de MAE, el modelo con mejor desempeño global es el LSTM con serie diaria

transformada ($MAE = 4.57$), seguido por LightGBM ($MAE = 8.03$). Los modelos lineales (ARIMA y SARIMAX) y el LSTM base quedan claramente rezagados, con errores absolutos un orden de magnitud mayores. La variante LSTM con exógenas sobre la serie original presenta el peor desempeño, lo que refuerza la conclusión de que la forma de representar la serie y de transformar las variables externas es tan importante como la elección de la familia de modelos.

Desde la perspectiva de uso en la organización, la tesis considera al LSTM con serie diaria transformada como el modelo de referencia en términos de precisión de pronóstico, y a LightGBM como una alternativa robusta y más simple de implementar y mantener en producción. En consecuencia, las recomendaciones operativas de compra se pueden apoyar en el LSTM como escenario principal y en LightGBM como modelo de respaldo y validación cruzada de señales.

Tabla 4. Comparación de desempeño en conjunto de prueba (10 % más reciente).

Modelo	MAE (COP)
ARIMA	96.70
SARIMAX	132.66
LSTM (base)	81.87
LSTM (TRM + Smurfit)	123.99
LightGBM	8.03
LSTM (serie diaria)	4.57

6. Conclusiones

Este estudio propuso resolver un problema de predicción del costo de la adquisición de sacos de papel kraft en una situación de alta incertidumbre y falta de instrumentos de cobertura financiera. Utilizando datos transaccionales internos (2014-2023) y variables exógenas públicas (TRM y precios de acciones de proveedores globales), se desarrolló una canalización de datos reproducible y se compararon los modelos ARIMA, SARIMAX, LSTM y LightGBM bajo el mismo protocolo temporal. A continuación, se presentan los principales hallazgos del análisis con importantes contribuciones, implicaciones para la gestión de compras, así como las principales limitaciones y direcciones para trabajos futuros.

6.1. Conclusiones generales

La tesis primero demuestra la validez de que la representación de la serie de precios no es menos importante que la elección del modelo. La limpieza de los registros duplicados de fecha-producto y la reconstrucción de la serie diaria con la técnica de Última Observación Llevada Adelante (LOCF) fueron esenciales para expresar la naturaleza escalonada del precio: largos tramos invariables con saltos discretos correspondientes a renegociaciones. Para la serie irregular original, tanto el modelo lineal como el LSTM base tienen altos errores absolutos y dificultades para encontrar los cambios de nivel con precisión; para la serie diaria transformada, los modelos no lineales pueden lograr ganancias sustanciales en la precisión.

En segundo lugar, la comparación metódica de los modelos indica que ARIMA y SARIMAX (modelos lineales) no son apropiados para series de esta naturaleza. A pesar de calibraciones diligentes, sus errores relativos en el conjunto de prueba permanecen en rangos que no ayudan a facilitar decisiones de compra precisas. El simple LSTM estándar mejora estos hallazgos hasta cierto punto, pero aún no logra la precisión esperada.

En tercer lugar, dos métodos no lineales son evidentes dentro de la serie diaria transformada y señales exógenas: (i) un modelo de Gradient Boosting con LightGBM presentado con rezagos de precio, retornos y volatilidad de las variables exógenas y del calendario, que reduce los errores relativos a menos del 1 %; (ii) un LSTM entrenado en la misma serie diaria enriquecida con características resulta en una mejora de los datos de salida que provoca una reducción adicional de aproximadamente la mitad del error absoluto en comparación con LightGBM y un ajuste extremadamente bueno en términos de varianza explicada para el último conjunto.

Finalmente, dado que este documento considera la precisión de las previsiones como un propósito importante, tomamos el LSTM con series diarias transformadas como un modelo de referencia del rendimiento predictivo. LightGBM sigue siendo una alternativa extremadamente poderosa que es más fácil de implementar y mantener en producción, al tiempo que proporciona un punto de contraste transparente. El enfoque práctico recomendado es adoptar el LSTM diario como fuente de previsión y LightGBM como modelo de respaldo y validación cruzada de las señales, compartiendo el mismo flujo de datos y el mismo esquema de validación temporal.

6.2. Contribuciones principales

Desde la perspectiva metodológica y aplicada, el trabajo deja tres contribuciones centrales:

- **Pipeline de datos anti-fuga para series escalonadas de compras.** Se construyó un flujo de datos reproducible, comenzando desde el ERP, sin duplicados, una serie diaria que se alinea con la lógica contractual y definiendo características (retardos, promedios móviles, retornos, volatilidad, calendario) únicamente con información disponible hasta $t - 1$. Este pipeline puede repetirse para otras entradas y también prepara bien el escenario para trabajos posteriores dentro de la organización.
- **Evidencia comparativa de modelos en un caso real de insumo industrial.** La tesis proporciona una comparación sistemática de ARIMA, SARIMAX, LSTM y LightGBM en el contexto de un problema real de compras industriales, demostrando por qué los modelos lineales son menos ventajosos que los métodos no lineales cuando una serie tiene estructura de pasos y se incorporan variables exógenas relevantes. El resultado más relevante no es simplemente identificar un “modelo ganador”, sino más bien demostrar que LOCF, señales de cambio y modelos no lineales juntos reducen el error a grados donde el pronóstico es accionable.
- **Traducción del pronóstico a un lenguaje útil para compras.** Más allá de los indicadores estadísticos, el trabajo describe cómo interpretar métricas como MAE, WAPE y sMAPE en términos de su impacto en el costo total de compra, y ver los pronósticos diarios como indicadores que pueden ayudar al planificar pedidos y negociar con proveedores. Esto sitúa los resultados en el contexto del lenguaje empresarial y facilita la apropiación de la herramienta por parte del departamento de compras.

6.3. Implicaciones para la gestión de compras

Los resultados producidos son de importancia inmediata para la gestión del riesgo de precios en el campo de las compras. En ausencia de instrumentos de cobertura financiera para este insumo, un pronóstico diario con un error relativo bajo puede ser tratado como otra señal cuantitativa que puede emplearse para informar las etapas de planificación y negociación.

Las aplicaciones gerenciales a un primer nivel abarcan la conversión de métricas, como MAE o WAPE, en magnitudes monetarias agregadas, con referencia al volumen total de compra del período. Permite evaluar si el error típico del modelo está dentro de rangos aceptables desde un punto de vista financiero, y cuantificar aproximadamente el costo esperado asociado con decisiones basadas en pronósticos imperfectos. Además, evaluar la estabilidad del error a lo largo del tiempo es crítico para determinar si el ruido producido por las señales del modelo es confiable: un error bajo y altamente inestable impedirá el uso operativo del modelo; un error pequeño y estable permitirá tomar decisiones recurrentes.

Bajo estas consideraciones, las previsiones obtenidas pueden apoyar decisiones operativas tales como:

- Identificar momentos en los que el modelo anticipa estabilidad o disminuciones del precio y, por tanto, informar la conveniencia de posponer compras o fortalecer la negociación para capturar ventanas temporales favorables.
- Detectar tendencias al alza sostenidas que sugieran adelantar pedidos, renegociar condiciones o revisar cláusulas contractuales para mitigar impactos de precio.
- Complementar la experiencia del comprador y las señales cualitativas del mercado con una estimación cuantitativa que integra el efecto de factores externos, como la TRM y el comportamiento de proveedores globales.

Un segundo nivel —propuesto explícitamente como trabajo futuro— sería formalizar reglas de decisión basadas en el pronóstico (por ejemplo, políticas como “comprar/esperar/renegociar”) y evaluar su impacto a través de simulaciones históricas. Este enfoque permitiría pasar de la señal predictiva a un sistema de apoyo a la decisión más automatizado orientado hacia resultados económicos.

En general, la tesis muestra que la función de compras puede avanzar hacia una capacidad más basada en datos, donde las decisiones ya no se basan únicamente en promedios históricos recientes o intuición, sino que incorporan sistemáticamente pronósticos validados bajo

rigurosos protocolos temporales. La contribución radica no solo en el modelo ganador, sino en demostrar la viabilidad de integrar datos internos y externos en un marco cuantitativo que se alinea con las necesidades reales de negociación y planificación.

6.4. Limitaciones del estudio

Para situar adecuadamente el alcance de las conclusiones, es necesario reconocer varias limitaciones:

- En primer lugar, el análisis se basa en una única entrada (sacos de papel kraft de 50 kg) y un horizonte de vista a corto plazo (un día por delante). Si bien su metodología puede aplicarse conceptualmente a otros materiales y horizontes temporales, esta generalización no se verifica empíricamente en la tesis.
- En segundo lugar, los modelos fueron evaluados con métricas de error estimadas en una muestra reciente, así como utilizando un protocolo temporal consistente, aunque no se realizaron pruebas estadísticas formales para comparar la precisión entre los modelos. Las afirmaciones de superioridad relativa están respaldadas por la escala y consistencia de estas métricas, no por pruebas de hipótesis.
- En tercer lugar, el estudio se basó en datos de una empresa específica y en el contexto de una cadena de suministro particular y una política de compras. La extrapolación a otras organizaciones o industrias debe hacerse con la debida precaución y comparación con nuevas investigaciones empíricas cuando sea posible.
- Finalmente, aunque se discuten las posibles aplicaciones operativas de las previsiones (por ejemplo, apoyo a la decisión sobre mover o posponer la adquisición), esta tesis no proporcionó una estructura de decisión integrada, un esquema de prueba de políticas retrospectivas o un sistema de alerta automatizado formalizado para su uso en los procesos de la empresa. Las limitaciones no son contraproducentes para los resultados identificados, sino que más bien limitan el alcance de las afirmaciones y destacan áreas identificables para el desarrollo.

Estas limitaciones no invalidan los resultados, pero sí acotan el alcance de las afirmaciones y señalan espacios claros para continuar el desarrollo.

6.5. Líneas de trabajo futuro

Finalmente, la tesis abre varias líneas de trabajo que pueden potenciar el valor generado:

- **Extensión a otros insumos y horizontes.** Utiliza una lógica de modelo y una canalización de datos similar en otros materiales críticos de la cadena de suministro y prueba largos períodos de pronóstico (por ejemplo, semanas o meses) y compara el equilibrio entre anticipación y precisión.
- **Profundización en la comparación formal de modelos.** En trabajos futuros, en particular, proporciona cálculos estadísticos apropiados para la comparabilidad de errores en modelos y esquemas de backtesting con ventanas móviles para estudiar la estabilidad del rendimiento entre diferentes subperíodos.
- **Diseño e implementación de reglas de decisión y sistema de alertas.** Una línea de trabajo futuro consistiría en implementar estos pronósticos en políticas de gasto reales (reglas del tipo “comprar / esperar / renegociar”) y evaluar sus efectos mediante simulaciones históricas. Sobre esa base, sería posible diseñar e implementar un sistema de alertas que pueda integrarse en las herramientas de trabajo diarias del área de compras.
- **Exploración de variantes de modelos.** Explora otros modelos (por ejemplo, variantes híbridas o métodos de boosting con pérdidas cuantílicas) que permitan no solo obtener un pronóstico puntual, sino también bandas de incertidumbre útiles para utilizar en la planificación de escenarios.

Colectivamente, estas líneas pretenden convertir los resultados de la tesis en una capacidad operacionalizada que esté integrada: a partir del ejercicio de comparación de modelos de un estudio de caso único, para convertirse en una parte arraigada del proceso de planificación de compras y gestión del riesgo de precios en la empresa.

REFERENCIAS

- [1] G. E. P. Box, G. M. Jenkins, G. C. Reinsel y G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 5.^a ed. Hoboken, NJ: Wiley, 2015, ISBN: 9781118675021.
- [2] R. J. Hyndman y G. Athanasopoulos, *Forecasting: Principles and Practice*, 2.^a ed. Melbourne: OTexts, 2018. dirección: <https://otexts.com/fpp2/>
- [3] A. J. Van Weele, *Purchasing and Supply Chain Management: Analysis, Strategy, Planning and Practice*, 5.^a ed. Andover: Cengage Learning, 2010, ISBN: 9781408018965.
- [4] M. Christopher y B. Gaudenzi, «Managing Risks in Sustainable Supply Chains,» *Sinergie Italian Journal of Management*, vol. 33, n.º 96, págs. 65-82, 2015. DOI: [10.7433/s96.2015.04](https://doi.org/10.7433/s96.2015.04)
- [5] L. M. Ellram, «Total Cost of Ownership: An Analysis Approach for Purchasing,» *International Journal of Physical Distribution & Logistics Management*, vol. 25, n.º 8, págs. 4-23, 1995. DOI: [10.1108/09600039510099928](https://doi.org/10.1108/09600039510099928)
- [6] S. Makridakis, E. Spiliotis y V. Assimakopoulos, «Statistical and Machine Learning Forecasting Methods: Concerns and Ways Forward,» *PLOS ONE*, vol. 13, n.º 3, e0194889, 2018. DOI: [10.1371/journal.pone.0194889](https://doi.org/10.1371/journal.pone.0194889)
- [7] S. Hochreiter y J. Schmidhuber, «Long Short-Term Memory,» *Neural Computation*, vol. 9, n.º 8, págs. 1735-1780, 1997. DOI: [10.1162/neco.1997.9.8.1735](https://doi.org/10.1162/neco.1997.9.8.1735)
- [8] J. H. Friedman, «Greedy Function Approximation: A Gradient Boosting Machine,» *Annals of Statistics*, vol. 29, n.º 5, págs. 1189-1232, 2001. DOI: [10.1214/aos/1013203451](https://doi.org/10.1214/aos/1013203451)
- [9] G. Ke et al., «LightGBM: A Highly Efficient Gradient Boosting Decision Tree,» en *Advances in Neural Information Processing Systems*, vol. 30, 2017, págs. 3146-3154. dirección: <https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree>
- [10] F. X. Diebold y R. S. Mariano, «Comparing Predictive Accuracy,» *Journal of Business & Economic Statistics*, vol. 13, n.º 3, págs. 253-263, 1995. DOI: [10.1080/07350015.1995.10524599](https://doi.org/10.1080/07350015.1995.10524599)
- [11] R. Zhang, H. Song, Q. Chen, Y. Wang, S. Wang e Y. Li, «Comparison of ARIMA and LSTM for prediction of hemorrhagic fever at different time scales in China,» *PLOS ONE*, vol. 17, n.º 1, e0262009, 2022. DOI: [10.1371/journal.pone.0262009](https://doi.org/10.1371/journal.pone.0262009)

- [12] T. J. Mbah, H. Ye, J. Zhang y M. Long, «Using LSTM and ARIMA to Simulate and Predict Limestone Price Variations,» *Mining, Metallurgy & Exploration*, vol. 38, n.º 2, págs. 913-926, 2021. DOI: [10.1007/s42461-020-00362-y](https://doi.org/10.1007/s42461-020-00362-y),
- [13] C. Xia, «Comparative Analysis of ARIMA and LSTM Models for Agricultural Product Price Forecasting,» *Highlights in Science, Engineering and Technology*, vol. 85, págs. 1032-1040, 2024. DOI: [10.54097/8q6nx369](https://doi.org/10.54097/8q6nx369)
- [14] S. Liu, «A study of short-term forecasting of raw material prices using a combined LSTM-ARIMA model,» en *Proceedings of the International Conference on Cloud Computing, Performance Computing and Deep Learning (CCPCDL 2023)*, vol. 12712, SPIE, 2023, pág. 1 271 213. DOI: [10.1117/12.2679048](https://doi.org/10.1117/12.2679048)
- [15] S. Ray, A. Lama, P. Mishra, T. Biswas, S. S. Das y B. Gurung, «An ARIMA-LSTM model for predicting volatile agricultural price series with random forest technique,» *Applied Soft Computing*, vol. 149, pág. 110 939, 2023. DOI: [10.1016/j.asoc.2023.110939](https://doi.org/10.1016/j.asoc.2023.110939).
- [16] Banco de la República de Colombia, *TRM diaria y series históricas de precios accionarios – 2014–2023*, Datos históricos 2014–2023. Accedido en 2025, 2023. dirección: <https://www.banrep.gov.co/es/estadisticas>

Tesis_PrediccionPrecioSacos_AndresMorales.pdf

INFORME DE ORIGINALIDAD

1 %

INDICE DE SIMILITUD

1 %

FUENTES DE INTERNET

0 %

PUBLICACIONES

1 %

TRABAJOS DEL
ESTUDIANTE

FUENTES PRIMARIAS

1

[idoc.pub](#)

Fuente de Internet

<1 %

2

[www.diplomarbeiten24.de](#)

Fuente de Internet

<1 %

3

[upcommons.upc.edu](#)

Fuente de Internet

<1 %

4

[Submitted to usach](#)

Trabajo del estudiante

<1 %

5

[zagan.unizar.es](#)

Fuente de Internet

<1 %

Excluir citas

Activo

Excluir coincidencias

< 20 words

Excluir bibliografía

Activo


```
# -*- coding: utf-8 -*-
"""NoteBookCompartido.ipynb
```

Automatically generated by Colab.

Original file is located at
<https://colab.research.google.com/drive/14RPghU61T0l4y3nWVvYCI1bWmBOZlmGH>

```
# EXPLORACIÓ" DE DATOS RAW DIRECTAMENTE CRUDOS DE SAP
"""
```

```
#@title 1) InstalaciÓn de librerÍ-as (una vez por sesiÓn) para lectura de Excel, anÁlisis y grÁficos. No incluimos nada de modelado aÓen.
```

```
!pip -q install --upgrade pandas numpy matplotlib plotly openpyxl pyarrow
```

```
#@title Seguimos con las bibliotecas necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
from pathlib import Path
import random # Import the random module
import warnings # Import the warnings module
import os # Import the os module
```

```
warnings.filterwarnings("ignore")
SEED = 42
random.seed(SEED); np.random.seed(SEED)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
# === ParÁmetros del proyecto ===
PATH = "/content/drive/MyDrive/Base_Bolsas_Cementos .xlsx" # <- tu ruta exacta (incluye el espacio antes de .xlsx)
OUTDIR = "/content/drive/MyDrive/smurfit_outputs/eda_raw"
os.makedirs(OUTDIR, exist_ok=True)
```

```
assert os.path.exists(PATH), f"No se encontrÓ el archivo: {PATH}"
print("Usando archivo:", PATH)
print("Artefactos se guardarÓn en:", OUTDIR)
```

```
#@title 3) Utilidades de estandarizaciÓn y ayuda
```

```
#Definimos funciones para:
# naked_case: nombres de columnas limpios y consistentes standardize_columns:
#alias comunes del proyecto (proveedor, texto_breve, fecha, cantidad_pedido, precio_net) iqr_bounds: detecciÓn informativa de outliers por IQR (no elimina)
```

```
import re # Import re module
```

```
def snake_case(s: str) -> str:
    base = str(s).strip().lower()
    tr = {
        "Í": "a", "Í": "e", "Í-": "i", "Íi": "o", "Íe": "u", "Í±": "n",
        "/": " ", "\\": " ", " ": " ", "-": " ",
        ",": " ", ";": " ", "_": " ", "?": " ", "Bí": " ", "BÍ": " ", "!": " ",
        "(": " ", ")": " ", "[": " ", "]": " ", "{": " ", "}": " "
    }
    for k,v in tr.items(): base = base.replace(k,v)
    base = re.sub(r"_"+"", " ", base)
    base = re.sub(r"^[a-z0-9_]+","", base)
    return base.strip("_")
```

```
def standardize_columns(df: pd.DataFrame) -> pd.DataFrame:
    mapping = {c: snake_case(c) for c in df.columns}
    df = df.rename(columns=mapping)
    # Aliases especÍ-ficos del caso
    alias = {
        "proveedor/centro_suministrador": "proveedor",
        "textobreve": "texto_breve",
        "fechadocumento": "fecha",
        "cantidaddepedido": "cantidad_pedido",
        "precioneto": "precio_net",
    }
    for k,v in alias.items():
        if k in df.columns and v not in df.columns:
            df = df.rename(columns={k:v})
    return df
```

```
def iqr_bounds(s: pd.Series, k=1.5):
    s = s.dropna()
    if s.empty: return np.nan, np.nan
    q1, q3 = s.quantile(0.25), s.quantile(0.75)
    iqr = q3 - q1
    # Use np.logical_xor instead of subtraction with boolean values
    return q1 - k * iqr, q3 + k * iqr
```

```
#@title 4) Carga del RAW sin modificaciones
```

```
#Leemos el archivo tal cual. Esta vista es la referencia original para toda decisiÓn a futuro.
```

```
raw = pd.read_excel(PATH, engine="openpyxl")
print("Dimensiones RAW:", raw.shape)
raw.head(5)
```

```
#@title 5) EstandarizaciÓn de nombres y tipificado preliminar
```

```
#Normalizamos encabezados a snake_case y generamos un diccionario preliminar: tipo inferido, nulos, cardinalidad y ejemplo.
#Se guarda en 01_diccionario_preliminar.csv.
```

```
df = standardize_columns(raw.copy())
print("Nombres estandarizados:\n", list(df.columns))
```

```
desc = pd.DataFrame({
    "columna": df.columns,
    "dtype_inferido": [str(df[c].dtype) for c in df.columns],
    "n_nulos": [df[c].isna().sum() for c in df.columns],
    "n_unicos": [df[c].nunique(dropna=True) for c in df.columns],
    "ejemplo": [df[c].dropna().iloc[0] if df[c].dropna().shape[0]>0 else None for c in df.columns],
})
desc.to_csv(f"{OUTDIR}/01_diccionario_preliminar.csv", index=False)
desc
```

```
#@title 6) Calidad de datos: nulos y duplicados (etiquetado, NO eliminaciÓn)
```

```
#Medimos nulos por columna (conteo y %) y etiquetamos duplicados a nivel fila completa en es_duplicado.
#Guardamos el resultado etiquetado en df_etiquetado_duplicados.parquet.
```

```
nulos = df.isna().sum().sort_values(ascending=False).to_frame("n_nulos")
nulos["pct"] = (nulos["n_nulos"]/len(df))*100
display(nulos.head(20))
nulos.to_csv(f"{OUTDIR}/02_nulos_por_columna.csv")
```

```
dup_count = df.duplicated().sum()
```

```
print("Duplicados (filas completas):", dup_count)

df["es_duplicado"] = df.duplicated()
df.to_parquet(f"{OUTDIR}/df_etiquetado_duplicados.parquet", index=False)

#@title 7) Perfilado: variables numéricas vs categóricas

#Identificamos columnas numéricas y categóricas para orientar análisis y codificaciones futuras.
#Guardamos resumen numérico en 03_resumen_numericas.csv.

num_cols = [c for c in df.columns if pd.api.types.is_numeric_dtype(df[c])]
cat_cols = [c for c in df.columns if c not in num_cols and c != "es_duplicado"]

print("Numéricas:", num_cols)
print("Categóricas:", cat_cols)

prof_num = df[num_cols].describe().T if num_cols else pd.DataFrame()
if not prof_num.empty:
    prof_num.to_csv(f"{OUTDIR}/03_resumen_numericas.csv")
prof_num

#@title 8) Medimos cardinalidad de cada categoría y el Top-1 más frecuente.
#Esto ayuda a: (a) detectar variables con demasiadas categorías, (b) decidir codificaciones, (c) entender concentración por proveedor/material, etc.
#Guardamos 04_cardinalidad_categoricas.csv.

rows = []
for c in cat_cols:
    vc = df[c].value_counts(dropna=False)
    top1_valor = vc.index[0] if len(vc)>0 else None
    top1_freq = int(vc.iloc[0]) if len(vc)>0 else 0
    rows.append({
        "columna": c,
        "n_unicos": df[c].nunique(dropna=True),
        "top1_valor": top1_valor,
        "top1_freq": top1_freq,
        "top1_pct": (top1_freq/len(df))*100 if len(df)>0 else 0.0
    })

card = pd.DataFrame(rows).sort_values("n_unicos", ascending=False)
card.to_csv(f"{OUTDIR}/04_cardinalidad_categoricas.csv", index=False)
card

#@title 9) Fechas y cobertura temporal

#Detectamos candidatas de fecha por nombre, intentamos parsear y reportamos cobertura mensual y rango.
#Guardamos 05_cobertura_mensual.csv si se identifica fecha.

# Candidatas de fecha por nombre
date_like = [c for c in df.columns if "fecha" in c or "date" in c]
print("Candidatas de fecha:", date_like)

fecha_col = None
for c in ["fecha", "fechadocumento", "fecha_documento", "date"]:
    if c in df.columns:
        fecha_col = c; break
if fecha_col is None and len(date_like)>0:
    fecha_col = date_like[0]

if fecha_col is not None:
    df[fecha_col] = pd.to_datetime(df[fecha_col], errors="coerce")
    print("Rango temporal:", df[fecha_col].min(), "a", df[fecha_col].max())

    tmp = df[[fecha_col]].copy()
    tmp["yyyymm"] = tmp[fecha_col].dt.to_period("M").astype(str)
    cov = tmp["yyyymm"].value_counts().sort_index().to_frame("conteo")
    cov.to_csv(f"{OUTDIR}/05_cobertura_mensual.csv")
    display(cov.tail(12))
else:
    print("No se identificó automáticamente la columna de fecha. La definiremos luego si aplica.")

#@title 10) === Solo detectar outliers en Precioneto (IQR) ===
PATH = "/content/drive/MyDrive/Base_Bolsas_Cementos.xlsx" # incluye el espacio antes de .xlsx
OUTDIR = "/content/drive/MyDrive/smurfite_outputs/outliers"
K_IQR = 1.5 # factor IQR (típico 1.5). Si quieres ser más estricto/relajado, ajusta aquí.

import os, re, numpy as np, pandas as pd
from google.colab import drive

# 1) Asegurar acceso a Drive
if not os.path.exists("/content/drive"):
    drive.mount("/content/drive")
else:
    try:
        # si ya está montado, esto no hará nada
        drive.mount("/content/drive", force_remount=False)
    except Exception:
        pass

os.makedirs(OUTDIR, exist_ok=True)
assert os.path.exists(PATH), f"No se encontró el archivo: {PATH}"

# 2) Cargar RAW
df_raw = pd.read_excel(PATH, engine="openpyxl")

# 3) Normalizar encabezados a snake_case y ubicar 'Precioneto'
def snake_case(s: str) -> str:
    base = s.strip().lower()
    tr = {"ñ": "a", "é": "e", "í": "i", "ó": "o", "ú": "u", "±": "n",
           "/", "\\": " ", " ": " ", "_": " ",
           ":", ";", ",": " ", "?": " ", "B": " ", "V": " ", "!": " ",
           "(": " ", ")": " ", "[": " ", "]": " ", "{": " ", "}": " "}
    for k,v in tr.items(): base = base.replace(k,v)
    base = re.sub(r"__+", " ", base)
    base = re.sub(r"^[a-z0-9_]+", "", base)
    return base.strip("_")

df = df_raw.rename(columns={c: snake_case(c) for c in df_raw.columns})

# Renombrar explícitamente Precioneto -> precio_netto
if "precio_netto" not in df.columns:
    for cand in ["precioneto", "precio_netto ", "precio_netto__"]:
        if cand in df.columns:
            df = df.rename(columns={cand: "precio_netto"})
            break

assert "precio_netto" in df.columns, "No se encontró la columna objetivo (Precioneto / precio_netto)."
```

```
# 4) Parseo robusto de 'precio_netto' para evitar que quede en miles/billones por formato
def parse_precio_netto_auto(s: pd.Series):
    s_str = s.astype(str).str.strip()

    # Candidato 1: eliminar comas (quita comas de miles, punto decimal)
```

```
s1 = pd.to_numeric(s_str.str.replace(',', ' ', regex=False), errors='coerce')

# Candidato 2: formato colombiano ('.' miles, ',' decimal)
s2 = pd.to_numeric(
    s_str.str.replace('.', ' ', regex=False).str.replace(',', '.', regex=False),
    errors='coerce'
)

# Candidato 3: patrón muy común '###.000' -> miles
frac_ends_000 = s_str.str.endswith('.000').mean()
if frac_ends_000 >= 0.6:
    s3 = pd.to_numeric(s_str.str.replace('.', ' ', regex=False), errors='coerce') / 1000.0
else:
    s3 = pd.Series(np.nan, index=s.index)

cands = {'std': s1, 'co': s2, 'miles000': s3}

def score(v):
    med = np.nanmedian(v)
    if not np.isfinite(med): return np.inf
    # rango razonable según tu dataset (400 a 1000 aprox.)
    if med < 300 or med > 2000:
        return abs(med - 700) + 10000
    return abs(med - 700) # el más cercano a 700

best_key = min(cands.keys(), key=lambda k: score(cands[k]))
print("Medianas candidatas:",
      {k: (None if not np.isfinite(np.nanmedian(v)) else float(np.nanmedian(v))) for k,v in cands.items()})
print(f"Sección: {best_key}")
return cands[best_key]

df["precio_netto_original"] = df["precio_netto"] # para referencia
df["precio_netto"] = parse_precio_netto_auto(df["precio_netto"])

# 5) Outliers por IQR
def iqr_bounds(s: pd.Series, k=1.5):
    s = s.dropna()
    if s.empty: return np.nan, np.nan
    q1, q3 = s.quantile(0.25), s.quantile(0.75)
    iqr = q3 - q1
    return q1 - k*iqr, q3 + k*iqr

serie = pd.to_numeric(df["precio_netto"], errors="coerce")
low_iqr, high_iqr = iqr_bounds(serie, k=K_IQR)
df["out_pre_netto_iqr"] = (serie < low_iqr) | (serie > high_iqr)

# 6) Salidas
total_out = int(df["out_pre_netto_iqr"].sum())
print(f"[IQR k={K_IQR}] Umbrales precio_netto (PESOS): low={low_iqr:.2f}, high={high_iqr:.2f}")
print(f"Total outliers en 'Precioneto' por IQR: {total_out} de {len(df)} registros")

# Listado de outliers con columnas útiles
cols_export = [c for c in ["proveedor/centro_suministrador", "proveedor",
                           "textobreve", "texto_breve", "material",
                           "cantidaddepedido", "cantidad_pedido",
                           "fechadocumento", "fecha",
                           "precio_netto_original", "precio_netto", "out_pre_netto_iqr"] if c in df.columns]
olist = df.loc[df["out_pre_netto_iqr"], cols_export].copy()

olist.to_csv(f"{OUTDIR}/outliers_precioneto_listado.csv", index=False)
df.to_parquet(f"{OUTDIR}/base_etiquetada_outliers.parquet", index=False)

print("Archivos guardados en:", OUTDIR)

#@title 11) Correlaciones entre numéricas (si aplica)

#Calculamos matriz de correlación Pearson como base (más adelante podremos mirar Spearman/mixtas).
#Guardamos 06_correlacion_pearson.csv.

import numpy as np
import matplotlib.pyplot as plt

# Reusamos la separación hecha antes:
num_cols = [c for c in df.columns if pd.api.types.is_numeric_dtype(df[c])]
if "es_duplicado" in num_cols:
    num_cols.remove("es_duplicado") # quitar etiqueta booleana si quedó como 0/1
if "out_any" in num_cols:
    num_cols.remove("out_any")
if "out_iqr" in num_cols:
    num_cols.remove("out_iqr")
if "out_z3" in num_cols:
    num_cols.remove("out_z3")
if "out_rz35" in num_cols:
    num_cols.remove("out_rz35")

if len(num_cols) >= 2:
    corr = df[num_cols].corr(method="pearson").round(2)
    corr.to_csv(f"{OUTDIR}/06_correlacion_pearson.csv")

    fig, ax = plt.subplots(figsize=(max(6, 0.6*len(num_cols)), max(5, 0.6*len(num_cols))))
    im = ax.imshow(corr.values, cmap="coolwarm", vmin=-1, vmax=1)

    # Ticks y etiquetas
    ax.set_xticks(np.arange(len(num_cols)))
    ax.set_yticks(np.arange(len(num_cols)))
    ax.set_xticklabels(num_cols, rotation=45, ha="right")
    ax.set_yticklabels(num_cols)

    # Anotaciones con los valores
    for i in range(len(num_cols)):
        for j in range(len(num_cols)):
            val = corr.values[i, j]
            ax.text(j, i, f"{val:.2f}", ha="center", va="center", color="black")

    ax.set_title("Matriz de correlación (Pearson) con anotaciones")
    fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
    plt.tight_layout()
    plt.show()

    # Guardar imagen
    fig.savefig(f"{OUTDIR}/06_correlacion_pearson_annot.png", dpi=160, bbox_inches="tight")
else:
    print("No hay suficientes columnas numéricas para matriz de correlación anotada.")

#@title Filtra Precioneto > 2064 COP y grafica la serie temporal
PATH = "/content/drive/MyDrive/Base_Bolsas_Cementos.xlsx" # usa esta ruta tal cual

import pandas as pd
import numpy as np
import plotly.express as px

# 1) Cargar RAW
df = pd.read_excel(PATH, engine="openpyxl")
```

assert "Fechadocumento" in df.columns, "No se encontr i la columna 'Fechadocumento'."
assert "Precioneto" in df.columns, "No se encontr i la columna 'Precioneto'."

2) Copia para trabajo y columnas necesarias
df_plot = df[["Fechadocumento", "Precioneto"]].copy()

3) Convertir a datetime para ordenar en el eje X (no toca los valores de precio)
df_plot["Fechadocumento"] = pd.to_datetime(df_plot["Fechadocumento"], errors="coerce")

4) Convertir Precioneto a num rico S LO para poder filtrar y graficar (sin escalas ni agregaciones)
- Si hay separadores como coma de miles, los removemos para que no queden "billones" por error de formato
pre_raw = df_plot["Precioneto"].astype(str).str.strip()
pre_num = pd.to_numeric(pre_raw.str.replace(",", "", regex=False), errors="coerce")

5) Filtrar: conservar solo precios <= 2064 COP
mask = (pre_num.notna()) & (pre_num <= 2064)
df_plot = df_plot.loc[mask].copy()
df_plot["Precioneto_num"] = pre_num.loc[mask]

6) Ordenar por fecha
df_plot = df_plot.sort_values("Fechadocumento")

7) Gr fico: Y = Precioneto (COP) | X = Fechadocumento
fig = px.line(
 df_plot,
 x="Fechadocumento",
 y="Precioneto_num",
 title="Precio neto (COP) vs 2064 en el tiempo",
)
fig.update_yaxes(title_text="Precio neto (COP)", separatethousands=True)
fig.update_xaxes(title_text="Fecha")
Hover muestra valor num rico y el texto original por si quieres verificar
fig.update_traces(
 hovertemplate="{x}
Precioneto (COP): {y:,.0f}<extra></extra>"
)
fig.show()

#title 12 B Filtra Precioneto al rango [400, 2064] y grafica la serie temporal resultante los autliers inferior y supier.
PATH = "/content/drive/MyDrive/Base_Bolsas_Cementos .xlsx" # usa esta ruta tal cual

import pandas as pd
import numpy as np
import plotly.express as px

1) Cargar RAW
df = pd.read_excel(PATH, engine="openpyxl")

2) Verificar columnas requeridas
assert "Fechadocumento" in df.columns, "No se encontr i la columna 'Fechadocumento'."
assert "Precioneto" in df.columns, "No se encontr i la columna 'Precioneto'."

3) Copia de trabajo y parsing m nimo para poder filtrar y graficar (sin escalar ni agregar)
df_f = df[["Fechadocumento", "Precioneto"]].copy()
df_f["Fechadocumento"] = pd.to_datetime(df_f["Fechadocumento"], errors="coerce")

Parseo NUM RICO de Precioneto:
- Quitamos comas de miles si las hubiera; dejamos punto como decimal (p.ej., 697.000 -> 697.0)
pre_num = pd.to_numeric(df_f["Precioneto"].astype(str).str.replace(",", "", regex=False),
 errors="coerce")

4) Filtro de rango [400, 2064] COP
mask = (pre_num >= 400) & (pre_num <= 2064)
before, after = len(df_f), int(mask.sum())
df_f = df_f.loc[mask].copy()
df_f["Precioneto_num"] = pre_num.loc[mask]

print(f"Filtrados: {before - after} registros fuera de [400, 2064].")
print(f"Quedan: {after} registros dentro del rango.")

5) Ordenar por fecha y graficar
df_f = df_f.sort_values("Fechadocumento")

fig = px.line(
 df_f,
 x="Fechadocumento",
 y="Precioneto_num",
 title="Precio neto (COP) en el tiempo B  Rango [400, 2064]",
)
fig.update_yaxes(title_text="Precio neto (COP)", separatethousands=True)
fig.update_xaxes(title_text="Fecha")
fig.update_traces(hovertemplate="{x}
Precioneto (COP): {y:,.0f}<extra></extra>")
fig.show()

""Exploraci n de datos (RAW vs SAP).
Concluimos la fase exploratoria de la base RAW proveniente de SAP. La informaci n presenta alta dispersi n y no se identifican correlaciones directas y explicativas.
Pr ximo paso: construir una base limpia y homog nea, filtrando a la referencia de saco m ys comprada y al proveedor principal que la abastece, para reducir heterogeneidad.

PRIMERA LIMPIEZA

A CONTINUACI N, SE HARA LA VISUALIZACI N DE LOS DATOS DE LA BASE DE DATOS MEJORARA EN DONDE SOLO TENDREMOS EN CUENTA EL PRECIO DE LA REFERENCIA MAS COMPRADA 4500098
""

#title 1) Carga + estandarizaci n de nombres para la base de datos mejorada una sola referencia-un solo porveedor.
PATH = "/content/drive/MyDrive/BATES_50KG.xlsx"
OUTDIR = "/content/drive/MyDrive/smurfit_outputs/eda_bates50kg"

import os, re, numpy as np, pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt

os.makedirs(OUTDIR, exist_ok=True)

def snake_case(s: str) -> str:
 base = str(s).strip().lower()
 tr = {"T ": "a", "T ": "e", "T-": "i", "Ti": "o", "Te": "u", "T ": "n",
 "/" : "_", "\\" : " ", " " : "_", "-": " ", " ": "", ";": "", ":": "",
 "?": "", "B ": "", "B ": "", "!": "", "(" : " ", ")" : "", "[" : "", "]" : "", "{" : "", "}": ""}
 for k,v in tr.items(): base = base.replace(k,v)
 base = re.sub(r"__+", " ", base)
 base = re.sub(r"^[a-z0-9_]+", "", base)
 return base.strip("_")

df_raw = pd.read_excel(PATH, engine="openpyxl")
df = df_raw.rename(columns={c: snake_case(c) for c in df_raw.columns})

Mapear alias t picos
alias = {
 "proveedor/centro_suministrador": "proveedor",
 "textobreve": "texto_breve",
 "fechadocumento": "fecha",
 "cantidaddepedido": "cantidad_pedido",
 "precioneto": "precio_netto"
}

```
for k,v in alias.items():
    if k in df.columns and v not in df.columns:
        df = df.rename(columns={k:v})

# Chequeos mΓ-nimos
expected = ["proveedor","texto_breve","material","cantidad_pedido","precio_net","fecha"]
missing = [c for c in expected if c not in df.columns]
print("Columnas presentes:", list(df.columns))
print("Faltantes esperadas:", missing)

#@title 2) Tipos, nulos y duplicados, validaciΓn adicional a la que se le hizo de forma manual.
desc = pd.DataFrame({
    "columna": df.columns,
    "dtype_inferido": [str(df[c].dtype) for c in df.columns],
    "n_nulos": [df[c].isna().sum() for c in df.columns],
    "n_unicos": [df[c].nunique(dropna=True) for c in df.columns],
    "ejemplo": [df[c].dropna().iloc[0] if df[c].dropna().shape[0]>0 else None for c in df.columns],
})
display(desc.head(20))
desc.to_csv(f"{OUTDIR}/01_diccionario_preliminar.csv", index=False)

nulos = df.isna().sum().sort_values(ascending=False).to_frame("n_nulos")
nulos["pct"] = (nulos["n_nulos"]/len(df))*100
display(nulos.head(20))
nulos.to_csv(f"{OUTDIR}/02_nulos_por_columna.csv")

dup_count = df.duplicated().sum()
df["es_duplicado"] = df.duplicated()
print("Duplicados (filas completas):", dup_count)

df.to_parquet(f"{OUTDIR}/df_etiquetado_duplicados.parquet", index=False)

#@title 3) Fecha y cobertura temporal
fecha_col = "fecha" if "fecha" in df.columns else None
if fecha_col is None:
    for c in ["fechadocumento","fecha_documento","date"]:
        if c in df.columns: fecha_col = c; break

assert fecha_col is not None, "No se identificΓ columna de fecha (e.g., Fechadocumento/fecha)."

df[fecha_col] = pd.to_datetime(df[fecha_col], errors="coerce")
df = df.sort_values(fecha_col)
print("Rango temporal:", df[fecha_col].min(), "BΓ", df[fecha_col].max())

tmp = df[[fecha_col]].copy()
tmp["yyyymm"] = tmp[fecha_col].dt.to_period("M").astype(str)
cov = tmp["yyyymm"].value_counts().sort_index().to_frame("conteo")
display(cov.tail(12))
cov.to_csv(f"{OUTDIR}/03_cobertura_mensual.csv")

#@title 4) Perfil numΓrico/categΓrico de las variables aun existentes en la base reducida a un proveedor con una sola referencia
num_cols = [c for c in df.columns if pd.api.types.is_numeric_dtype(df[c])]
cat_cols = [c for c in df.columns if c not in num_cols and c not in [fecha_col, "es_duplicado"]]

print("NumΓricas:", num_cols)
print("CategΓricas:", cat_cols)

prof_num = df[num_cols].describe().T if num_cols else pd.DataFrame()
if not prof_num.empty:
    display(prof_num)
    prof_num.to_csv(f"{OUTDIR}/04_resumen_numericas.csv")

card = []
for c in cat_cols:
    vc = df[c].value_counts(dropna=False)
    card.append({
        "columna": c,
        "n_unicos": df[c].nunique(dropna=True),
        "top1_valor": (vc.index[0] if len(vc)>0 else None),
        "top1_freq": int(vc.iloc[0]) if len(vc)>0 else 0,
        "top1_pct": (int(vc.iloc[0])/len(df))*100 if len(df)>0 and len(vc)>0 else 0.0
    })
card = pd.DataFrame(card).sort_values("n_unicos", ascending=False)
display(card)
card.to_csv(f"{OUTDIR}/05_cardinalidad_categoricas.csv", index=False)

#@title 5) DistribuciΓn y outliers (IQR) para precio_net - se hace computacionalmente, ya que, se habia realizado manual.
assert "precio_net" in df.columns, "No se encontrΓ 'precio_net'."
df["precio_net"] = pd.to_numeric(df["precio_net"], errors="coerce")

def iqr_bounds(s: pd.Series, k=1.5):
    s = s.dropna()
    if s.empty: return np.nan, np.nan
    q1, q3 = s.quantile(0.25), s.quantile(0.75)
    iqr = q3 - q1
    return q1 - k*iqr, q3 + k*iqr

low_iqr, high_iqr = iqr_bounds(df["precio_net"], k=1.5)
print(f"[IQR] Umbrales precio_net: low={low_iqr:.2f}, high={high_iqr:.2f}")

fig = px.histogram(df, x="precio_net", nbins=50, title="DistribuciΓn B· precio_net")
fig.add_vline(x=low_iqr, line_dash="dash")
fig.add_vline(x=high_iqr, line_dash="dash")
fig.update_yaxes(title_text="Frecuencia")
fig.update_xaxes(title_text="precio_net (COP)")
fig.show()

plt.figure(figsize=(6,3))
plt.boxplot(df["precio_net"].dropna(), vert=False, whis=1.5)
plt.title("Boxplot B· precio_net")
plt.xlabel("precio_net (COP)")
plt.show()

"""Umbrales precio_net: low=409.50, high=797.50 pero se considera hasta los 1000 COP precio aceptable por conocimiento del negocio.

"""

#@title 6) Matriz de correlaciΓn con anotaciones
import numpy as np

num_cols2 = [c for c in df.columns if pd.api.types.is_numeric_dtype(df[c])]
# quitar banderas auxiliares si quedaron numΓricas
for aux in ["es_duplicado"]:
    if aux in num_cols2: num_cols2.remove(aux)

if len(num_cols2) >= 2:
    corr = df[num_cols2].corr(method="pearson").round(2)
    fig, ax = plt.subplots(figsize=(max(6, 0.6*len(num_cols2)), max(5, 0.6*len(num_cols2))))
    im = ax.imshow(corr.values, cmap="coolwarm", vmin=-1, vmax=1)

    ax.set_xticks(np.arange(len(num_cols2)))
    ax.set_yticks(np.arange(len(num_cols2)))
    ax.set_xticklabels(num_cols2, rotation=45, ha="right")
```

```
ax.set_yticklabels(num_cols2)

for i in range(len(num_cols2)):
    for j in range(len(num_cols2)):
        ax.text(j, i, f"{corr.values[i, j]:.2f}", ha="center", va="center", color="black")

ax.set_title("Matriz de correlaci3n (Pearson) entre valores anotados")
fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
plt.tight_layout()
plt.show()

corr.to_csv(f"{OUTDIR}/06_correlacion_pearson.csv")
else:
    print("No hay suficientes columnas num3ricas para matriz de correlaci3n.")
```

""Por conocimiento del negocio, la relación que hay entre la cantidad de pedido y el precio neto es inexistente, ya que, son precios pactados por un tiempo determinado

```
#title 7) Lf-nea temporal precio_netto + media mFivil
df_plot = df[[fecha_col, "precio_netto"]].dropna().copy()
df_plot = df_plot.sort_values(fecha_col)
df_plot["mm_rolling"] = df_plot["precio_netto"].rolling(window=28, min_periods=7).mean()

fig = px.line(df_plot, x=fecha_col, y="precio_netto",
              title="precio_netto (COP) en el tiempo BF base focalizada")
fig.update_traces(name="precio_netto (COP)",
                  hovertemplate="<x>{x}</b><br>precio_netto: {y:,.0f} COP<extra></extra>")

fig.add_scatter(x=df_plot[fecha_col], y=df_plot["mm_rolling"],
                mode="lines", name="Media mFivil (B%€28 dF-as)")
fig.update_yaxes(title_text="Precio netto (COP)", separatethousands=True)
fig.update_xaxes(title_text="Fecha")
fig.show()
```

```
# Guardar artefactos clave
df_plot[[fecha_col,"precio_netto","mm_rolling"]].to_csv(f"{OUTDIR}/07_serie_precios_mm.csv", index=False)
print("Artefactos guardados en:", OUTDIR)
```

""La tendencia es claramente alcista, con aceleraciTin a partir de 2021 post covid y saltos discretos que parecen ajustes de contrato y renegociaciTin de precios. Us

```
#@title Descripcion del Resumen de lo hecho anteriormente
import numpy as np
```

```
p25 = float(np.nanpercentile(df["precio_neto"], 25)) if df["precio_neto"].notna().any() else None
p50 = float(np.nanpercentile(df["precio_neto"], 50)) if df["precio_neto"].notna().any() else None
p75 = float(np.nanpercentile(df["precio_neto"], 75)) if df["precio_neto"].notna().any() else None
```

```
print(f"""
Resumen Б" BATES_50KG
Б" Observaciones: {len(df):},
Б" Rango temporal: {df[fecha_col].min()} Б" {df[fecha_col].max()}
Б" precio_netto Б" p25/p50/p75 (COP): {p25:,.0f} / {p50:,.0f} / {p75:,.0f}
Б" Duplicados (etiquetados, no eliminados): {int(df['es_duplicado'].sum()):},
Б" Umbrales IQR precio_netto (COP): low={low_iqr:,.0f}| high={high_iqr:,.0f}

```

```

siguiente (CRISP-DM B: Data Preparation):
1) Tipificaci3n final y creaci3n de seřales temporales (sin/cos mes/semana, mes, ařto).
2) Reglas de tratamiento de outliers (mantener informaci3n sin destruirla: winsorizaci3n/robust scaling si aplica).
3) Definir frecuencia de trabajo (D/W/M) y split temporal para backtesting.
4) Preparar dataset en formato de modelo listo para ARIMA/SARIMAX y, luego, LSTM.
"""

```

```
""# COMPARACION DE RENDIMIENTOS MODELOS ARIMA - SARIMAX- LSTM
**AHORA LO QUE HAREMOS SERA: PREPARAR LOS DATOS PARA ENTRENAR LOS TRES MODELOS ARIMA, SARIMAX Y LSTM. PARA LUEGO MEDIR PARAMETROS, COMPARAR Y ELEGIR.**
```

Preparar datos de /content/drive/MyDrive/BATES_50KG.xlsx

Dividir en Entrenamiento / Validación / Test (según el marco teórico)

Entrenar ARIMA, SARIMAX y LSTM

Reportar métricas (MAE, RMSE, sMAPE, MASE) por modelo

```

Ventanas de diseño (alineadas con tu nteproyecto/estado del arte):
- Serie univariada: precio_neto. Para SARIMAX a7adidos exPigenas simples de calendario (sin/cos estacional) y, si existe, cantidad_pedido.
- Divisi7in temporal: 70% train, 15% valid, 15% test (sin mezcla).
- Selecci7in ARIMA/SARIMAX: mini grid por AIC en train+valid (te7irico cl7f7ico) y evaluaci7in fuera de muestra en test.
- LSTM: ventanas deslizantes con W=12 por defecto (si trabajas semanal); puedes cambiar a diario/mensual ajustando FREQ y WINDOW.
"""

```

```
#@title 0) Configuraci3n y carga

#Estandarizamos nombres, parseamos fecha y definimos una frecuencia de trabajo (c3mbiala si lo requieres).
```

```

--- Par metros clave ---
PATH      = "/content/drive/MyDrive/BATES_50KG.xlsx"
TARGET    = "precio_neto"          # ser  el nombre estandarizado
DATE      = "fecha"               # se detecta a partir de 'Fechadocumento'
FREQ      = "w"                   # 'D' diario, 'W' semanal, 'M' mensual
WINDOW    = 12                    # tama o de ventana para LSTM (12 semanas por defecto)

```

```
# --- Instalaci3n de librer7as compatibles ---
```

```
!pip install -q --upgrade numpy scipy statsmodels tensorflow
```

```
import re, numpy as np, pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
```

```
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
# --- utilidades ---
def snake_case(s: str) -> str:
    base = str(s).strip().lower()
    tr = {"rY": "a", "r@": "e", "r-": "i", "ri": "o", "re": "u", "rt+": "n",
          "r ": " ", "r\\": " ", "r'": " ", "r_": " ", "r!": " ", "r%": " ", "r^": " ",
          "r~": " ", "Bj": "y", "BŸ": "ü", "B!": " ", "B-": " ", "B(": "(", "B)": ")", "B[": "[", "B]": "]", "B{": "{", "B}": "}"
    }
    for k,v in tr.items(): base = base.replace(k,v)
    base = re.sub(r"_+", "-", base)
    base = re.sub(r"[^a-z0-9_]+", "", base)
    return base.strip(" ")
```

```
def smape(y_true, y_pred):
    yt, yp = np.array(y_true), np.array(y_pred)
    denom = (np.abs(yt) + np.abs(yp)) / 2.0
    mask = denom != 0
    return np.mean( np.abs(yt[mask] - yp[mask]) / denom[mask] ) * 100
```

```
def mase(y_train, y_true, y_pred, m=1):
    # Escala por naive forecast con desfase m (m=1: naive de 1 paso)
```



```
y_train = np.asarray(y_train)
scale = np.mean(np.abs(y_train[m:] - y_train[:-m]))
return np.mean(np.abs(np.asarray(y_true) - np.asarray(y_pred))) / (scale if scale != 0 else 1)

# --- carga y estandarizaci3n ---
raw = pd.read_excel(PATH, engine="openpyxl")
df = raw.rename(columns={c: snake_case(c) for c in raw.columns})

# alias esperados
aliases = {
    "proveedor_centro_suministrador": "proveedor",
    "textobreve": "texto_breve",
    "fechadocumento": "fecha",
    "cantidaddepedido": "cantidad_pedido",
    "precioneto": "precio_netto"
}
for k,v in aliases.items():
    if k in df.columns and v not in df.columns:
        df = df.rename(columns={k:v})

assert "precio_netto" in df.columns, "No se encontr3i Precioneto/precio_netto."
# parseo de fecha
if "fecha" not in df.columns:
    for c in ["fechadocumento", "fecha_documento", "date"]:
        if c in df.columns:
            df = df.rename(columns={c: "fecha"})
            break
assert "fecha" in df.columns, "No se encontr3i columna de fecha."

df["fecha"] = pd.to_datetime(df["fecha"], errors="coerce")
df = df.sort_values("fecha").dropna(subset=["fecha"])
df["precio_netto"] = pd.to_numeric(df["precio_netto"], errors="coerce")

# agregaci3n/regularizaci3n a FREQ elegida (promedio del periodo)
ts = (df.set_index("fecha")
      .resample(FREQ) ["precio_netto"].mean()
      .asfreq(FREQ))

# exTigenas potenciales: cantidad_pedido (agregada) + estacionales
exog_df = pd.DataFrame(index=ts.index)
if "cantidad_pedido" in df.columns:
    exog_df["cantidad_pedido"] = (df.set_index("fecha")
                                  .resample(FREQ) ["cantidad_pedido"]
                                  .sum() # sum para volumen por periodo
                                  .reindex(ts.index))

# estacionales (mes/woy segTen frecuencia)
exog_df["sin_year"] = np.sin(2*np.pi*exog_df.index.dayofyear/365.25)
exog_df["cos_year"] = np.cos(2*np.pi*exog_df.index.dayofyear/365.25)

# dataset final limpio (drop de NaN sincronizados)
data = pd.concat([ts.rename(TARGET), exog_df], axis=1).dropna()
print("Longitud serie (", FREQ, "):", len(data))
display(data.tail())

#@title 1) Divisi3n temporal: Train / Valid / Test

#Mantiene orden temporal. Teor3a: evitamos look-ahead bias y reservamos test solo para evaluaci3n final.

# proporciones 70/15/15
n = len(data)
n_train = int(n*0.70)
n_valid = int(n*0.15)
n_test = n - n_train - n_valid

train = data.iloc[:n_train]
valid = data.iloc[n_train:n_train+n_valid]
test = data.iloc[n_train+n_valid:]

y_train, y_valid, y_test = train[TARGET], valid[TARGET], test[TARGET]
X_train = train.drop(columns=[TARGET])
X_valid = valid.drop(columns=[TARGET])
X_test = test.drop(columns=[TARGET])

print(f"Splits B+ train:{len(train)}, valid:{len(valid)}, test:{len(test)}")

#@title 2) ARIMA (endTigeno)

#Marco te3irico: ARIMA modela componente lineal y dependencia temporal. Seleccionamos (p,d,q) por AIC en un grid compacto sobre train+valid, luego evaluamos en test.
#M3tricas: MAE, RMSE, SMAPE, MASE (escala interpretable y comparables entre modelos).

import warnings
warnings.filterwarnings("ignore")

def arima_grid_search(y_tr, y_va, p_range=range(0,4), d_range=range(0,3), q_range=range(0,4)):
    best = {"aic": np.inf, "order": None}
    y_tv = pd.concat([y_tr, y_va]) # ajustar con train+valid para selecci3n
    for p in p_range:
        for d in d_range:
            for q in q_range:
                try:
                    model = ARIMA(y_tv, order=(p,d,q))
                    res = model.fit(method_kwars={"warn_convergence":False})
                    if res.aic < best["aic"]:
                        best = {"aic": res.aic, "order": (p,d,q)}
                except Exception:
                    continue
    return best

best_arima = arima_grid_search(y_train, y_valid)
print("Mejor ARIMA por AIC:", best_arima)

# Refit con train+valid y pronTistico sobre test
y_tv = pd.concat([y_train, y_valid])
arima_fit = ARIMA(y_tv, order=best_arima["order"]).fit()

# one-shot forecast igual al largo de test
fc_arima = arima_fit.forecast(steps=len(y_test))
fc_arima.index = y_test.index

# m3tricas
def metrics_block(y_true, y_pred, y_train_ref):
    mae = mean_absolute_error(y_true, y_pred)
    rmse = mean_squared_error(y_true, y_pred)**0.5 # Calculate RMSE by taking the square root
    sm = smape(y_true, y_pred)
    ms = mase(y_train_ref, y_true, y_pred, m=1)
    return {"MAE":mae, "RMSE":rmse, "sMAPE%":sm, "MASE":ms}

metrics_arima = metrics_block(y_test, fc_arima, y_train)
metrics_arima

#@title 3) SARIMAX (ARIMA + exTigenas)

#Marco te3irico: incorpora variables exTigenas (estacionales y de volumen) para explicar variaci3n adicional. Selecci3n (p,d,q) por AIC con mismas exTigenas; evaluaci3n
```

```
def sarimax_grid_search(y_tr, y_va, X_tr, X_va, p_range=range(0,4), d_range=range(0,3), q_range=range(0,4)):
    best = {"aic": np.inf, "order": None}
    y_tv = pd.concat([y_tr, y_va])
    X_tv = pd.concat([X_tr, X_va])
    for p in p_range:
        for d in d_range:
            for q in q_range:
                try:
                    model = SARIMAX(y_tv, exog=X_tv, order=(p,d,q), enforce_stationarity=False, enforce_invertibility=False)
                    res = model.fit(dispatch=False)
                    if res.aic < best["aic"]:
                        best = {"aic": res.aic, "order": (p,d,q)}
                except Exception:
                    continue

    return best

best_sarimax = sarimax_grid_search(y_train, y_valid, X_train, X_valid)
print("Mejor SARIMAX por AIC:", best_sarimax)

# Refit con train+valid y forecast sobre test (con exógenas de test)
y_tv, X_tv = pd.concat([y_train, y_valid]), pd.concat([X_train, X_valid])
sarimax_fit = SARIMAX(y_tv, exog=X_tv, order=best_sarimax["order"],
                      enforce_stationarity=False, enforce_invertibility=False).fit(dispatch=False)

fc_sarimax = sarimax_fit.forecast(steps=len(y_test), exog=X_test)
fc_sarimax.index = y_test.index

metrics_sarimax = metrics_block(y_test, fc_sarimax, y_train)
metrics_sarimax

#@title 4) LSTM (red recurrente)

#Marco teórico: LSTM capta no linealidades y dependencias de largo plazo.
#Preparación: normalizamos precio neto (y exógenas si las usas), generamos secuencias de longitud WINDOW y validamos en test.
#Arquitectura mF-nima y early stopping (práctica común).
#Si no deseas exógenas en LSTM, usa solo precio_neto.

# ---- Preparación de datos para LSTM (solo endógena precio_neto, simple) ----
series_all = data[TARGET].astype(float)

# Split igual que arriba (usamos mismos índices)
series_train = series_all.loc[y_train.index]
series_valid = series_all.loc[y_valid.index]
series_test = series_all.loc[y_test.index]

# Escalado MinMax en train (evita fuga de info)
scaler = MinMaxScaler()
series_train_sc = pd.DataFrame(scaler.fit_transform(series_train), index=series_train.index, columns=[TARGET])
series_valid_sc = pd.DataFrame(scaler.transform(series_valid), index=series_valid.index, columns=[TARGET])
series_test_sc = pd.DataFrame(scaler.transform(series_test), index=series_test.index, columns=[TARGET])

def make_sliding(X, window):
    Xs, ys = [], []
    vals = X.values.astype(np.float32)
    for i in range(window, len(vals)):
        Xs.append(vals[i-window:i, :])
        ys.append(vals[i, 0])
    return np.array(Xs), np.array(ys)

X_tr, y_tr = make_sliding(series_train_sc, WINDOW)
X_va, y_va = make_sliding(pd.concat([series_train_sc.tail(WINDOW), series_valid_sc]), WINDOW)
X_te, y_te = make_sliding(pd.concat([series_valid_sc.tail(WINDOW), series_test_sc]), WINDOW)

X_tr.shape, X_va.shape, X_te.shape

# ---- Modelo LSTM ----
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping

tf.random.set_seed(42)

model = Sequential([
    LSTM(32, input_shape=(X_tr.shape[1], X_tr.shape[2])),
    Dense(1)
])

model.compile(optimizer="adam", loss="mse")
es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)

hist = model.fit(
    X_tr, y_tr,
    validation_data=(X_va, y_va),
    epochs=200,
    batch_size=32,
    callbacks=[es],
    verbose=0
)

# Predicción sobre test (a escala MinMax); luego desescalar
y_pred_sc = model.predict(X_te).ravel()
# Reconstruir vectores verdaderos en el mismo alineamiento
y_true_sc = y_te

# invertir escala
y_pred = scaler.inverse_transform(y_pred_sc.reshape(-1,1)).ravel()
y_true = scaler.inverse_transform(y_true_sc.reshape(-1,1)).ravel()

metrics_lstm = {
    "MAE": mean_absolute_error(y_true, y_pred),
    "RMSE": mean_squared_error(y_true, y_pred)**0.5,
    "sMAPE%": smape(y_true, y_pred),
    "MASE": mase(series_train[TARGET].values, y_true, y_pred, m=1)
}
metrics_lstm

#@title 5) Comparación rápida de resultados (por separado + cuadro)

# Primero miras rendimiento individual arriba; aquí los reunimos en un cuadro para visualizar.

results = pd.DataFrame([
    {"Modelo": "ARIMA", **metrics_arima},
    {"Modelo": "SARIMAX", **metrics_sarimax},
    {"Modelo": "LSTM", **metrics_lstm},
]).set_index("Modelo")
results
```


#@title 6) Gráficos de pronóstico vs real (cada modelo)

```
fig_arima = px.line(pd.DataFrame({"Real":y_test, "ARIMA":fc_arima}), title="ARIMA vs Real vs Pronóstico")
fig_arima.update_yaxes(title_text="Precio neto (COP)", separatethousands=True); fig_arima.update_xaxes(title_text="Fecha")
fig_arima.show()
```

```
fig_sarimax = px.line(pd.DataFrame({"Real":y_test, "SARIMAX":fc_sarimax}), title="SARIMAX vs Real vs Pronóstico")
fig_sarimax.update_yaxes(title_text="Precio neto (COP)", separatethousands=True); fig_sarimax.update_xaxes(title_text="Fecha")
fig_sarimax.show()
```

```
df_lstm_plot = pd.DataFrame({"Real": y_true, "LSTM": y_pred}, index=y_test.index[-len(y_true):])
fig_lstm = px.line(df_lstm_plot, title="LSTM vs Real vs Pronóstico")
fig_lstm.update_yaxes(title_text="Precio neto (COP)", separatethousands=True); fig_lstm.update_xaxes(title_text="Fecha")
fig_lstm.show()
```

"""7) Razones teóricas

ARIMA (p,d,q): capta estructura lineal y dependencia; elegimos (p,d,q) minimizando AIC en train+valid para balancear ajuste vs. parsimonia.

SARIMAX: añade exógenas (estacionales y volumen) para explicar variaciones; misma selección AIC sobre train+valid; evaluación en test.

LSTM: arquitectura capaz de no linealidad y dependencias de largo plazo; ventana WINDOW y early stopping evitan sobreajuste; evaluación en test tras desescalar.

Métricas:

MAE y RMSE (error absoluto y cuadrático: interpretación en COP y penalización a picos).

sMAPE (porcentaje simétrico, independiente de la escala).

MASE (normaliza contra un naive, útil para comparar familias de modelos).

****Con esos números, el LSTM es el mejor: tiene los menores MAE (81.4), RMSE (83.1) y sMAPE (9.54%). ARIMA queda 2do y SARIMAX el peor (exógenas no ayudaron en esta ocasión).**

Por qué gana el LSTM

Errores absolutos y cuadráticos más bajos → predice más cerca del valor real en promedio y con menos picos grandes de error.

Captura mejor no linealidades y posibles cambios de régimen (tus meses escalones).

Señales/alertas (importantes)

MASE muy > 1 en los tres modelos (ver el 21). Usualmente MASE < 1 indica que el modelo supera a un naive (random walk/naive estacional). Aquí sugiere que:

El naive usado podría ser muy fuerte para este horizonte/serie (por los escalones), o

Hay un desalineamiento en el cálculo (p. ej., comparas 1-paso con predicciones multi-paso, o el denominador del MASE se calcula en otra frecuencia/ventana).
Revisa la implementación del MASE y que el horizonte y la frecuencia coincidan con el naive.

ENTRENAMIENTO DE MODELO LSTM PARA TRES BASES DE DATOS DISTINTAS CON ADICIÓN DE VARIABLES EXÓGENAS PARA MEDIR MEJORAS Y AJUSTES BASE DE DATOS: BASE -BASE+TRM Y BASE+ACCIONES

Vamos a entrenar tres LSTM (base + +TRM + +TRM+Acciones) y a medir cómo mejora el rendimiento al añadir exógenas.
El siguiente bloque es Colab-ready, sin instalaciones ni pasos duplicados. Solo copia/pega y ejecuta en este orden. Incluye:

Carga y estandarización de columnas (soporta tus nombres exactos).

Agregación a frecuencia semanal (cámbiala si lo requieres).

Correlaciones (con números anotados en el heatmap) entre precio_neto y exógenas por cada base.

División temporal train/valid/test (70/15/15) sobre todos los datos disponibles.

Preparación de ventanas para LSTM, escalado solo con train (evitar fuga).

Entrenamiento LSTM con early stopping.

Métricas: MAE, RMSE, sMAPE, MASE (valid y test).

Gráfico único por variante donde se ve lo entrenado, validado y testeado (con bandas de color y predicciones).

Marco teórico aplicado:
• División temporal 70/15/15 (train/valid/test) para evitar look-ahead.
• LSTM con ventana WINDOW=12 (semanas) y EarlyStopping (paciencia=10).
• Escalado MinMax ajustado solo con train (sin fuga de información).
• MASE respecto a naive(1) sobre train (escala interpretable y comparable).
"""

```
#@title === Parámetros del experimento ===
PATHS = {
    "BASE":      "/content/drive/MyDrive/BATES_50KG.xlsx",
    "TRM":       "/content/drive/MyDrive/BATES_50KG_TRM.xlsx",
    "TRM+ACC":   "/content/drive/MyDrive/BATES_50KG_TRM+ACCIONES.xlsx",
}
FREQ    = "W"      # 'D' diario, 'W' semanal, 'M' mensual (ajusta si lo requieres)
WINDOW  = 12       # tamaño de ventana para LSTM (12 semanas por defecto)
BATCH   = 32
EPOCHS  = 200
PATIENCE = 10      # early stopping
```

```
TARGET_COL_CANON = "precio_neto"
```

```
import re, numpy as np, pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt
from pathlib import Path
```

```
!pip install --upgrade numpy scikit-learn
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import LSTM, Dense
from tensorflow.keras.callbacks import EarlyStopping
```

```
tf.random.set_seed(42)
np.random.seed(42)
```

```
# ----- utils: nombres y numéricos -----
def snake_case(s: str) -> str:
    base = str(s).strip().lower()
    tr = {"Ÿ": "a", "©": "e", "r-": "i", "ri": "o", "re": "u", "r±": "n",
           "/" : " ", "\ " : " ", " " : " ", "-": " ",
           ",": " ", " ": " ", "¿": " ", "Bí": " ", "Ÿ": " ", "¡": " ",
           "(": " ", ")": " ", "[" : " ", "]" : " ", "{": " ", "}" : " "}
    for k,v in tr.items(): base = base.replace(k,v)
    base = re.sub(r"__+", " ", base)
    base = re.sub(r"^[a-z0-9_]+", "", base)
    return base.strip("_")
```

```
def standardize_columns(df: pd.DataFrame) -> pd.DataFrame:
```

```
df = df.rename(columns={c: snake_case(c) for c in df.columns})
alias = {
    "proveedor_centro_suministrador": "proveedor",
    "proveedor/centro_suministrador": "proveedor",
    "texto_breve": "texto_breve",
    "textobreve": "texto_breve",
    "material": "material",
    "cantidad_depedido": "cantidad_pedido",
    "cantidaddepedido": "cantidad_pedido",
    "precio_neto": "precio_neto",
    "precioneto": "precio_neto",
    "fecha_documento": "fecha",
    "fechadocumento": "fecha",
    "trm": "trm",
    "klabin": "klabin",
    "mondi": "mondi",
    "smurfit": "smurfit",
}
for k,v in alias.items():
    if k in df.columns and v not in df.columns:
        df = df.rename(columns={k: v})
return df

def to_numeric_robust(s: pd.Series) -> pd.Series:
    """Convierte strings con posibles separadores de miles/decimales a n mero."""
    s = s.astype(str).str.strip()
    # 1) quitar comas de miles, interpretar punto como decimal
    s1 = pd.to_numeric(s.str.replace(",", "", regex=False), errors="coerce")
    # 2) si hay muchos '.000' puede venir en miles: detecta y corrige
    if s.str.endswith(".000").mean() >= 0.6:
        s2 = pd.to_numeric(s.str.replace(".", "", regex=False), errors="coerce") / 1000.0
        # elige la serie con mediana dentro de un rango razonable
        med1 = np.nanmedian(s1); med2 = np.nanmedian(s2)
        if (300 <= (med2 if np.isfinite(med2) else -1) <= 2000) or (not np.isfinite(med1)):
            return s2
    return s1

# ----- m tricas y secuencias -----
def smape(y_true, y_pred):
    yt, yp = np.array(y_true), np.array(y_pred)
    denom = (np.abs(yt) + np.abs(yp)) / 2.0
    mask = denom != 0
    return np.mean(np.abs(yt[mask] - yp[mask]) / denom[mask]) * 100

def mase(y_train, y_true, y_pred, m=1):
    y_train = np.asarray(y_train)
    scale = np.mean(np.abs(y_train[m:] - y_train[:-m]))
    return np.mean(np.abs(np.asarray(y_true) - np.asarray(y_pred))) / (scale if scale != 0 else 1)

def make_sliding(X: pd.DataFrame, y: pd.Series, window: int):
    Xs, ys = [], []
    Xv = X.values.astype(np.float32)
    yv = y.values.astype(np.float32)
    for i in range(window, len(X)):
        Xs.append(Xv[i-window:i, :])
        ys.append(yv[i])
    return np.array(Xs), np.array(ys)

def metrics_block(y_true, y_pred, y_train_ref):
    return {
        "MAE": mean_absolute_error(y_true, y_pred),
        "RMSE": mean_squared_error(y_true, y_pred, squared=False),
        "sMAPE%": smape(y_true, y_pred),
        "MASE": mase(y_train_ref, y_true, y_pred, m=1),
    }

#@title 1) Carga, limpieza ligera y resampleo por variante

#Qu  hace: Para cada dataset, carga, estandariza, tipa fecha, convierte precio_neto y ex genas a num rico, y resamplea a FREQ (promedio del precio y suma del volumen)

def load_variant(path: str, freq="W"):
    df = pd.read_excel(path, engine="openpyxl")
    df = standardize_columns(df)
    assert "fecha" in df.columns, f"No se encontr  columna de fecha en {Path(path).name}"
    assert TARGET_COL_CANON in df.columns, f"No se encontr  {TARGET_COL_CANON} en {Path(path).name}"

    # Tipos
    df["fecha"] = pd.to_datetime(df["fecha"], errors="coerce")
    df = df.dropna(subset=["fecha"]).sort_values("fecha")

    # Num ricos robustos
    df["precio_neto"] = to_numeric_robust(df["precio_neto"])
    if "cantidad_pedido" in df.columns:
        df["cantidad_pedido"] = to_numeric_robust(df["cantidad_pedido"])
    for ex in ["trm", "klabin", "mondi", "smurfit"]:
        if ex in df.columns:
            df[ex] = to_numeric_robust(df[ex])

    # Resampleo a freq: precio promedio, cantidad sumada
    grp = df.set_index("fecha")
    y = grp["precio_neto"].resample(freq).mean()

    X = pd.DataFrame(index=y.index)
    if "cantidad_pedido" in df.columns:
        X["cantidad_pedido"] = grp["cantidad_pedido"].resample(freq).sum().reindex(y.index)

    # Se ales estacionales (anuales) Ё  teor a SARIMAX/LSTM: ayudan a capturar estacionalidad
    X["sin_year"] = np.sin(2*np.pi*X.index.dayofyear/365.25)
    X["cos_year"] = np.cos(2*np.pi*X.index.dayofyear/365.25)

    # Ex genas si existen
    for ex in ["trm", "klabin", "mondi", "smurfit"]:
        if ex in df.columns:
            X[ex] = grp[ex].resample(freq).mean().reindex(y.index)

    # Dataset final: drop NaN sincronizados
    data = pd.concat([y.rename(TARGET_COL_CANON), X], axis=1).dropna()
    return data

datasets = {}
for name, p in PATHS.items():
    print(f"Cargando {name} -> {p}")
    datasets[name] = load_variant(p, freq=FREQ)
    display(datasets[name].tail(3))

#@title 2) Correlaciones (target vs ex genas) por variante

#Qu  hace: Muestra la matriz (Pearson) y anota valores en cada celda; enfoca lectura en la columna precio_neto.

def plot_corr_annot(df: pd.DataFrame, title="Correlaciones"):
    num_cols = [c for c in df.columns if pd.api.types.is_numeric_dtype(df[c])]
    if len(num_cols) < 2:
```

```
        print("No hay suficientes columnas numéricas para correlación.")
        return
corr = df[num_cols].corr(method="pearson").round(2)

fig, ax = plt.subplots(figsize=(max(6, 0.6*len(num_cols)), max(5, 0.6*len(num_cols))))
im = ax.imshow(corr.values, cmap="coolwarm", vmin=-1, vmax=1)
ax.set_xticks(np.arange(len(num_cols))); ax.set_yticks(np.arange(len(num_cols)))
ax.set_xticklabels(num_cols, rotation=45, ha="right"); ax.set_yticklabels(num_cols)
for i in range(len(num_cols)):
    for j in range(len(num_cols)):
        ax.text(j, i, f"{corr.values[i,j]:.2f}", ha="center", va="center", color="black")
ax.set_title(title)
fig.colorbar(im, ax=ax, fraction=0.046, pad=0.04)
plt.tight_layout(); plt.show()

for name, dfv in datasets.items():
    print(f"== Correlaciones {name} ==")
    plot_corr_annot(dfv, title=f"Correlaciones {name} (Pearson)")
    if any(c != TARGET_COL_CANON for c in dfv.columns):
        print(dfv.corr()[TARGET_COL_CANON].sort_values(ascending=False))

#@title 3) Split temporal y visualización de segmentos

#Quito hace: 70/15/15 Train/Valid/Test y grafica la serie completa con marcas de corte.

def temporal_split(data: pd.DataFrame, target=TARGET_COL_CANON, tr=0.70, va=0.15):
    n = len(data)
    n_tr = int(n*tr); n_va = int(n*va); n_te = n - n_tr - n_va
    train = data.iloc[:n_tr]
    valid = data.iloc[n_tr:n_tr+n_va]
    test = data.iloc[n_tr+n_va:]
    y_train = train[target].astype(float).to_numpy()
    y_valid = valid[target].astype(float).to_numpy()
    y_test = test[target].astype(float).to_numpy()
    return train, valid, test, y_train, y_valid, y_test

def plot_splits(data, train, valid, test, name=""):
    fig = px.line(data, y=TARGET_COL_CANON, title=f"{name} B· precio neto (COP) con cortes Train/Valid/Test")
    for cut, label in [(valid.index[0], "Valid start"), (test.index[0], "Test start")]:
        # Convert Timestamp to string for plotting and remove annotation
        fig.add_vline(x=str(cut), line_dash="dash") # Removed annotation_text and annotation_position
    fig.update_yaxes(title_text="Precio neto (COP)", separatethousands=True)
    fig.update_xaxes(title_text="Fecha")
    fig.show()

splits = {}
for name, dfv in datasets.items():
    tr, va, te, y_tr, y_va, y_te = temporal_split(dfv)
    splits[name] = (tr, va, te)
    plot_splits(dfv, tr, va, te, name=name)
    print(name, "B+", tuple(map(len, (tr,va,te))))

#@title 4) Preparación de secuencias y escalado por variante

#Quito hace: Escala con MinMax (fit solo en train para evitar fuga), concatena exógenas si hay, y crea ventanas deslizantes.

def prepare_lstm_data(train, valid, test, window=WINDOW, target=TARGET_COL_CANON):
    # Separar X (todas menos target) e y
    X_tr, y_tr = train.drop(columns=[target]), train[target]
    X_va, y_va = valid.drop(columns=[target]), valid[target]
    X_te, y_te = test.drop(columns=[target]), test[target]

    # Escalado por columnas usando MinMax (fit en train)
    scaler_X = MinMaxScaler()
    scaler_y = MinMaxScaler()

    X_tr_sc = pd.DataFrame(scaler_X.fit_transform(X_tr), index=X_tr.index, columns=X_tr.columns)
    X_va_sc = pd.DataFrame(scaler_X.transform(X_va), index=X_va.index, columns=X_va.columns)
    X_te_sc = pd.DataFrame(scaler_X.transform(X_te), index=X_te.index, columns=X_te.columns)

    y_tr_sc = pd.Series(scaler_y.fit_transform(y_tr.values.reshape(-1,1)).ravel(), index=y_tr.index, name=target)
    y_va_sc = pd.Series(scaler_y.transform(y_va.values.reshape(-1,1)).ravel(), index=y_va.index, name=target)
    y_te_sc = pd.Series(scaler_y.transform(y_te.values.reshape(-1,1)).ravel(), index=y_te.index, name=target)

    # Ventanas
    Xtr, ytr = make_sliding(pd.concat([X_tr_sc, y_tr_sc], axis=1), y_tr_sc, window) # incluye target como última col para contexto
    Xva, yva = make_sliding(pd.concat([X_va_sc, y_va_sc], axis=1), y_va_sc, window)
    Xte, yte = make_sliding(pd.concat([X_te_sc, y_te_sc], axis=1), y_te_sc, window)

    return (Xtr, ytr, Xva, yva, Xte, yte, scaler_y)

prep = {}
for name, (tr,va,te) in splits.items():
    prep[name] = prepare_lstm_data(tr, va, te, window=WINDOW)
    print(name, "shapes:", tuple(a.shape for a in prep[name][:6]))

# =====
#@title LSTM BASE (sin TRM): BATES 50KG.xlsx
# Vars: Proveedor/Centro suministrador, Textobreve, Material,
#       Cantidaddepedido, Precioneto, Fechadocumento
# Marco teórico: normalización, one-hot con rarefacción, split temporal,
# ventanas deslizantes, LSTM (128B+64) + LayerNorm, regularización y callbacks.
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KG.xlsx"
df = pd.read_excel(PATH)
```

```
# ===== 2) Normalizar nombres y mapear a canFinicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"^[^\\w\\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "proveedor/centro_suministrador": "proveedor",
    "proveedor_centro_suministrador": "proveedor",
    "textobreve": "texto_breve",
    "cantidaddepedido": "cantidad_pedido",
    "cantidad_depedido": "cantidad_pedido",
    "precioneto": "precio_netto",
    "fecha_documento": "fecha_documento",
    "fechadocumento": "fecha_documento",
    "material": "material",
}

df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) Filtrar estrictamente columnas base =====
required = ["proveedor", "texto_breve", "material", "cantidad_pedido", "precio_netto", "fecha_documento"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"

df = df[required].copy()

# ===== 4) Parseo de fecha, orden temporal y features de calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# Seftales de estacionalidad (opcionales pero fctiles y no exFigenas)
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_netto"
cat_cols = ["proveedor", "texto_breve", "material"]
num_base = ["cantidad_pedido", "sin_month", "cos_month", "sin_dow", "cos_dow"] # sPilo numFOricas internas

# ===== 6) RarefacciFin top-K para categFiricas =====
TOP_K = 50
def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")

top_sets = {c: topk_set(df_train[c], TOP_K) for c in cat_cols}
for c in cat_cols:
    df_train[c] = to_topk(df_train[c], top_sets[c])
    df_val[c] = to_topk(df_val[c], top_sets[c])
    df_test[c] = to_topk(df_test[c], top_sets[c])

# ===== 7) Escalado: X con StandardScaler, y con MinMax =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

# OneHot categFiricas + StandardScaler numFOricas
ohe = OneHotEncoder(handle_unknown="ignore")
from sklearn.preprocessing import FunctionTransformer

# Prepara matrices tabulares para cada split
def make_tab(df_part):
    X_num = df_part[num_base].to_numpy(dtype=np.float32)
    X_cat = ohe.fit_transform(df_train[cat_cols]).toarray() if df_part is df_train else ohe.transform(df_part[cat_cols]).toarray()
    return np.hstack([X_num, X_cat]).astype(np.float32)

# Ajuste del OHE sPilo con TRAIN
ohe.fit(df_train[cat_cols])
X_train_tab = make_tab(df_train)
X_val_tab = make_tab(df_val)
X_test_tab = make_tab(df_test)

# StandardScaler para numFOricas + mantener OHE ya binario
num_scaler = StandardScaler().fit(df_train[num_base].to_numpy(dtype=np.float32))
def scale_tab(X_tab):
    n_num = len(num_base)
    Xn = X_tab.copy()
    Xn[:, :n_num] = num_scaler.transform(Xn[:, :n_num])
    return Xn

X_train_tab = scale_tab(X_train_tab)
X_val_tab = scale_tab(X_val_tab)
X_test_tab = scale_tab(X_test_tab)

# ===== 8) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
    np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
    np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 9) Modelo LSTM (128->64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
```

```
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 10) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 11) Evaluaci3n y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva).ravel()
pred_te = model.predict(Xte).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACI3N ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 12) Gr1ficas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de p1rdida"); plt.xlabel("T3poca"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicci3n")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("T3ndice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 13) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_base_precio_net0.keras"))
import joblib
joblib.dump(ohe, os.path.join(SAVE_DIR, "ohe_categorias.joblib"))
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 14) Notas de tuning r1fido =====
# - Ajusta LOOKBACK (p.ej. 28, 60, 90) seg1en autocorrelaci3n/estacionalidad.
# - Si hay alta cardinalidad en categor1icas: baja TOP_K o filtra categor1as muy raras.
# - Sobreajuste: aumenta dropout/recurrent_dropout o L2; reduce unidades.
# - Infraajuste: aumenta unidades/capas o LOOKBACK.

#@title ===== Gr1fico final corregido: Real vs LSTM con fechas alineadas =====
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# 1) Predicciones de TRAIN (si no las ten1as calculadas)
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas con el tama1o real de cada bloque de predicci3n
# - TRAIN: len = len(df_train) - LOOKBACK -> fechas desde LOOKBACK
# - VALID: len = len(df_val) - 1 -> fechas desde 1
# - TEST : len = len(df_test) - 1 -> fechas desde 1
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos r1fidos de longitudes
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"

# 4) Concatenar todo para una sola l1-ne3 LSTM
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_net0"].to_numpy()

# 6) Plot
plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)

plt.title("Precio neto (COP) B3 Real vs LSTM (2014B32023)")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.xlim(pd.Timestamp("2014-01-01"), pd.Timestamp("2023-12-31"))
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

#@title Grafico #2 del rendimiento LSTM inicial
import numpy as np, matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error, mean_squared_error
import math
```



```
# --- 1) Serie temporal con cortes y predicciones ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_netto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_netto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_netto"], label="TEST (real)", alpha=0.8)

# Predicciones (ya alineadas con las fechas correctas)
ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

# Cortes
ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validaci3n", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title("LSTM B· precio_netto (COP) B· Real vs Predicho")
ax.set_ylabel("Precio netto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

# M3tricas (solo TEST) anotadas
mae = mean_absolute_error(df_test["precio_netto"].iloc[1:].values, pred_te_u)
rmse = math.sqrt(mean_squared_error(df_test["precio_netto"].iloc[1:].values, pred_te_u))
def smape(a,f):
    a=np.array(a); f=np.array(f); d=(np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:-m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)

mase_val = mase(df_train["precio_netto"].values, df_test["precio_netto"].iloc[1:].values, pred_te_u, m=1)
txt = f"TEST B· MAE={mae:,.1f} | RMSE={rmse:,.1f} | SMAPE={smape(df_test['precio_netto'].iloc[1:], pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))

plt.tight_layout(); plt.show()

# --- 2) Curvas de entrenamiento ---
fig, ax = plt.subplots(figsize=(8,3.5))
ax.plot(history.history["loss"], label="loss (train)")
ax.plot(history.history["val_loss"], label="val_loss")
ax.set_title("Curva de p3rdida (MSE)"); ax.set_xlabel("T3poca"); ax.grid(True, alpha=0.3); ax.legend(); plt.tight_layout(); plt.show()

# --- 3) Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
y_true = df_test["precio_netto"].iloc[1:].values
ax.scatter(y_true, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true.min(), pred_te_u.min()), max(y_true.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1) # l3-nea y=x
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title("Paridad en Test"); ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

# --- 4) Residuos en TEST ---
res = y_true - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
# 4a: residuos vs tiempo (T-ndice de ventana)
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("T3ndice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
# 4b: histograma residuos
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

# =====
#@title LSTM + TRM: BATES_50KG_TRM.xlsx
# Vars: Proveedor/Centro suministrador, Textobreve, Material,
# Cantidaddepedido, Precioneto, Fechadocumento, TRM
# Pipeline: normalizaci3n, one-hot con rarefaci3n top-K, split temporal,
# ventanas deslizantes, LSTM (128B+64) + LayerNorm, regularizaci3n y callbacks.
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KG_TRM.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y mapear a can3nicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"[^\\w\\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "proveedor/centro_suministrador": "proveedor",
    "proveedor_centro_suministrador": "proveedor",
    "textobreve": "texto_breve",
    "cantidaddepedido": "cantidad_pedido",
    "cantidad_depedido": "cantidad_pedido",
    "precioneto": "precio_netto",
    "fecha_documento": "fecha_documento",
    "fechadocumento": "fecha_documento",
    "material": "material",
    "trm": "trm",
```

```
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) Filtrar estrictamente columnas requeridas (incluye TRM) =====
required = ["proveedor", "texto_breve", "material", "cantidad_pedido", "precio_net", "fecha_documento", "trm"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"

df = df[required].copy()

# ===== 4) Tipos, parseo de fecha, orden temporal y features de calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
# Forzar numT@ricos
for c in ["cantidad_pedido", "precio_net", "trm"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

# Orden y limpieza mT-nima
df = df.dropna(subset=["fecha_documento", "precio_net"]).sort_values("fecha_documento").reset_index(drop=True)

# Llenado simple de TRM (si hay huecos): primero hacia adelante y luego hacia atrTŸs
if df["trm"].isna().any():
    df["trm"] = df["trm"].ffill().bfill()

# SeTtales de estacionalidad internas (no exTigenas externas)
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_net"
cat_cols = ["proveedor", "texto_breve", "material"]
# numT@ricos internas + TRM
num_base = ["cantidad_pedido", "trm", "sin_month", "cos_month", "sin_dow", "cos_dow"]

# ===== 6) RarefacciTn top-K para categTricas =====
TOP_K = 50
def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")

top_sets = {c: topk_set(df_train[c], TOP_K) for c in cat_cols}
for c in cat_cols:
    df_train[c] = to_topk(df_train[c], top_sets[c])
    df_val[c] = to_topk(df_val[c], top_sets[c])
    df_test[c] = to_topk(df_test[c], top_sets[c])

# ===== 7) Escalado: y con MinMax; X num con StandardScaler; OHE robusto =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

# OHE solo con TRAIN
ohe = OneHotEncoder(handle_unknown="ignore")
ohe.fit(df_train[cat_cols])

def to_dense(X):
    # Convierte a denso solo si es matriz dispersa (compatibilidad sklearn 1.1B"1.5+)
    return X.toarray() if hasattr(X, "toarray") else X

def make_tab(df_part):
    X_num = df_part[num_base].to_numpy(dtype=np.float32)
    X_cat = to_dense(ohe.transform(df_part[cat_cols]))
    return np.hstack([X_num, X_cat]).astype(np.float32)

X_train_tab = make_tab(df_train)
X_val_tab = make_tab(df_val)
X_test_tab = make_tab(df_test)

# StandardScaler para numT@ricos (mantiene OHE tal cual)
num_scaler = StandardScaler().fit(df_train[num_base].to_numpy(dtype=np.float32))
def scale_tab(X_tab):
    n_num = len(num_base)
    Xn = X_tab.copy()
    Xn[:, :n_num] = num_scaler.transform(Xn[:, :n_num])
    return Xn

X_train_tab = scale_tab(X_train_tab)
X_val_tab = scale_tab(X_val_tab)
X_test_tab = scale_tab(X_test_tab)

# ===== 8) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 9) Modelo LSTM (128->64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()
```

```
# ===== 10) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 11) Evaluaci3n y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va);      pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACI3N ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 12) Gr1ficas b1sicas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de p3rdida"); plt.xlabel("T3poca"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicci3n")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("T3ndice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 13) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_trm_precio_net0.keras"))
import joblib
joblib.dump(ohe, os.path.join(SAVE_DIR, "ohe_categorias.joblib"))
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 14) Notas de tuning r3pido =====
# - Ajusta LOOKBACK (p.ej. 28, 60, 90) seg3n autocorrelaci3n/estacionalidad.
# - Si hay alta cardinalidad en categoricas: baja TOP_K o filtra categor3as muy raras.
# - Sobreajuste: aumenta dropout/recurrent_dropout o L2; reduce unidades.
# - Infraajuste: aumenta unidades/capas o LOOKBACK.

# ===== Gr1fico final: Real vs LSTM con fechas alineadas =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas conforme al tama1o de cada bloque de predicci3n
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos de longitudes
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"

# 4) Concatenar todo para una sola l3nea LSTM
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_net0"].to_numpy()

# 6) Plot (rango din1mico)
min_y, max_y = real_full.min(), real_full.max()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio neto (COP) B3 Real vs LSTM ({year_min}B3{year_max})")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario (cortes + m3tricas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_net0"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_net0"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_net0"], label="TEST (real)", alpha=0.8)
ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")
ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validaci3n", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title("LSTM B3 precio_net0 (COP) B3 Real vs Predicho (con TRM)")
ax.set_ylabel("Precio neto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")
```



```
y_true_te = df_test["precio_neto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:-m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_neto"].values, y_true_te, pred_te_u, m=1)

txt = f"TEST ㄸ' MAE={mae:,.1f} | RMSE={rmse:,.1f} | sMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()
```

```
# --- Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title("Paridad en Test (con TRM)"); ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()
```

```
# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("TKndice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()
```

```
# =====
#@title LSTM + TRM + ACCIONES: BATES_50KG_TRM+ACCIONES.xlsx
# Vars: Proveedor/Centro suministrador, Textobreve, Material,
#       Cantidaddepedido, Precioneto, Fechadocumento, TRM, Klabin, Mondi, Smurfit
# Pipeline: rarefaciTiin top-K, escalado numTOrico, ventanas deslizantes,
#           LSTM (128ㄸ'64) + LayerNorm + regularizaciTiin + callbacks.
# =====
```

```
import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
SEED = 42
random.seed(SEED); np.random.seed(SEED)
```

```
import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization
```

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

```
# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")
```

```
PATH = "/content/drive/MyDrive/BATES_50KG_TRM+ACCIONES.xlsx"
df = pd.read_excel(PATH)
```

```
# ===== 2) Normalizar nombres y mapear a canTinicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"^\w\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]
```

```
rename_map = {
    "proveedor/centro_suministrador": "proveedor",
    "proveedor_centro_suministrador": "proveedor",
    "textobreve": "texto_breve",
    "cantidaddepedido": "cantidad_pedido",
    "cantidad_depedido": "cantidad_pedido",
    "precioneto": "precio_neto",
    "fecha_documento": "fecha_documento",
    "fechadocumento": "fecha_documento",
    "material": "material",
    "trm": "trm",
    "klabin": "klabin",
    "mondi": "mondi",
    "smurfit": "smurfit",
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})
```

```
# ===== 3) Columnas requeridas (incluye TRM y acciones) =====
required = [
    "proveedor","texto_breve","material",
    "cantidad_pedido","precio_neto","fecha_documento",
    "trm","klabin","mondi","smurfit"
]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"
```

```
df = df[required].copy()
```

```
# ===== 4) Tipos, fecha, orden temporal y seTiales calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
```

```
for c in ["cantidad_pedido","precio_neto","trm","klabin","mondi","smurfit"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")
```

```
df = df.dropna(subset=["fecha_documento","precio_neto"]).sort_values("fecha_documento").reset_index(drop=True)
```

```
# Relleno simple de exTigenas (si hay huecos)
for c in ["trm","klabin","mondi","smurfit"]:
    if df[c].isna().any():
        df[c] = df[c].ffill().bfill()
```

```
# SeTiales internas
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
```

```
df["cos_dow"]    = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val    = df.iloc[i_train:i_val].copy()
df_test   = df.iloc[i_val:].copy()

TARGET = "precio_neto"
cat_cols = ["proveedor","texto_breve","material"]
# NumIrricas internas + exIrigenas
num_base = [
    "cantidad_pedido","trm","klabin","mondi","smurfit",
    "sin_month","cos_month","sin_dow","cos_dow"
]

# ===== 6) RarefacciIn top-K en categIricas =====
TOP_K = 50
def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")

top_sets = {c: topk_set(df_train[c], TOP_K) for c in cat_cols}
for c in cat_cols:
    df_train[c] = to_topk(df_train[c], top_sets[c])
    df_val[c]   = to_topk(df_val[c],   top_sets[c])
    df_test[c]  = to_topk(df_test[c],  top_sets[c])

# ===== 7) Escalado (y con MinMax; X num con StandardScaler; OHE robusto) =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val    = y_scaler.transform(df_val[[TARGET]])
y_test   = y_scaler.transform(df_test[[TARGET]])

ohe = OneHotEncoder(handle_unknown="ignore")
ohe.fit(df_train[cat_cols])

def to_dense(X):
    return X.toarray() if hasattr(X, "toarray") else X

def make_tab(df_part):
    X_num = df_part[num_base].to_numpy(dtype=np.float32)
    X_cat = to_dense(ohe.transform(df_part[cat_cols]))
    return np.hstack([X_num, X_cat]).astype(np.float32)

X_train_tab = make_tab(df_train)
X_val_tab   = make_tab(df_val)
X_test_tab  = make_tab(df_test)

num_scaler = StandardScaler().fit(df_train[num_base].to_numpy(dtype=np.float32))
def scale_tab(X_tab):
    n_num = len(num_base)
    Xn = X_tab.copy()
    Xn[:, :n_num] = num_scaler.transform(Xn[:, :n_num])
    return Xn

X_train_tab = scale_tab(X_train_tab)
X_val_tab   = scale_tab(X_val_tab)
X_test_tab  = scale_tab(X_test_tab)

# ===== 8) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 9) Modelo LSTM (128Bt'64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x   = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x   = LayerNormalization()(x)
x   = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x   = LayerNormalization()(x)
x   = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x   = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 10) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 11) EvaluaciIn y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()

pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
```

```
        "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACION ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 12) Gráfico de rMSE =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de pérdida"); plt.xlabel("época"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicción")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("Índice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 13) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_trm_acc_precio_netto.keras"))
import joblib
joblib.dump(ohe, os.path.join(SAVE_DIR, "ohe_categorias.joblib"))
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 14) Gráfico final con fechas alineadas + panel complementario =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas por bloque
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos
assert len(fechas_tr) == len(pred_tr_u)
assert len(fechas_va) == len(pred_va_u)
assert len(fechas_te) == len(pred_te_u)

# 4) Concatenación
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa y plot principal
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_netto"].to_numpy()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio neto (COP) vs Real vs LSTM con TRM+Acciones ({year_min}-{year_max})")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario (cortes + métricas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_netto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_netto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_netto"], label="TEST (real)", alpha=0.8)

ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validación", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title("LSTM vs precio neto (COP) vs Real vs Predicho (TRM + Klabin + Mondy + Smurfit)")
ax.set_ylabel("Precio neto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

y_true_te = df_test["precio_netto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))

def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:-m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_netto"].values, y_true_te, pred_te_u, m=1)

txt = f"TEST vs MAE={mae:.1f} | RMSE={rmse:.1f} | sMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()

# --- Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title("Paridad en Test (TRM+Acciones)"); ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("Índice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

# =====
```

```
#@title LSTM: BATES_50KG_TRMSMURFIT.xlsx
# Vars: Fechadocumento, Precioneto (target), TRM, Smurfit
# Pipeline: renombre columnas, split temporal, seTtales calendario,
#           escalado numT@rico, ventanas deslizantes, LSTM (128n†'64),
#           callbacks, mT@tricas y grTŶficos con fechas alineadas.
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KG_TRMSMURFIT.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y renombrar a canPinicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"^[^w\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "precioneto": "precio_netto",
    "trm": "trm",
    "smurfit": "smurfit",
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) VerificaciTn de columnas =====
required = ["fecha_documento", "precio_netto", "trm", "smurfit"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"
df = df[required].copy()

# ===== 4) Tipos, orden temporal y seTtales calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
for c in ["precio_netto", "trm", "smurfit"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

# Orden y limpieza mT-nima
df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# Relleno simple de exTigenas si hay huecos
for c in ["trm", "smurfit"]:
    if df[c].isna().any():
        df[c] = df[c].ffill().bfill()

# SeTtales internas (no exTigenas externas)
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_netto"
num_feats = ["trm", "smurfit", "sin_month", "cos_month", "sin_dow", "cos_dow"]

# ===== 6) Escalado =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

X_train_tab = df_train[num_feats].to_numpy(dtype=np.float32)
X_val_tab = df_val[num_feats].to_numpy(dtype=np.float32)
X_test_tab = df_test[num_feats].to_numpy(dtype=np.float32)

num_scaler = StandardScaler().fit(X_train_tab)
X_train_tab = num_scaler.transform(X_train_tab)
X_val_tab = num_scaler.transform(X_val_tab)
X_test_tab = num_scaler.transform(X_test_tab)

# ===== 7) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 8) Modelo LSTM (128n†'64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
```

```
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 9) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 10) Evaluaci3n y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACI3N ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 11) Gr3ficas b3sicas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de p3rdida"); plt.xlabel("T3poca"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicci3n")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("T3ndice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 12) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_trm_smurfit_precio_net0.keras"))
import joblib
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 13) Gr3fico final con fechas alineadas + panel complementario =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas por bloque
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"

# 4) Concatenaci3n de todo el recorrido LSTM
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa y plot principal
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_net0"].to_numpy()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio neto (COP) b3 Real vs LSTM (TRM + Smurfit) {year_min}b3{year_max}")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario (cortes + m3tricas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_net0"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_net0"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_net0"], label="TEST (real)", alpha=0.8)

ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validaci3n", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")
```



```
ax.set_title("LSTM B· precio_netto (COP) vs Real vs Predicho (TRM + Smurfit)")
ax.set_ylabel("Precio netto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

y_true_te = df_test["precio_netto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:-m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_netto"].values, y_true_te, pred_te_u, m=1)
txt = f"TEST s+ MAE={mae:,.1f} | RMSE={rmse:,.1f} | sMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()

# --- Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title("Paridad en Test (TRM + Smurfit)")
ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("Índice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

# =====
#@title Runner de comparaciIn LSTM: BASE vs TRM vs TRM+ACCIONES vs TRM+SMURFIT
# - Mismo pipeline (split temporal, ventanas, escalado, LSTM 128s+64, callbacks)
# - Métricas: MAE, RMSE, sMAPE, MASE (eval. en TEST)
# - Tabla comparativa ordenada por MASE y gráfico de barras
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ----- Montar Drive -----
from google.colab import drive
drive.mount("/content/drive")

# ----- ConfiguraciIn de variantes -----
VARIANTS = [
    {
        "name": "BASE",
        "path": "/content/drive/MyDrive/BATES_50KG.xlsx",
        # columnas esperadas (pueden venir con variaciones)
        "expect": ["proveedor/centro suministrador", "textobreve", "material",
                  "cantidaddepedido", "precioneto", "fechadocumento"]
    },
    {
        "name": "TRM",
        "path": "/content/drive/MyDrive/BATES_50KG_TRM.xlsx",
        "expect": ["proveedor/centro suministrador", "textobreve", "material",
                  "cantidaddepedido", "precioneto", "fechadocumento", "trm"]
    },
    {
        "name": "TRM+ACCIONES",
        "path": "/content/drive/MyDrive/BATES_50KG_TRM+ACCIONES.xlsx",
        "expect": ["proveedor/centro suministrador", "textobreve", "material",
                  "cantidaddepedido", "precioneto", "fechadocumento", "trm", "klabin", "mondi", "smurfit"]
    },
    {
        "name": "TRM+SMURFIT",
        "path": "/content/drive/MyDrive/BATES_50KG_TRMSMURFIT.xlsx",
        "expect": ["fechadocumento", "precioneto", "trm", "smurfit"]
    },
],

# ----- Utilidades comunes -----
def slugify(s):
    s = str(s).strip()
    s = re.sub(r"^[^w\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

def smape(a, f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr)
    scale=np.mean(np.abs(ytr[m:]-ytr[:-m]))
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale and not np.isnan(scale) else 1)

def make_model(input_timesteps, input_features):
    inp = Input(shape=(input_timesteps, input_features))
    x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
    x = LayerNormalization()(x)
    x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
    x = LayerNormalization()(x)
    x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
    x = Dropout(0.2)(x)
    out = Dense(1, activation="linear")(x)
```

```
model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
return model

def detect_schema(df):
    """Devuelve diccionario con columnas canFinicas disponibles."""
    cmap = {
        "proveedor/centro_suministrador": "proveedor",
        "proveedor_centro_suministrador": "proveedor",
        "proveedorcentro_suministrador": "proveedor",
        "proveedorcentro_suministrador_": "proveedor",
        "textobreve": "texto_breve",
        "material": "material",
        "cantidaddepedido": "cantidad_pedido",
        "cantidad_depedido": "cantidad_pedido",
        "precioneo": "precio_net",
        "precio_net": "precio_net",
        "fechadocumento": "fecha_documento",
        "fecha_documento": "fecha_documento",
        "trm": "trm",
        "klabin": "klabin",
        "mondi": "mondi",
        "smurfit": "smurfit",
    }
    cols = {slugify(c): c for c in df.columns}
    canon = {}
    for raw, can in cmap.items():
        sr = slugify(raw)
        if sr in cols:
            canon[can] = cols[sr]
    return canon

def load_and_prepare(path):
    df = pd.read_excel(path)
    df.columns = [slugify(c) for c in df.columns]
    # mapear a canFinicos
    rename_map = {
        "proveedor/centro_suministrador": "proveedor",
        "proveedor_centro_suministrador": "proveedor",
        "textobreve": "texto_breve",
        "material": "material",
        "cantidaddepedido": "cantidad_pedido",
        "cantidad_depedido": "cantidad_pedido",
        "precioneo": "precio_net",
        "fecha_documento": "fecha_documento",
        "fechadocumento": "fecha_documento",
        "trm": "trm",
        "klabin": "klabin",
        "mondi": "mondi",
        "smurfit": "smurfit",
    }
    df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

    # columnas disponibles
    avail = set(df.columns)

    # fechas + target
    assert "fecha_documento" in avail, "Falta columna de fecha"
    assert "precio_net" in avail, "Falta columna objetivo 'precio_net' (renombra desde Precioneto)"

    # cast tipos
    df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
    for c in ["precio_net", "cantidad_pedido", "trm", "klabin", "mondi", "smurfit"]:
        if c in avail:
            df[c] = pd.to_numeric(df[c], errors="coerce")

    # ordenar y limpieza mT-nima
    df = df.dropna(subset=["fecha_documento", "precio_net"]).sort_values("fecha_documento").reset_index(drop=True)

    # rellenar exFigenas si hay huecos
    for c in ["trm", "klabin", "mondi", "smurfit", "cantidad_pedido"]:
        if c in avail and df[c].isna().any():
            df[c] = df[c].ffill().bfill()

    # seTtales calendario
    df["dow"] = df["fecha_documento"].dt.dayofweek
    df["month"] = df["fecha_documento"].dt.month
    df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
    df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
    df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
    df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

    # columnas categFiricas si existen
    cat_cols = [c for c in ["proveedor", "texto_breve", "material"] if c in avail]
    # numT@ricas base (armar segTen existencia)
    num_cols = [c for c in ["cantidad_pedido", "trm", "klabin", "mondi", "smurfit"] if c in avail]
    num_cols += ["sin_month", "cos_month", "sin_dow", "cos_dow"]

    return df, cat_cols, num_cols

def make_windows(tab_X, y, lookback=60, horizon=1):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

def run_variant(name, path, lookback=60, epochs=200, batch_size=64, top_k=50, verbose_fit=0):
    print(f"\n===== {name} =====")
    df, cat_cols, num_cols = load_and_prepare(path)

    # split temporal
    n = len(df); i_train = int(n*0.70); i_val = int(n*0.85)
    df_train, df_val, df_test = df.iloc[:i_train].copy(), df.iloc[i_train:i_val].copy(), df.iloc[i_val:].copy()

    # target scalers
    y_scaler = MinMaxScaler()
    y_train = y_scaler.fit_transform(df_train[["precio_net"]])
    y_val = y_scaler.transform(df_val[["precio_net"]])
    y_test = y_scaler.transform(df_test[["precio_net"]])

    # rarefacciFin categFirica (si aplica)
    if cat_cols:
        TOP_K = top_k
        def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
        def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")
        top_sets = {c: topk_set(df_train[c], TOP_K) for c in cat_cols}
        for c in cat_cols:
            df_train[c] = to_topk(df_train[c], top_sets[c])
            df_val[c] = to_topk(df_val[c], top_sets[c])
            df_test[c] = to_topk(df_test[c], top_sets[c])
```

```

        ohe = OneHotEncoder(handle_unknown="ignore")
        ohe.fit(df_train[cat_cols])
        def _to_dense(X): return X.toarray() if hasattr(X,"toarray") else X
    else:
        ohe = None

# tabulares
X_train_num = df_train[num_cols].to_numpy(dtype=np.float32)
X_val_num   = df_val[num_cols].to_numpy(dtype=np.float32)
X_test_num  = df_test[num_cols].to_numpy(dtype=np.float32)

num_scaler = StandardScaler().fit(X_train_num)
X_train_num = num_scaler.transform(X_train_num)
X_val_num   = num_scaler.transform(X_val_num)
X_test_num  = num_scaler.transform(X_test_num)

if ohe:
    X_train_cat = _to_dense(ohe.transform(df_train[cat_cols]))
    X_val_cat   = _to_dense(ohe.transform(df_val[cat_cols]))
    X_test_cat  = _to_dense(ohe.transform(df_test[cat_cols]))
    X_train_tab = np.hstack([X_train_num, X_train_cat]).astype(np.float32)
    X_val_tab   = np.hstack([X_val_num,   X_val_cat  ]).astype(np.float32)
    X_test_tab  = np.hstack([X_test_num,  X_test_cat ]).astype(np.float32)
else:
    X_train_tab, X_val_tab, X_test_tab = X_train_num, X_val_num, X_test_num

# ventanas
Xtr, ytr = make_windows(X_train_tab, y_train, lookback, 1)
Xva, yva = make_windows(np.vstack([X_train_tab[-lookback+1:], X_val_tab]),
                        np.vstack([y_train[-lookback+1:], y_val]), lookback, 1)
Xte, yte = make_windows(np.vstack([X_val_tab[-lookback+1:], X_test_tab]),
                        np.vstack([y_val[-lookback+1:], y_test]), lookback, 1)

# modelo
model = make_model(lookback, Xtr.shape[-1])
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)
hist = model.fit(Xtr, ytr, validation_data=(Xva, yva),
                epochs=epochs, batch_size=batch_size, callbacks=[es, rlr],
                verbose=verbose_fit)

# predicciones y mÉtricas (TEST)
invy = lambda z: y_scaler.inverse_transform(z.reshape(-1,1)).ravel()
yte_hat = model.predict(Xte, verbose=0).ravel()
yte_hat_u = invy(yte_hat)
yte_u = invy(yte.ravel())

mae = mean_absolute_error(yte_u, yte_hat_u)
rmse = math.sqrt(mean_squared_error(yte_u, yte_hat_u))
smp = smape(yte_u, yte_hat_u)
ms = mase(df_train["precio_neto"].values, yte_u, yte_hat_u, m=1)

# fechas para gráfico de test
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy() # por la construcción de ventanas
out = {
    "name": name, "mae": mae, "rmse": rmse, "smape": smp, "mase": ms,
    "fechas_te": fechas_te, "y_true_te": yte_u, "y_pred_te": yte_hat_u,
    "history": hist.history, "model": model
}
print(f'{name} B+ MAE={mae:.2f} | RMSE={rmse:.2f} | sMAPE={smp:.2f}% | MASE={ms:.3f}')
return out

# ----- Ejecutar todas las variantes disponibles -----
results = []
for v in VARIANTS:
    if os.path.exists(v["path"]):
        try:
            r = run_variant(v["name"], v["path"], lookback=60, epochs=200, batch_size=64, verbose_fit=1)
            results.append(r)
        except Exception as e:
            print(f"[WARN] Falló {v['name']}: {e}")
    else:
        print(f"[SKIP] No existe: {v['path']}")

# ----- Tabla comparativa -----
if not results:
    raise SystemExit("No se pudo entrenar ninguna variante. Revisa las rutas.")

df_metrics = pd.DataFrame([
    {"VARIANTE": r["name"],
     "MAE": r["mae"],
     "RMSE": r["rmse"],
     "sMAPE_%": r["smape"],
     "MASE": r["mase"],
    } for r in results])

# Ordena por MASE (menor es mejor). Cambia a 'sMAPE_%' si prefieres.
df_metrics = df_metrics.sort_values(by="MASE", ascending=True).reset_index(drop=True)
print("\n=== COMPARATIVA (ordenado por MASE) ===")
print(df_metrics.to_string(index=False))

# ----- Gráfico de barras (MASE y sMAPE) -----
plt.figure(figsize=(8,4))
plt.bar(df_metrics["VARIANTE"], df_metrics["MASE"])
plt.title("Comparación MASE (B+ mejor)")
plt.ylabel("MASE")
plt.grid(True, axis="y", alpha=0.3)
plt.show()

plt.figure(figsize=(8,4))
plt.bar(df_metrics["VARIANTE"], df_metrics["sMAPE_%"])
plt.title("Comparación sMAPE % (B+ mejor)")
plt.ylabel("sMAPE (%)")
plt.grid(True, axis="y", alpha=0.3)
plt.show()

# ----- (Opcional) Paridad de TEST por variante -----
cols = len(results)
fig, axes = plt.subplots(1, cols, figsize=(5*cols, 4), squeeze=False)
for i, r in enumerate(results):
    ax = axes[0, i]
    y_true = r["y_true_te"]; y_pred = r["y_pred_te"]
    lims = [min(y_true.min(), y_pred.min()), max(y_true.max(), y_pred.max())]
    ax.scatter(y_true, y_pred, s=10, alpha=0.7)
    ax.plot(lims, lims, 'k--', lw=1)
    ax.set_xlim(lims); ax.set_ylim(lims)
    ax.set_title(f"Paridad TEST B· {r['name']}")
    ax.set_xlabel("Real"); ax.set_ylabel("Predicho")
    ax.grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

# ----- (Opcional) Serie TEST real vs predicho (cada variante) -----
for r in results:
```



```
plt.figure(figsize=(11,4))
plt.plot(r["fechas_te"], r["y_true_te"], label="Real")
plt.plot(r["fechas_te"], r["y_pred_te"], label="Predicho")
plt.title(f"TEST B: Real vs Predicho B") {r['name']}")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.legend(); plt.grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

# ----- Guardar mÉtricas a CSV -----
out_csv = "/content/lstm_comparativa_metrics.csv"
df_metrics.to_csv(out_csv, index=False)
print(f"\nMÉtricas guardadas en: {out_csv}")

# =====
#@title LSTM: BATES_50KG_TRMSMURFIT-47.xlsx
# Vars: Fechadocumento, Precioneto (target), TRM, Smurfit
# Extra: opcíon para desplazar Smurfit -47 observaciones (lead) si se desea.
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 0) Config rÉpida =====
DO_SHIFT_SMURFIT = False # <-- pon True si quieres usar Smurfit con -47
SHIFT_STEPS = 47 # nEmero de observaciones a desplazar

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KG_TRMSMURFIT-47.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y renombrar a canÓnicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"^\w\s/", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "precioneto": "precio_netto",
    "trm": "trm",
    "smurfit": "smurfit",
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) VerificaciÓn de columnas =====
required = ["fecha_documento", "precio_netto", "trm", "smurfit"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"
df = df[required].copy()

# ===== 4) Tipos, orden temporal y seÓtales calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
for c in ["precio_netto", "trm", "smurfit"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

# Orden y limpieza mÉnima
df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# (Opcional) Desplazar Smurfit -47 observaciones (lead); Tetil si Smurfit antecede al precio
# Nota: shift(-47) mueve valores futuros hacia el presente (riesgo de fuga si no se justifica).
if DO_SHIFT_SMURFIT:
    df["smurfit_shifted"] = df["smurfit"].shift(-SHIFT_STEPS)
    df["smurfit_shifted"] = df["smurfit_shifted"].ffill().bfill()
    smurfit_feature = "smurfit_shifted"
else:
    smurfit_feature = "smurfit"

# Relleno simple de exÓgenas si hay huecos
for c in ["trm", smurfit_feature]:
    if df[c].isna().any():
        df[c] = df[c].ffill().bfill()

# SeÓtales internas (no exÓgenas externas)
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_netto"
num_feats = ["trm", smurfit_feature, "sin_month", "cos_month", "sin_dow", "cos_dow"]

# ===== 6) Escalado =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

X_train_tab = df_train[num_feats].to_numpy(dtype=np.float32)
X_val_tab = df_val[num_feats].to_numpy(dtype=np.float32)
X_test_tab = df_test[num_feats].to_numpy(dtype=np.float32)

num_scaler = StandardScaler().fit(X_train_tab)
```

```
X_train_tab = num_scaler.transform(X_train_tab)
X_val_tab = num_scaler.transform(X_val_tab)
X_test_tab = num_scaler.transform(X_test_tab)

# ===== 7) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 8) Modelo LSTM (128B+64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 9) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 10) Evaluaci3n y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACI3N ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 11) Gr3ficas b3sicas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de p3rdida"); plt.xlabel("T3poca"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicci3n")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("T3ndice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 12) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_trm_smt47_precio_net0.keras"))
import joblib
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 13) Gr3fico final con fechas alineadas + panel complementario =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas por bloque
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"

# 4) Concatenaci3n de todo el recorrido LSTM
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa y plot principal
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_net0"].to_numpy()
```

```
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
titulo_extra = " (Smurfit -47)" if DO_SHIFT_SMURFIT else ""
plt.title(f"Precio neto (COP) vs Real vs LSTM (TRM + Smurfit){titulo_extra} {year_min} vs {year_max}")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario (cortes + métricas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_neto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_neto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_neto"], label="TEST (real)", alpha=0.8)

ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validación", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title("LSTM vs precio neto (COP) vs Real vs Predicho (TRM + Smurfit)")
ax.set_ylabel("Precio neto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

y_true_te = df_test["precio_neto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:~m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_neto"].values, y_true_te, pred_te_u, m=1)
txt = f"TEST vs MAE={mae:.1f} | RMSE={rmse:.1f} | sMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()

# --- Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title("Paridad en Test (TRM + Smurfit)")
ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("Índice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

# =====
#@title CCF y verificación SARIMAX (Colab)
# Autor: ChatGPT (colab-ready)
# Descripción:
# - Lee: /content/drive/MyDrive/BATES_50KG_TRMSMURFIT.xlsx
# - Indexa por Fechadocumento y re-muestrea a FREQ ('W-MON' o 'D')
# - Convierte Precioneto y Smurfit a log-retornos (o diferencias si hay no-positivos)
# - Calcula CCF para lags en [-LAG_MAX, +LAG_MAX] con convención:
#   k > 0 => Smurfit antecede a Precioneto (lead)
# - Selecciona k* que maximiza |CCF| y grafica CCF con bandas ±1.96/sqrt(N)
# - Verificación: SARIMAX sobre retornos con Smurfit desplazada en k ∈ {k*, 0, k*-1, k*+1}
#   Muestra MAE / RMSE / sMAPE y concluye el mejor delay
# =====

# ===== Dependencias =====
!pip -q install pandas==2.2.2 numpy==1.26.4 matplotlib==3.8.4 statsmodels==0.14.2 openpyxl==3.1.5 scikit-learn==1.5.1

import warnings, math
warnings.filterwarnings("ignore")

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from google.colab import drive
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== Parámetros del usuario =====
PATH = "/content/drive/MyDrive/BATES_50KG_TRMSMURFIT.xlsx"

# Frecuencia de trabajo: "W-MON" (semanal, lunes) o "D" (diaria)
FREQ = "W-MON" # cámbialo a "D" si prefieres diario
LAG_MAX = 90 # semanas si FREQ="W-MON", días si FREQ="D"

# Resample rule: para precios suele ser "last"; si prefieres "mean", cámbialo
RESAMPLE_HOW = "last" # "last" o "mean"

# Split temporal para verificación SARIMAX
TEST_SIZE_FRAC = 0.2 # 20% final para test

# Orden SARIMAX simple (sobre retornos ya estacionarios). Puedes ajustar si deseas.
SARIMAX_ORDER = (1, 0, 1)

# ===== Utilidades =====
def to_datetime_index(df, date_col="Fechadocumento"):
    df = df.copy()
    df[date_col] = pd.to_datetime(df[date_col], errors="coerce", utc=False)
    df = df.dropna(subset=[date_col])
    df = df.sort_values(date_col).set_index(date_col)
    return df

def resample_df(df, freq="W-MON", how="last"):
    if how == "last":
        return df.resample(freq).last()
    elif how == "mean":
        return df.resample(freq).mean()
    else:
```

```
        raise ValueError("RESAMPLE_HOW debe ser 'last' o 'mean'.")

def log_returns_or_diff(s: pd.Series, min_positive=1e-12):
    """Devuelve log-retornos si todos los valores son >0 y no hay ceros; si no, diferencias simples.
    Devuelve (serie_ret, metodo_str)."""
    s = s.astype(float)
    if (s > 0).all() and (s.min() > 0):
        # log-retorno
        lr = np.log(s) - np.log(s.shift(1))
        return lr, "log-return"
    else:
        # diferencia
        diff = s - s.shift(1)
        return diff, "diff"

def smape(y_true, y_pred, eps=1e-8):
    num = np.abs(y_pred - y_true)
    den = (np.abs(y_true) + np.abs(y_pred)).clip(min=eps)
    return 100.0 * np.mean(2.0 * num / den)

def compute_ccf(smurf_ret, prec_ret, lag_max=90):
    """CCF por definici n: corr(S_t, P_{t+k}) para k in [-lag_max, +lag_max].
    k>0 => S antecede a P.
    """
    smurf = smurf_ret.copy()
    prec = prec_ret.copy()

    # alineaci n base
    out_lags, out_ccf = [], []
    for k in range(-lag_max, lag_max + 1):
        # P_{t+k} => prec.shift(k)
        s_k = smurf
        p_k = prec.shift(k)
        pair = pd.concat([s_k, p_k], axis=1).dropna()
        if len(pair) < 5:
            out_lags.append(k)
            out_ccf.append(np.nan)
            continue
        c = pair.iloc[:,0].corr(pair.iloc[:,1])
        out_lags.append(k)
        out_ccf.append(c)
    return np.array(out_lags), np.array(out_ccf)

def significance_bounds(n):
    # Aproximaci n de bandas de significancia para CCF: B 1.96/sqrt(N)
    if n <= 1:
        return np.nan, np.nan
    bound = 1.96 / math.sqrt(n)
    return -bound, bound

def shift_exog_for_k(smurf_ret, k):
    """Devuelve exog alineada para explicar endog_t con Smurfit_{t-k}? No:
    Por nuestra convenci n, en la CCF corr(S_t, P_{t+k}) => si queremos modelar P_t,
    necesitamos S_{t-k} (porque S antecede k pasos: S_{t-k} -> P_t).
    Por tanto, exog = Smurfit.shift(k_neg) con k_neg = -k.
    """
    return smurf_ret.shift(-k)

def train_test_split_by_time(series: pd.Series, test_frac=0.2):
    n = len(series.dropna())
    n_test = max(1, int(round(n * test_frac)))
    end = series.dropna().index[-1]
    start = series.dropna().index[0]
    ser = series.loc[start:end]
    split_idx = len(ser) - n_test
    return ser.iloc[:split_idx], ser.iloc[split_idx:]

def align_endog_exog(endog: pd.Series, exog: pd.Series):
    df = pd.concat([endog.rename("y"), exog.rename("x")], axis=1).dropna()
    return df["y"], df["x"]

def eval_sarimax_for_k(endog_ret, smurf_ret, k, order=(1,0,1)):
    """Ajusta SARIMAX(endog_ret, exog=Smurfit_shifted(-k)) y eval a en test."""
    exog_k = shift_exog_for_k(smurf_ret, k)
    y_al, x_al = align_endog_exog(endog_ret, exog_k)
    if len(y_al) < 20:
        return None # muy pocos datos

    y_tr, y_te = train_test_split_by_time(y_al, test_frac=TEST_SIZE_FRAC)
    x_tr = x_al.loc[y_tr.index]
    x_te = x_al.loc[y_te.index]

    # Asegurar longitudes iguales
    y_tr, x_tr = y_tr.align(x_tr, join="inner")
    y_te, x_te = y_te.align(x_te, join="inner")

    if len(y_tr) < 10 or len(y_te) < 5:
        return None

    # Entrenar SARIMAX
    try:
        model = SARIMAX(endog=y_tr, exog=x_tr, order=order, enforce_stationarity=False, enforce_invertibility=False)
        res = model.fit(dispatch=False)
        # Pron stico para el tramo de test
        pred = res.predict(start=y_te.index[0], end=y_te.index[-1], exog=x_te)
        mae = mean_absolute_error(y_te, pred)
        rmse = math.sqrt(mean_squared_error(y_te, pred))
        smp = smape(y_te.values, pred.values)
        return {"k": k, "MAE": mae, "RMSE": rmse, "sMAPE( )": smp, "n_train": len(y_tr), "n_test": len(y_te)}
    except Exception as e:
        return None

# ===== 1) Montar Drive y cargar =====
drive.mount("/content/drive")

raw = pd.read_excel(PATH)
assert {"Fechadocumento", "Precioneto", "TRM", "Smurfit"}.issubset(raw.columns), \
    f"El archivo debe contener columnas: Fechadocumento, Precioneto, TRM, Smurfit. Tiene: {list(raw.columns)}"

df = to_datetime_index(raw, "Fechadocumento")
df = df[["Precioneto", "TRM", "Smurfit"]].sort_index()

# ===== 2) Re-muestreo y alineaci n =====
df_rs = resample_df(df, freq=FREQ, how=RESAMPLE_HOW)
# Relleno prudente si quedan huecos entre periodos (opcional):
df_rs = df_rs.ffill()

# ===== 3) Transformar a retornos/diferencias =====
prec_ret, prec_method = log_returns_or_diff(df_rs["Precioneto"])
smurf_ret, smurf_method = log_returns_or_diff(df_rs["Smurfit"])

# Limpiar primeras filas con NaN por el shift
ret = pd.concat([prec_ret.rename("prec_ret"), smurf_ret.rename("smurf_ret")], axis=1).dropna()
```

```
N_eff = len(ret)

print(f"Frecuencia usada: {FREQ} | Observaciones efectivas (retornos): {N_eff}")
print(f"Transformaci3n Precioneto: {prec_method} | Transformaci3n Smurfit: {smur_method}")

# ===== 4) CCF: lags [-LAG_MAX, +LAG_MAX] =====
lags, ccf_vals = compute_ccf(ret["smur_ret"], ret["prec_ret"], lag_max=LAG_MAX)

# Encontrar k* (m3ximo |CCF|), ignorando NaN
valid = ~np.isnan(ccf_vals)
lags_v = lags[valid]
ccf_v = ccf_vals[valid]
idx_max = np.argmax(np.abs(ccf_v))
k_star = int(lags_v[idx_max])
ccf_star = float(ccf_v[idx_max])
signo = "positivo (Smurfit antecede)" if k_star > 0 else ("negativo (Precioneto antecede)" if k_star < 0 else "cero (simult3neo)")

# Bandas de significancia
low_b, up_b = significance_bounds(N_eff)

print("\n=== RESULTADOS CCF ===")
print(f"k* = {k_star} | signo: {signo} | CCF(k*): {ccf_star:.4f}")
print(f"Bandas aprox. de significancia: [{low_b:.3f}, {up_b:.3f}]" )

# ===== 5) Gr3fico CCF con bandas =====
plt.figure(figsize=(12,5))
plt.axhline(0, linewidth=1)
if not np.isnan(low_b):
    plt.axhline(up_b, linestyle="--")
    plt.axhline(low_b, linestyle="--")
plt.bar(lags, ccf_vals, width=0.8, align="center")
plt.title(f"CCF Smurfit vs Precioneto B3 lags en {'semanas' if FREQ.startswith('W') else 'd3-as'} (k>0: Smurfit antecede)")
plt.xlabel(f"Lag ({'semanas' if FREQ.startswith('W') else 'd3-as'})")
plt.ylabel("CCF")
plt.tight_layout()
plt.show()

# ===== 6) Verificaci3n SARIMAX para k in {k*, 0, k*-1, k*+1} =====
candidatos = sorted(set([0, k_star, k_star-1, k_star+1]))
resultados = []

for k in candidatos:
    r = eval_sarimax_for_k(ret["prec_ret"], ret["smur_ret"], k, order=SARIMAX_ORDER)
    if r is not None:
        resultados.append(r)

# Mostrar tabla de resultados
if len(resultados) == 0:
    print("\nNo fue posible entrenar/evaluar SARIMAX con los datos disponibles (Bimuy pocos puntos?).")
else:
    df_res = pd.DataFrame(resultados).sort_values("sMAPE(%)")
    print("\n=== VERIFICACI3N SARIMAX (retornos) ===")
    print(f"Orden SARIMAX usado: {SARIMAX_ORDER} | Split Test: {int(TEST_SIZE_FRAC*100)}% final")
    print(df_res.to_string(index=False, formatters={"MAE": "{:.6f}".format, "RMSE": "{:.6f}".format, "sMAPE(%)": "{:.3f}".format}))

    # Conclusi3n r3pida
    best = df_res.iloc[0]
    concl = (
        f"\nConclusi3n: el delay que mejor explica Precioneto (seg3n sMAPE) es k = {int(best['k'])} "
        f"({'Smurfit antecede' if int(best['k'])>0 else ('Precioneto antecede' if int(best['k'])<0 else 'simult3neo')}). "
        f"CCF(k*) = {ccf_star:.4f} en k* = {k_star}. "
        f"Robustez: comparamos contra k=0 y vecinos (k31). "
    )
    print(concl)

#@title Mejor lag positivo (Smurfit -> Precioneto)
mask_pos = (lags_v > 0)
if mask_pos.any():
    idx_pos = np.argmax(np.abs(ccf_v[mask_pos]))
    k_pos = int(lags_v[mask_pos][idx_pos])
    ccf_pos = float(ccf_v[mask_pos][idx_pos])
    print(f"Mejor lag con Smurfit antecediendo: k_pos = {k_pos}, CCF = {ccf_pos:.4f}")
else:
    print("No hay lags positivos v3lidos para evaluar (Smurfit -> Precioneto).")

# =====
#@title LSTM: BATES_50KGTRMSMURFIT+14.xlsx
# Vars esperadas: Fechadocumento, Precioneto (target), TRM, Smurfit
# Nota: Por el sufijo "+14", se usa Smurfit rezagado 14 pasos (smurfit_lag14)
# para evitar leakage. Ajusta SHIFT_SMURFIT si lo necesitas.
# Pipeline: renombre columnas, creaci3n lag, split temporal, se3tales calendario,
# escalado num3rico, ventanas deslizantes, LSTM (128B364), callbacks,
# m3tricas y gr3ficos con fechas alineadas.
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KGTRMSMURFIT+14.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y renombrar a can3nicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"[^w3s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "precioneto": "precio_net",
}
```



```
        "trm": "trm",
        "smurfit": "smurfit",
    }
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) Verificaci3n de columnas =====
required = ["fecha_documento", "precio_net", "trm", "smurfit"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"
df = df[required].copy()

# ===== 4) Tipos, orden temporal y series calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
for c in ["precio_net", "trm", "smurfit"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

# Orden y limpieza m3nima
df = df.dropna(subset=["fecha_documento", "precio_net"]).sort_values("fecha_documento").reset_index(drop=True)

# Relleno simple de ex3genas si hay huecos
for c in ["trm", "smurfit"]:
    if df[c].isna().any():
        df[c] = df[c].ffill().bfill()

# ===== 4b) Crear rezago de Smurfit por el sufijo "+14" =====
SHIFT_SMURFIT = 14 # cambia aqu3- si quieres otro rezago
df["smurfit_lag14"] = df["smurfit"].shift(SHIFT_SMURFIT)

# Series internas
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# El rezago introduce NaNs al inicio: recorta filas iniciales sin informaci3n completa
df = df.dropna(subset=["smurfit_lag14"]).reset_index(drop=True)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_net"
# Usamos TRM actual y Smurfit rezagado 14, m3s series calendario
num_feats = ["trm", "smurfit_lag14", "sin_month", "cos_month", "sin_dow", "cos_dow"]

# ===== 6) Escalado =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

X_train_tab = df_train[num_feats].to_numpy(dtype=np.float32)
X_val_tab = df_val[num_feats].to_numpy(dtype=np.float32)
X_test_tab = df_test[num_feats].to_numpy(dtype=np.float32)

num_scaler = StandardScaler().fit(X_train_tab)
X_train_tab = num_scaler.transform(X_train_tab)
X_val_tab = num_scaler.transform(X_val_tab)
X_test_tab = num_scaler.transform(X_test_tab)

# ===== 7) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 8) Modelo LSTM (128B+64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 9) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 10) Evaluaci3n y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
```

```
d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}

print("== VALIDACION ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 11) Gráficoas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de pérdida"); plt.xlabel("Época"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicción")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("Índice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 12) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_trm_smurfit_lag14_precio_netto.keras"))
import joblib
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 13) Gráfico final con fechas alineadas + panel complementario =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas por bloque
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"

# 4) Concatenación de todo el recorrido LSTM
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa y plot principal
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_netto"].to_numpy()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio neto (COP) vs Real vs LSTM (TRM + Smurfit_lag{SHIFT_SMURFIT}) {year_min} vs {year_max}")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario (cortes + métricas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_netto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_netto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_netto"], label="TEST (real)", alpha=0.8)

ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validación", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title(f"LSTM vs precio_netto (COP) vs Real vs Predicho (TRM + Smurfit_lag{SHIFT_SMURFIT})")
ax.set_ylabel("Precio neto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

y_true_te = df_test["precio_netto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:-m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_netto"].values, y_true_te, pred_te_u, m=1)
txt = f"TEST vs MAE={mae:.1f} | RMSE={rmse:.1f} | sMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()

# --- Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title(f"Paridad en Test (TRM + Smurfit_lag{SHIFT_SMURFIT})")
ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("Índice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
```

```
plt.tight_layout(); plt.show()

"""# ENTRENAMIENTO DE MODELO LSTM PARA TRES BASES DE DATOS DISTINTAS CON ADICIT N DE VARIABLES EXOGENAS PARA MEDIR MEJORAS Y AJUTES BASE DE DATOS: BASE -BASE+TRM Y BASE+TRM+TRM
# =====
#@title LSTM BASE (CON COVID): BATE_50KG_CONCOVID.xlsx
# Vars: Proveedor/Centro suministrador, Textobreve, Fechadocumento,
#       Material, Precioneto (target)
# Pipeline: renombrar columnas, split temporal, se ales calendario,
#           rarefacci n + OneHot, escalado, ventanas LSTM (128+64),
#           callbacks, m tricas y gr ficos alineados.
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATE_50KG_CONCOVID.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y renombrar a can nicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"[^\w\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "proveedor/centro_suministrador": "proveedor",
    "proveedor_centro_suministrador": "proveedor",
    "textobreve": "texto_breve",
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "material": "material",
    "precioneto": "precio_netto",
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) Verificaci n de columnas =====
required = ["proveedor", "texto_breve", "fecha_documento", "material", "precio_netto"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"
df = df[required].copy()

# ===== 4) Tipos, orden temporal y se ales calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
df["precio_netto"] = pd.to_numeric(df["precio_netto"], errors="coerce")

df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# Se ales internas (no ex genas externas)
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_netto"
cat_cols = ["proveedor", "texto_breve", "material"]
num_base = ["sin_month", "cos_month", "sin_dow", "cos_dow"] # solo se ales calendario

# ===== 6) Rarefacci n top-K en categoricas =====
TOP_K = 50
def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")

top_sets = {c: topk_set(df_train[c], TOP_K) for c in cat_cols}
for c in cat_cols:
    df_train[c] = to_topk(df_train[c], top_sets[c])
    df_val[c] = to_topk(df_val[c], top_sets[c])
    df_test[c] = to_topk(df_test[c], top_sets[c])

# ===== 7) Escalado (y con MinMax; X num con StandardScaler; OHE robusto) =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

ohe = OneHotEncoder(handle_unknown="ignore")
ohe.fit(df_train[cat_cols])

def _to_dense(X):
    return X.toarray() if hasattr(X, "toarray") else X

def make_tab(df_part):
    X_num = df_part[num_base].to_numpy(dtype=np.float32)
    X_cat = _to_dense(ohe.transform(df_part[cat_cols]))
    return np.hstack([X_num, X_cat]).astype(np.float32)

X_train_tab = make_tab(df_train)
X_val_tab = make_tab(df_val)
X_test_tab = make_tab(df_test)

num_scaler = StandardScaler().fit(df_train[num_base].to_numpy(dtype=np.float32))
def scale_tab(X_tab):
```



```
n_num = len(num_base)
Xn = X_tab.copy()
Xn[:, :n_num] = num_scaler.transform(Xn[:, :n_num])
return Xn

X_train_tab = scale_tab(X_train_tab)
X_val_tab = scale_tab(X_val_tab)
X_test_tab = scale_tab(X_test_tab)

# ===== 8) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 9) Modelo LSTM (128B+64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 10) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 11) Evaluaci3n y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACI3N ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 12) GrÁficas bÁsicas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de pÉrdida"); plt.xlabel("Época"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicci3n")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("Índice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 13) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_base_concovid_precio_net0.keras"))
import joblib
joblib.dump(ohe, os.path.join(SAVE_DIR, "ohe_categorias.joblib"))
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 14) GrÁficos finales con fechas alineadas =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas por bloque
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"

# 4) Concatenar todo el recorrido LSTM
```

```
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa y plot principal
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_neto"].to_numpy()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio neto (COP) vs Real vs LSTM (Con COVID) {year_min} vs {year_max}")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario (cortes + métricas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_neto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_neto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_neto"], label="TEST (real)", alpha=0.8)

ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validación", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title("LSTM vs precio neto (COP) vs Real vs Predicho (Con COVID)")
ax.set_ylabel("Precio neto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

y_true_te = df_test["precio_neto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:~m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_neto"].values, y_true_te, pred_te_u, m=1)
txt = f"TEST vs MAE={mae:.1f} | RMSE={rmse:.1f} | SMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()

# --- Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title("Paridad en Test (Con COVID)")
ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("Tiempo (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

# =====
#@title LSTM (CON COVID) + TRM: BATES_50KG_TRMCONCOVID.xlsx
# Vars: Proveedor/Centro suministrador, Textobreve, Fechadocumento,
#       Material, Precioneto (target), TRM
# Pipeline: rarefacción top-K, OHE robusto, escalado numérico,
#           ventanas deslizantes, LSTM (128+64) + LayerNorm, callbacks.
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KG_TRMCONCOVID.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y mapear a canónicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"[^\w\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/","_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "proveedor/centro_suministrador": "proveedor",
    "proveedor_centro_suministrador": "proveedor",
    "textobreve": "texto_breve",
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "material": "material",
    "precioneto": "precio_neto",
    "trm": "trm",
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})
```

```
# ===== 3) Filtrar columnas requeridas =====
required = ["proveedor","texto_breve","fecha_documento","material","precio_net","trm"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"
df = df[required].copy()

# ===== 4) Tipos, orden temporal y seTiales calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
for c in ["precio_net","trm"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

df = df.dropna(subset=["fecha_documento","precio_net"]).sort_values("fecha_documento").reset_index(drop=True)

# Relleno simple de TRM (si hay huecos)
if df["trm"].isna().any():
    df["trm"] = df["trm"].ffill().bfill()

# SeTiales internas (no exTigenas externas)
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_net"
cat_cols = ["proveedor","texto_breve","material"]
num_base = ["trm","sin_month","cos_month","sin_dow","cos_dow"] # numT@ricas internas + TRM

# ===== 6) Rarefacci@n top-K para categT@ricas =====
TOP_K = 50
def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")

top_sets = {c: topk_set(df_train[c], TOP_K) for c in cat_cols}
for c in cat_cols:
    df_train[c] = to_topk(df_train[c], top_sets[c])
    df_val[c] = to_topk(df_val[c], top_sets[c])
    df_test[c] = to_topk(df_test[c], top_sets[c])

# ===== 7) Escalado: y con MinMax; X num con StandardScaler; OHE robusto =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

# OHE solo con TRAIN (sin 'sparse=False' para evitar incompatibilidades)
ohe = OneHotEncoder(handle_unknown="ignore")
ohe.fit(df_train[cat_cols])

def _to_dense(X):
    return X.toarray() if hasattr(X, "toarray") else X

def make_tab(df_part):
    X_num = df_part[num_base].to_numpy(dtype=np.float32)
    X_cat = _to_dense(ohe.transform(df_part[cat_cols]))
    return np.hstack([X_num, X_cat]).astype(np.float32)

X_train_tab = make_tab(df_train)
X_val_tab = make_tab(df_val)
X_test_tab = make_tab(df_test)

# Estandarizar solo los numT@ricos (mantener OHE tal cual)
num_scaler = StandardScaler().fit(df_train[num_base].to_numpy(dtype=np.float32))
def scale_tab(X_tab):
    n_num = len(num_base)
    Xn = X_tab.copy()
    Xn[:, :n_num] = num_scaler.transform(Xn[:, :n_num])
    return Xn

X_train_tab = scale_tab(X_train_tab)
X_val_tab = scale_tab(X_val_tab)
X_test_tab = scale_tab(X_test_tab)

# ===== 8) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1 # puedes ajustar el LOOKBACK tras revisar ACF/PACF/estacionalidad

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 9) Modelo LSTM (128Bt+64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 10) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
```

```
epochs=200,
batch_size=64,
callbacks=[es, rlr],
verbose=1
)

# ===== 11) EvaluaciIn y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACIIn ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 12) GrIfficas bIfficas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de pOrdida"); plt.xlabel("T%poca"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="PredicciIn")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("Tndice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 13) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_trm_concovid_precio_netto.keras"))
import joblib
joblib.dump(ohe, os.path.join(SAVE_DIR, "ohe_categorias.joblib"))
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 14) GrIfficos finales con fechas alineadas =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas por bloque
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"

# 4) Concatenar todo el recorrido LSTM
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa y plot principal
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_netto"].to_numpy()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio neto (COP) bReal vs LSTM (Con COVID + TRM) {year_min}b{year_max}")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario (cortes + mIfficas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_netto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_netto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_netto"], label="TEST (real)", alpha=0.8)

ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio ValidaciIn", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title("LSTM B precio_netto (COP) bReal vs Predicho (Con COVID + TRM)")
ax.set_ylabel("Precio neto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

y_true_te = df_test["precio_netto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:-m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_netto"].values, y_true_te, pred_te_u, m=1)
txt = f"TEST b MAE={mae:.1f} | RMSE={rmse:.1f} | sMAPE={smape(y_true_te, pred_te_u)::2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()

# --- Paridad (Real vs Predicho) en TEST ---
```

```
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title("Paridad en Test (Con COVID + TRM)")
ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("Índice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

# =====
#@title LSTM (CON COVID) + TRM + ACCIONES
# Path: /content/drive/MyDrive/BATES_50KG_TRMACCIONESCONCOVID.xlsx
# Vars: Proveedor/Centro suministrador, Textobreve, Fechadocumento,
#       Material, Precioneto (target), TRM, Klabin, Mondi, Smurfit
# Pipeline: rarefaciIn top-K, OHE robusto, escalado numTOrico,
#           ventanas deslizantes, LSTM (128B+64) + LayerNorm, callbacks,
#           mT@tricas y grTÝficos con fechas alineadas.
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KG_TRMACCIONESCONCOVID.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y mapear a canFinicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"[^w\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "proveedor/centro_suministrador": "proveedor",
    "proveedor_centro_suministrador": "proveedor",
    "textobreve": "texto_breve",
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "material": "material",
    "precioneto": "precio_netto",
    "trm": "trm",
    "klabin": "klabin",
    "mondi": "mondi",
    "smurfit": "smurfit",
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) Filtrar columnas requeridas =====
required = ["proveedor", "texto_breve", "fecha_documento", "material",
            "precio_netto", "trm", "klabin", "mondi", "smurfit"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"
df = df[required].copy()

# ===== 4) Tipos, orden temporal y seTtales calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
for c in ["precio_netto", "trm", "klabin", "mondi", "smurfit"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

# Ordenar y limpieza mT-nima
df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# Relleno simple de exTigenas (si aparecen huecos por carga)
for c in ["trm", "klabin", "mondi", "smurfit"]:
    if df[c].isna().any():
        df[c] = df[c].ffill().bfill()

# SeTtales internas calendario
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_netto"
cat_cols = ["proveedor", "texto_breve", "material"]
num_base = ["trm", "klabin", "mondi", "smurfit", "sin_month", "cos_month", "sin_dow", "cos_dow"]

# ===== 6) RarefaciIn top-K en categTiricas =====
TOP_K = 50
def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")
```



```
top_sets = {c: topk_set(df_train[c], TOP_K) for c in cat_cols}
for c in cat_cols:
    df_train[c] = to_topk(df_train[c], top_sets[c])
    df_val[c] = to_topk(df_val[c], top_sets[c])
    df_test[c] = to_topk(df_test[c], top_sets[c])

# ===== 7) Escalado: y con MinMax; X num con StandardScaler; OHE robusto =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

ohe = OneHotEncoder(handle_unknown="ignore") # sin 'sparse=False' para compatibilidad
ohe.fit(df_train[cat_cols])

def _to_dense(X):
    return X.toarray() if hasattr(X, "toarray") else X

def make_tab(df_part):
    X_num = df_part[num_base].to_numpy(dtype=np.float32)
    X_cat = _to_dense(ohe.transform(df_part[cat_cols]))
    return np.hstack([X_num, X_cat]).astype(np.float32)

X_train_tab = make_tab(df_train)
X_val_tab = make_tab(df_val)
X_test_tab = make_tab(df_test)

# Estandarizar solo numéricos (mantén OHE tal cual)
num_scaler = StandardScaler().fit(df_train[num_base].to_numpy(dtype=np.float32))
def scale_tab(X_tab):
    n_num = len(num_base)
    Xn = X_tab.copy()
    Xn[:, :n_num] = num_scaler.transform(Xn[:, :n_num])
    return Xn

X_train_tab = scale_tab(X_train_tab)
X_val_tab = scale_tab(X_val_tab)
X_test_tab = scale_tab(X_test_tab)

# ===== 8) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 9) Modelo LSTM (128B+64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 10) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 11) Evaluación y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()

pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACIÓN ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 12) Gráficas rápidas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de pérdida"); plt.xlabel("época"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicción")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("Índice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()
```

```
res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()
```

```
# ===== 13) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_trm_acc_concovid_precio_netto.keras"))
import joblib
joblib.dump(ohe, os.path.join(SAVE_DIR, "ohe_categorias.joblib"))
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)
```

```
# ===== 14) Gráficos finales con fechas alineadas =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)
```

```
# 2) Fechas alineadas por bloque
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()
```

```
# 3) Chequeos
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"
```

```
# 4) Concatenar todo el recorrido LSTM
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])
```

```
# 5) Serie real completa y plot principal
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_netto"].to_numpy()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year
```

```
plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio netto (COP) vs Real vs LSTM (Con COVID + TRM + Acciones) {year_min} vs {year_max}")
plt.xlabel("Fecha"); plt.ylabel("Precio netto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()
```

```
# --- Panel complementario (cortes + métricas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_netto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_netto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_netto"], label="TEST (real)", alpha=0.8)
```

```
ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")
```

```
ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validación", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")
```

```
ax.set_title("LSTM vs precio_netto (COP) vs Real vs Predicho (Con COVID + TRM + Acciones)")
ax.set_ylabel("Precio netto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")
```

```
y_true_te = df_test["precio_netto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:-m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_netto"].values, y_true_te, pred_te_u, m=1)
txt = f"TEST vs MAE={mae:.1f} | RMSE={rmse:.1f} | sMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()
```

```
# --- Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title("Paridad en Test (Con COVID + TRM + Acciones)")
ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()
```

```
# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("Índice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()
```

```
# =====
#@title LSTM (Con COVID): BATES_50kg_TRMSMURFIT_ConCovid.xlsx
# Vars: Fechadocumento, Precioneto (target), TRM, Smurfit
# Pipeline: renombre columnas vs split temporal vs series calendario vs
#           escalado numérico vs ventanas vs LSTM (128 vs 64) + LayerNorm vs
#           métricas y gráficos con fechas alineadas.
# =====
```

```
import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
SEED = 42
random.seed(SEED); np.random.seed(SEED)
```

```
import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization
```

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50kg_TRMSMURFIT_ConCovid.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y renombrar a canPinicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"^\w\s/", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "precioneto": "precio_netto",
    "trm": "trm",
    "smurfit": "smurfit",
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) VerificaciIn de columnas =====
required = ["fecha_documento", "precio_netto", "trm", "smurfit"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"
df = df[required].copy()

# ===== 4) Tipos, orden temporal y seTtales calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
for c in ["precio_netto", "trm", "smurfit"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

# Orden y limpieza mT-nima
df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# Relleno simple de exTigenas si hay huecos
for c in ["trm", "smurfit"]:
    if df[c].isna().any():
        df[c] = df[c].ffill().bfill()

# SeTtales internas (no exTigenas externas)
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_netto"
num_feats = ["trm", "smurfit", "sin_month", "cos_month", "sin_dow", "cos_dow"]

# ===== 6) Escalado =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

X_train_tab = df_train[num_feats].to_numpy(dtype=np.float32)
X_val_tab = df_val[num_feats].to_numpy(dtype=np.float32)
X_test_tab = df_test[num_feats].to_numpy(dtype=np.float32)

num_scaler = StandardScaler().fit(X_train_tab)
X_train_tab = num_scaler.transform(X_train_tab)
X_val_tab = num_scaler.transform(X_val_tab)
X_test_tab = num_scaler.transform(X_test_tab)

# ===== 7) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1 # ajusta LOOKBACK segTen ACF/estacionalidad

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 8) Modelo LSTM (128B†64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 9) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
```



```
        batch_size=64,
        callbacks=[es, rlr],
        verbose=1
    )

# ===== 10) Evaluaci3n y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACI3N ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 11) Gr3ficos r3pidas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de p3rdida"); plt.xlabel("T3poca"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicci3n")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("T3ndice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 12) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_trm_smurfit_concovid_precio_netto.keras"))
import joblib
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 13) Gr3ficos con fechas alineadas =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas por bloque
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"

# 4) Concatenar todo el recorrido LSTM
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa y plot principal
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_netto"].to_numpy()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio neto (COP) B3 Real vs LSTM (Con COVID B3 TRM + Smurfit) {year_min}B3{year_max}")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario (cortes + m3tricas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_netto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_netto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_netto"], label="TEST (real)", alpha=0.8)

ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validaci3n", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title("LSTM B3 precio_netto (COP) B3 Real vs Predicho (Con COVID B3 TRM + Smurfit)")
ax.set_ylabel("Precio neto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

y_true_te = df_test["precio_netto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:~m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_netto"].values, y_true_te, pred_te_u, m=1)
txt = f"TEST B3 MAE={mae:.1f} | RMSE={rmse:.1f} | sMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()

# --- Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
```

```
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title("Paridad en Test (TRM + Smurfit)")
ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("Índice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

"""**Debemos corroborar aquí a ver si como en la base de datos sion Covid, 14 dΓ-as sigue siendo el regazo de dΓ-as para la mayor correlaciΓin con Precio_net0**"""

#@title === CCF diaria: BicuΓYntos dΓ-as despuΓos se siente Smurfit en Precioneto? ===
import numpy as np, pandas as pd, re, matplotlib.pyplot as plt
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50kg_TRMSMURFIT_ConCovid.xlsx"

# ----- utilidades -----
def slugify(s):
    s = s.strip()
    s = re.sub(r"^\w\s/", "", s)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

RENAME = {
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "precioneto": "precio_net0",
    "smurfit": "smurfit",
}

# ----- parΓmetros -----
# cΓimo tratar varios registros el mismo dΓ-a: "mean", "last" o "sum"
DAILY_AGG = "mean"
# transformaciΓin para quitar tendencia/heterocedasticidad: "pct" (retornos), "diff" (primera diferencia) o None
TRANSFORM = "pct" # <-- suele dar CCF mΓYs interpretable
MAX_LAG_DAYS = 120 # ventana de lags BΓMAX_LAG_DAYS (ajusta si quieres)
FILL_METHOD = "ffill" # cΓimo rellenar dΓ-as sin observaciΓin: "ffill" o None

# ----- cargar y preparar -----
df = pd.read_excel(PATH)
df.columns = [slugify(c) for c in df.columns]
df = df.rename(columns={k:v for k,v in RENAME.items() if k in df.columns})
req = ["fecha_documento", "precio_net0", "smurfit"]
missing = [c for c in req if c not in df.columns]
assert not missing, f"Faltan columnas: {missing}"

df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce")
df["precio_net0"] = pd.to_numeric(df["precio_net0"], errors="coerce")
df["smurfit"] = pd.to_numeric(df["smurfit"], errors="coerce")
df = df.dropna(subset=["fecha_documento", "precio_net0", "smurfit"]).sort_values("fecha_documento")

# Agregar por dΓ-a si hay mΓeltiples registros el mismo dΓ-a
if DAILY_AGG == "mean":
    daily = df.set_index("fecha_documento")[["precio_net0", "smurfit"]].groupby(pd.Grouper(freq="D")).mean()
elif DAILY_AGG == "last":
    daily = df.set_index("fecha_documento")[["precio_net0", "smurfit"]].groupby(pd.Grouper(freq="D")).last()
else:
    daily = df.set_index("fecha_documento")[["precio_net0", "smurfit"]].groupby(pd.Grouper(freq="D")).sum()

# Opcional: rellenar huecos diarios (mantener alineaciΓin dΓ-a a dΓ-a)
if FILL_METHOD == "ffill":
    daily = daily.ffmpeg()

daily = daily.dropna()

# Transformaciones para estacionarizar
if TRANSFORM == "pct":
    S = daily.pct_change().replace([np.inf, -np.inf], np.nan).dropna()
elif TRANSFORM == "diff":
    S = daily.diff().dropna()
else:
    S = daily.copy()

# Estandarizar (z-score) para CCF
Z = (S - S.mean()) / S.std(ddof=0)
y = Z["precio_net0"].values
x = Z["smurfit"].values

# ----- CCF diaria -----
# ConvenciΓin: k > 0 => Smurfit (x) antecede a Precioneto (y) por k dΓ-as
max_lag = int(MAX_LAG_DAYS)
lags = np.arange(-max_lag, max_lag+1)
ccf = []
n = len(Z)
for k in lags:
    if k > 0:
        xs = x[:-k]; ys = y[k:]
    elif k < 0:
        xs = x[-k:]; ys = y[:k]
    else:
        xs = x; ys = y
    ccf.append(np.corrcoef(xs, ys)[0,1] if len(xs) >= 3 else np.nan)
ccf = np.array(ccf, dtype=float)

# Bandas de significancia aproximadas (95%)
N_k = np.array([n - abs(k) for k in lags], dtype=float)
bands = 1.96 / np.sqrt(N_k)
ub, lb = bands, -bands

# Mejor lag "causal" (k>0: Smurfit antecede)
mask_pos = (lags > 0) & ~np.isnan(ccf)
if mask_pos.any():
    k_best_pos = int(lags[mask_pos][np.nanargmax(np.abs(ccf[mask_pos]))])
    ccf_best_pos = float(ccf[lags == k_best_pos])
else:
    k_best_pos, ccf_best_pos = None, np.nan

# TambiΓon reporto el mΓYximo absoluto (por si queda en k<=0)
mask_all = ~np.isnan(ccf)
k_best_any = int(lags[mask_all][np.nanargmax(np.abs(ccf[mask_all]))])
ccf_best_any = float(ccf[lags == k_best_any])
```

```
print(f"Observaciones diarias: {n}")
print(f"Mejor lag positivo (Smurfit antecede) = {k_best_pos} dT-as; CCF = {ccf_best_pos:.3f}" if k_best_pos is not None else
      "No se encontr i lag positivo vT lido en el rango dado.")
print(f"MT ximo absoluto de |CCF| en lags B (MAX_LAG_DAYS) dT-as: k = {k_best_any} (CCF={ccf_best_any:.3f}) "
      f"{'(Smurfit antecede)' if k_best_any>0 else '(Smurfit rezagada)' if k_best_any<0 else '(contempor neo)'}")

# B es significativo el mejor lag positivo?
if k_best_pos is not None:
    sig = abs(ccf_best_pos) > bands[lags == k_best_pos][0]
    print(f"B Significativo (B 95%)? {'SI-' if sig else 'No'} (banda B  B (bands[lags==k_best_pos][0]:.3f))")

# ----- gr fico -----
plt.figure(figsize=(12,4.5))
plt.bar(lags, ccf, width=0.9)
plt.plot(lags, ub, 'k--', linewidth=1)
plt.plot(lags, lb, 'k--', linewidth=1)
plt.axhline(0, color='k', linewidth=0.8)
plt.title("CCF diaria B  Smurfit vs Precioneto (k>0: Smurfit antecede en k dT-as)")
plt.xlabel("Lag (dT-as)")
plt.ylabel("CCF")
plt.tight_layout()
plt.show()

# =====
#@title LSTM (Con COVID) B  TRM + Smurfit rezagado 14
# Path: /content/drive/MyDrive/BATES_50KG_TRMSMURFIT_ConCovid+14.xlsx
# Vars: Fechadocumento, Precioneto (target), TRM, Smurfit
# Nota: se usa smurfit_lag14 = shift(14) como exT gena.
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KG_TRMSMURFIT_ConCovid+14.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y renombrar a can nicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"^\w\s/", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "precioneto": "precio_netto",
    "trm": "trm",
    "smurfit": "smurfit",
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) Verificaci n de columnas =====
required = ["fecha_documento", "precio_netto", "trm", "smurfit"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"
df = df[required].copy()

# ===== 4) Tipos, orden temporal y seT ales calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
for c in ["precio_netto", "trm", "smurfit"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

# Orden y limpieza m nima
df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# Relleno simple de exT genas si hay huecos
for c in ["trm", "smurfit"]:
    if df[c].isna().any():
        df[c] = df[c].ffill().bfill()

# ===== 4b) Crear rezago de Smurfit (+14 pasos) =====
SHIFT_SMURFIT = 14
df["smurfit_lag14"] = df["smurfit"].shift(SHIFT_SMURFIT)

# SeT ales internas (calendario)
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# El rezago introduce NaNs al inicio: recorta filas sin info completa
df = df.dropna(subset=["smurfit_lag14"]).reset_index(drop=True)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_netto"
# Usamos TRM contempor nea + Smurfit rezagada + seT ales calendario
num_feats = ["trm", "smurfit_lag14", "sin_month", "cos_month", "sin_dow", "cos_dow"]

# ===== 6) Escalado =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
```

```
y_test = y_scaler.transform(df_test[[TARGET]])

X_train_tab = df_train[num_feats].to_numpy(dtype=np.float32)
X_val_tab = df_val[num_feats].to_numpy(dtype=np.float32)
X_test_tab = df_test[num_feats].to_numpy(dtype=np.float32)

num_scaler = StandardScaler().fit(X_train_tab)
X_train_tab = num_scaler.transform(X_train_tab)
X_val_tab = num_scaler.transform(X_val_tab)
X_test_tab = num_scaler.transform(X_test_tab)

# ===== 7) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1 # ajusta LOOKBACK si tu ACF sugiere otra memoria

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 8) Modelo LSTM (128B†'64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 9) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 10) Evaluaci¶n y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACION ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 11) Gr¶ficos b¶sicos =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de p¶rdida"); plt.xlabel("T¶poca"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicci¶n")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("T¶ndice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 12) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_trm_smulag14_concovid_precio_netto.keras"))
import joblib
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 13) Gr¶ficos con fechas alineadas =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas por bloque
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"
```

```
# 4) Concatenar todo el recorrido LSTM
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa y plot principal
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_netto"].to_numpy()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio netto (COP) vs Real vs LSTM (TRM + Smurfit_lag{SHIFT_SMURFIT} B Con COVID) {year_min} vs {year_max}")
plt.xlabel("Fecha"); plt.ylabel("Precio netto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario (cortes + métricas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_netto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_netto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_netto"], label="TEST (real)", alpha=0.8)

ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validación", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title(f"LSTM B precio_netto (COP) vs Real vs Predicho (TRM + Smurfit_lag{SHIFT_SMURFIT} B Con COVID)")
ax.set_ylabel("Precio netto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

y_true_te = df_test["precio_netto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:-m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_netto"].values, y_true_te, pred_te_u, m=1)
txt = f"TEST B MAE={mae:.1f} | RMSE={rmse:.1f} | SMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()

# --- Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title(f"Paridad en Test (TRM + Smurfit_lag{SHIFT_SMURFIT})")
ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("Índice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

# =====
#@title LSTM (Con COVID) B TRM + Smurfit rezagado 42
# Path: /content/drive/MyDrive/BATES_50KG_TRMSMURFIT_ConCOVID+42.xlsx
# Vars: Fechadocumento, Precioneto (target), TRM, Smurfit
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KG_TRMSMURFIT_ConCOVID+42.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y renombrar =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"[^\\w\\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "precioneto": "precio_netto",
    "trm": "trm",
    "smurfit": "smurfit",
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) Verificación de columnas =====
required = ["fecha_documento", "precio_netto", "trm", "smurfit"]
missing = [c for c in required if c not in df.columns]
```



```
assert not missing, f"Faltan columnas requeridas: {missing}"
df = df[required].copy()

# ===== 4) Tipos, orden temporal y seTtales calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
for c in ["precio_netto", "trm", "smurfit"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# Relleno simple de exTigenas si hay huecos
for c in ["trm", "smurfit"]:
    if df[c].isna().any():
        df[c] = df[c].ffill().bfill()

# ===== 4b) Crear rezago de Smurfit (+42) =====
SHIFT_SMURFIT = 42
df["smurfit_lag42"] = df["smurfit"].shift(SHIFT_SMURFIT)

# SeTtales de calendario
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# Quita las filas iniciales sin el rezago completo
df = df.dropna(subset=["smurfit_lag42"]).reset_index(drop=True)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_netto"
num_feats = ["trm", "smurfit_lag42", "sin_month", "cos_month", "sin_dow", "cos_dow"]

# ===== 6) Escalado =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

X_train_tab = df_train[num_feats].to_numpy(dtype=np.float32)
X_val_tab = df_val[num_feats].to_numpy(dtype=np.float32)
X_test_tab = df_test[num_feats].to_numpy(dtype=np.float32)

num_scaler = StandardScaler().fit(X_train_tab)
X_train_tab = num_scaler.transform(X_train_tab)
X_val_tab = num_scaler.transform(X_val_tab)
X_test_tab = num_scaler.transform(X_test_tab)

# ===== 7) Ventanas deslizantes =====
LOOKBACK, HORIZON = 60, 1 # ajusta si tu ACF sugiere otra memoria

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 8) Modelo LSTM (128B†'64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 9) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 10) EvaluaciTn y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACIITN ==", metrics_val)
```

```
print("== TEST ==", metrics_test)

# ===== 11) Gráficas rípidas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de pérdida"); plt.xlabel("Época"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicción")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("Índice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 12) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_trm_smulag42_concovid_precio_netto.keras"))
import joblib
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 13) Gráficos con fechas alineadas =====
# 1) Predicciones de TRAIN
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_tr_u = inv_y(pred_tr)

# 2) Fechas alineadas por bloque
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

# 3) Chequeos
assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"

# 4) Concatenación
fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])

# 5) Serie real completa y plot principal
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_netto"].to_numpy()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio neto (COP) vs Real vs LSTM (TRM + Smurfit_lag{SHIFT_SMURFIT}) B· Con COVID) {year_min} vs {year_max}")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_netto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_netto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_netto"], label="TEST (real)", alpha=0.8)

ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validación", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title(f"LSTM B· precio_netto (COP) vs Real vs Predicho (TRM + Smurfit_lag{SHIFT_SMURFIT}) B· Con COVID)")
ax.set_ylabel("Precio neto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

y_true_te = df_test["precio_netto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr); scale=np.mean(np.abs(ytr[m:]-ytr[:-m]));
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_netto"].values, y_true_te, pred_te_u, m=1)
txt = f"TEST at MAE={mae:.1f} | RMSE={rmse:.1f} | SMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()

# --- Paridad y residuos ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title(f"Paridad en Test (TRM + Smurfit_lag{SHIFT_SMURFIT})")
ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("Índice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

"""Después de haber entrenado los modelos LSTM con las diferentes bases de datos, vamos a ver por métricas cual de todos fue el de mejor rendimiento.

A continuación:
"""

# =====
#@title BENCHMARK LSTM B· Comparación de múltiples datasets
# - Split temporal fijo (70/15/15)
# - Rarefacción + OneHot (si hay categorías)
```

```
# - ExRigenas numF©tricas seg¶en dataset (TRM, Klabin, Mondi, Smurfit)
# - Opci¶in de rezago (p.ej. Smurfit_lag14)
# - Ventanas LSTM (LOOKBACK=60, HORIZON=1)
# - M¶©tricas: MAE, RMSE, sMAPE, MASE (VALID y TEST)
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ----- utilidades comunes -----
def slugify(s):
    s = s.strip()
    s = re.sub(r"^[^\\w\\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\\s+", "_", s)
    return s.lower()

RENAME_MAP = {
    "proveedor/centro_suministrador": "proveedor",
    "proveedor_centro_suministrador": "proveedor",
    "textobreve": "texto_breve",
    "fechadocumento": "fecha_documento",
    "fecha_documento": "fecha_documento",
    "material": "material",
    "precioneto": "precio_netto",
    "trm": "trm",
    "klabin": "klabin",
    "mondi": "mondi",
    "smurfit": "smurfit",
    "cantidaddepedido": "cantidad_pedido",
    "cantidad_depedido": "cantidad_pedido",
}

def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

def mase(ytr, yt, yp, m=1):
    ytr=np.asarray(ytr)
    scale=np.mean(np.abs(ytr[m:]-ytr[:-m]))
    scale = scale if (scale is not None and scale!=0 and not np.isnan(scale)) else 1.0
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/scale

def make_windows(tab_X, y, lookback=60, horizon=1):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

def build_lstm(input_timesteps, n_features):
    inp = Input(shape=(input_timesteps, n_features))
    x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
    x = LayerNormalization()(x)
    x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
    x = LayerNormalization()(x)
    x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
    x = Dropout(0.2)(x)
    out = Dense(1, activation="linear")(x)
    model = models.Model(inp, out)
    opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
    model.compile(optimizer=opt, loss="mse", metrics=["mae"])
    return model

def to_dense(X):
    return X.toarray() if hasattr(X, "toarray") else X

# ----- funci¶in principal por dataset -----
def run_experiment(
    name,
    path,
    use_cat=False,
    cat_cols=("proveedor","texto_breve","material"),
    num_exog=(),
    lag_feat=None, # e.g. {"col":"smurfit", "lag":14, "alias":"smurfit_lag14"}
    lookback=60,
    horizon=1,
    top_k=50,
    epochs=200,
    batch_size=64,
    verbose=0
):
    # Cargar
    df = pd.read_excel(path)
    df.columns = [slugify(c) for c in df.columns]
    df = df.rename(columns={k:v for k,v in RENAME_MAP.items() if k in df.columns})

    # Requeridos b¶sicos
    required = ["fecha_documento", "precio_netto"]
    if use_cat:
        required += list(cat_cols)
    required += list(num_exog)
    missing = [c for c in required if c not in df.columns]
    assert not missing, f"[{name}] Faltan columnas requeridas: {missing}"

    df = df[required].copy()

    # Tipos y fecha
    df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
    df["precio_netto"] = pd.to_numeric(df["precio_netto"], errors="coerce")
    for c in num_exog:
        df[c] = pd.to_numeric(df[c], errors="coerce")

    df = df.dropna(subset=["fecha_documento","precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

    # Relleno exRigenas si hay
    for c in num_exog:
        if df[c].isna().any():
            df[c] = df[c].ffill().bfill()
```



```

# Lag opcional
if lag_feat:
    col = lag_feat["col"]; lag = lag_feat.get("lag", 14); alias = lag_feat.get("alias", f"{col}_lag{lag}")
    df[alias] = df[col].shift(lag)
    df = df.dropna(subset=[alias]).reset_index(drop=True)
    # sustituir en num_exog por el alias
    num_exog = tuple([alias if c==col else c for c in num_exog])

# Seriales calendario
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

num_base = list(num_exog) + ["sin_month", "cos_month", "sin_dow", "cos_dow"]

# Split temporal
n = len(df); i_train = int(n*0.7); i_val = int(n*0.85)
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

# y scaler
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[["precio_netto"]])
y_val = y_scaler.transform(df_val[["precio_netto"]])
y_test = y_scaler.transform(df_test[["precio_netto"]])

# OHE si aplica
if use_cat and len(cat_cols)>0:
    # rarefaci n top-K por columna
    def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
    def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")
    for c in cat_cols:
        top = topk_set(df_train[c], top_k)
        df_train[c] = to_topk(df_train[c], top)
        df_val[c] = to_topk(df_val[c], top)
        df_test[c] = to_topk(df_test[c], top)

    ohe = OneHotEncoder(handle_unknown="ignore")
    ohe.fit(df_train[list(cat_cols)])

    def make_tab(df_part):
        X_num = df_part[num_base].to_numpy(dtype=np.float32)
        X_cat = to_dense(ohe.transform(df_part[list(cat_cols)]))
        return np.hstack([X_num, X_cat]).astype(np.float32)
else:
    def make_tab(df_part):
        return df_part[num_base].to_numpy(dtype=np.float32)

X_train_tab = make_tab(df_train)
X_val_tab = make_tab(df_val)
X_test_tab = make_tab(df_test)

# Escalado num rico (solo primeras columnas num ricas)
num_scaler = StandardScaler().fit(df_train[num_base].to_numpy(dtype=np.float32))
n_num = len(num_base)
def scale_tab(X):
    Xn = X.copy()
    Xn[:, :n_num] = num_scaler.transform(Xn[:, :n_num])
    return Xn
X_train_tab = scale_tab(X_train_tab)
X_val_tab = scale_tab(X_val_tab)
X_test_tab = scale_tab(X_test_tab)

# Ventanas
Xtr, ytr = make_windows(X_train_tab, y_train, lookback, horizon)
Xva, yva = make_windows(np.vstack([X_train_tab[-lookback+1:], X_val_tab]),
                        np.vstack([y_train[-lookback+1:], y_val]), lookback, horizon)
Xte, yte = make_windows(np.vstack([X_val_tab[-lookback+1:], X_test_tab]),
                        np.vstack([y_val[-lookback+1:], y_test]), lookback, horizon)

# Modelo
model = build_lstm(lookback, Xtr.shape[-1])
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1 if verbose else 0)
model.fit(Xtr, ytr, validation_data=(Xva, yva),
        epochs=epochs, batch_size=batch_size,
        callbacks=[es, rlr], verbose=verbose)

# Predicciones (desescaladas)
inv_y = lambda y: y_scaler.inverse_transform(y.reshape(-1,1)).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()
yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

# M tricas
def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
metrics = {
    "NAME": name,
    "VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
    "VAL_RMSE": rmse(yva_true, pred_va_u),
    "VAL_sMAPE(%)": smape(yva_true, pred_va_u),
    "TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
    "TEST_RMSE": rmse(yte_true, pred_te_u),
    "TEST_sMAPE(%)": smape(yte_true, pred_te_u),
    "TEST_MASE": mase(df_train["precio_netto"].values, yte_true, pred_te_u, m=1),
    "N": len(df)
}
return metrics

# ----- Lista de experimentos -----
EXPERIMENTS = [
    # BASE (sin TRM, sin acciones) B  con categoricas
    dict(name="BASE (sin TRM)", path="/content/drive/MyDrive/BATES_50KG.xlsx",
        use_cat=True, num_exog=()),

    # TRM
    dict(name="TRM", path="/content/drive/MyDrive/BATES_50KG_TRM.xlsx",
        use_cat=True, num_exog=("trm",)),

    # TRM + Acciones
    dict(name="TRM+ACC", path="/content/drive/MyDrive/BATES_50KG_TRM+ACCIONES.xlsx",
        use_cat=True, num_exog=("trm", "klabin", "mondi", "smurfit")),

    # TRM+Smurfit (sin categoricas)
    dict(name="TRM+Smurfit", path="/content/drive/MyDrive/BATES_50KG_TRMSMURFIT.xlsx",
        use_cat=False, num_exog=("trm", "smurfit")),

```

```
# TRM+Smurfit (lag14)
dict(name="TRM+Smurfit_lag14", path="/content/drive/MyDrive/BATES_50KGTRMSMURFIT+14.xlsx",
      use_cat=False, num_exog=("trm", "smurfit"),
      lag_feat={"col": "smurfit", "lag": 14, "alias": "smurfit_lag14"}),

# Con COVID (sin exFigenas) bB" con categTiricas
dict(name="ConCOVID (BASE)", path="/content/drive/MyDrive/BATE_50KG_CONCOVID.xlsx",
      use_cat=True, num_exog=()),

# Con COVID + TRM bB" con categTiricas
dict(name="ConCOVID + TRM", path="/content/drive/MyDrive/BATES_50KG_TRMCONCOVID.xlsx",
      use_cat=True, num_exog=("trm",)),

# Con COVID + TRM+ACC bB" con categTiricas
dict(name="ConCOVID + TRM+ACC", path="/content/drive/MyDrive/BATES_50KG_TRMACCIONESCONCOVID.xlsx",
      use_cat=True, num_exog=("trm", "klabin", "mondi", "smurfit")),

# Con COVID B· TRM+Smurfit (sin categTiricas)
dict(name="ConCOVID B· TRM+Smurfit", path="/content/drive/MyDrive/BATES_50kg_TRMSMURFIT_ConCovid.xlsx",
      use_cat=False, num_exog=("trm", "smurfit")),

# Con COVID B· TRM+Smurfit (lag14)
dict(name="ConCOVID B· TRM+Smurfit_lag14", path="/content/drive/MyDrive/BATES_50KG_TRMSMURFIT_ConCovid+14.xlsx",
      use_cat=False, num_exog=("trm", "smurfit"),
      lag_feat={"col": "smurfit", "lag": 14, "alias": "smurfit_lag14"}),

]

# ----- Ejecutar benchmarks -----
from google.colab import drive
drive.mount("/content/drive")

results = []
for cfg in EXPERIMENTS:
    try:
        print(f"\n>>> Ejecutando: {cfg['name']}")
        m = run_experiment(*cfg, epochs=200, batch_size=64, verbose=1)
        results.append(m)
    except Exception as e:
        print(f"[ERROR] {cfg['name']}: {e}")

df_res = pd.DataFrame(results)
if not df_res.empty:
    cols = ["NAME", "N", "VAL_MAE", "VAL_RMSE", "VAL_sMAPE(%)", "TEST_MAE", "TEST_RMSE", "TEST_sMAPE(%)", "TEST_MASE"]
    df_res = df_res[cols].sort_values("TEST_RMSE").reset_index(drop=True)
    print("\n=== TABLA COMPARATIVA (ordenada por TEST_RMSE) ===")
    from IPython.display import display
    display(df_res)
else:
    print("No se obtuvieron resultados (revisa errores anteriores).")

"""#*****
# RUNNER COMPARATIVO DE MODELOS (LSTM)
# Compara mFeltiples datasets/variantes (BASE, TRM, ACCIONES, COVID, lags)#
"""

# =====
#@title RUNNER COMPARATIVO DE MODELOS (LSTM)
# Compara mFeltiples datasets/variantes (BASE, TRM, ACCIONES, COVID, lags)
# Salida: tabla de mFetricas (MAE, RMSE, sMAPE, MASE), ranking y grFřficos
# =====

import os, re, math, random, warnings, gc
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ----- Helpers -----
def slugify(s):
    s = s.strip()
    s = re.sub(r"^[^w\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

def rename_canon(df):
    m = {
        "proveedor/centro_suministrador": "proveedor",
        "proveedor_centro_suministrador": "proveedor",
        "textobreve": "texto_breve",
        "fechadocumento": "fecha_documento",
        "fecha_documento": "fecha_documento",
        "material": "material",
        "precioneto": "precio_netto",
        "trm": "trm",
        "klabin": "klabin",
        "mondi": "mondi",
        "smurfit": "smurfit",
    }
    df.columns = [slugify(c) for c in df.columns]
    df = df.rename(columns={k:v for k,v in m.items() if k in df.columns})
    return df

def add_calendar(df):
    df["dow"] = df["fecha_documento"].dt.dayofweek
    df["month"] = df["fecha_documento"].dt.month
    df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
    df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
    df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
    df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)
    return df

def make_windows(tab_X, y, lookback=60, horizon=1):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

def build_lstm(input_dim, lookback=60):
    inp = Input(shape=(lookback, input_dim))
    x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
```

```
x = LayerNormalization() (x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1) (x)
x = LayerNormalization() (x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4)) (x)
x = Dropout(0.2) (x)
out = Dense(1, activation="linear") (x)
model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
return model

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100
def mase(ytr, yt, yp, m=1):
    ytr = np.asarray(ytr); denom = np.mean(np.abs(ytr[m:]-ytr[:-m]))
    denom = denom if denom!=0 else 1.0
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/denom

# ----- Configs: agrega/quita segUn lo que tengas -----
DATASETS = [
    # BASES sin TRM (si tienes esta)
    ("/content/drive/MyDrive/BATES_50KG.xlsx", dict()),
    # TRM
    ("/content/drive/MyDrive/BATES_50KG_TRM.xlsx", dict()),
    # TRM + ACCIONES
    ("/content/drive/MyDrive/BATES_50KG_TRM+ACCIONES.xlsx", dict()),
    # TRM + Smurfit
    ("/content/drive/MyDrive/BATES_50KG_TRMSMURFIT.xlsx", dict()),
    ("/content/drive/MyDrive/BATES_50KGTRMSMURFIT+14.xlsx", dict(lag_smurfit=14)),
    # Con COVID (categOricas + calendario)
    ("/content/drive/MyDrive/BATE_50KG_CONCOVID.xlsx", dict()),
    # Con COVID + TRM
    ("/content/drive/MyDrive/BATES_50KG_TRMCONCOVID.xlsx", dict()),
    # Con COVID + TRM + ACCIONES
    ("/content/drive/MyDrive/BATES_50KG_TRMACCIONESCONCOVID.xlsx", dict()),
    # Con COVID + TRM + Smurfit (contemporOrneo y lags)
    ("/content/drive/MyDrive/BATES_50kg_TRMSMURFIT_ConCovid.xlsx", dict()),
    ("/content/drive/MyDrive/BATES_50KG_TRMSMURFIT_ConCovid+14.xlsx", dict(lag_smurfit=14)),
    ("/content/drive/MyDrive/BATES_50KG_TRMSMURFIT_ConCOVID+42.xlsx", dict(lag_smurfit=42)),
]

LOOKBACK, HORIZON = 60, 1
train_ratio, val_ratio = 0.7, 0.15

results = []
best_models_cache = {} # opcional: guarda el modelo por path

for path, opts in DATASETS:
    if not os.path.exists(path):
        print(f"[SKIP] No existe: {path}")
        continue

    print(f"\n=== Procesando: {os.path.basename(path)} ===")
    df = pd.read_excel(path)
    df = rename_canon(df)

    # Determinar columnas disponibles
    cols = df.columns.tolist()
    required_min = ["fecha_documento", "precio_net"]
    for c in required_min:
        if c not in cols:
            print(f"[SKIP] Faltan columnas mOr-nimas en {path}: {required_min}")
            break
    else:
        # Tipos y orden
        df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
        df["precio_net"] = pd.to_numeric(df["precio_net"], errors="coerce")
        # num exOrigenas opcionales
        for c in ["trm", "klabin", "mondi", "smurfit"]:
            if c in df.columns:
                df[c] = pd.to_numeric(df[c], errors="coerce")

        df = df.dropna(subset=["fecha_documento", "precio_net"]).sort_values("fecha_documento").reset_index(drop=True)

        # Relleno sencillo para exOrigenas numOricas
        for c in ["trm", "klabin", "mondi", "smurfit"]:
            if c in df.columns and df[c].isna().any():
                df[c] = df[c].ffill().bfill()

        # Lags Smurfit si aplica por config o por nombre de archivo
        lag = opts.get("lag_smurfit", 0)
        if lag == 0:
            # TambiOrn autodetecta del nombre del archivo +14/+42
            if "+14" in path: lag = 14
            if "+42" in path: lag = 42

        if ("smurfit" in df.columns) and lag>0:
            df[f"smurfit_lag{lag}"] = df["smurfit"].shift(lag)
            df = df.dropna(subset=[f"smurfit_lag{lag}"]).reset_index(drop=True)

        # SeOrtales calendario
        df = add_calendar(df)

        # ParticiOrn temporal
        n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
        if i_val <= LOOKBACK + 5:
            print(f"[SKIP] Muy pocos datos tras split/lag para: {path}")
            continue

        df_train = df.iloc[:i_train].copy()
        df_val = df.iloc[i_train:i_val].copy()
        df_test = df.iloc[i_val:].copy()

        # Definir features segUn lo disponible
        cat_cols = [c for c in ["proveedor", "texto_breve", "material"] if c in df.columns]
        num_feats = []
        # exOrigenas numOricas posibles
        for c in ["trm", "klabin", "mondi", "smurfit"]:
            if c in df.columns:
                num_feats.append(c)
        # si tenemos lag, usar el lag en vez de smurfit contemporOrneo
        if lag>0 and ("smurfit" in num_feats):
            num_feats.remove("smurfit")
            num_feats.append(f"smurfit_lag{lag}")

        # seOrtales calendario siempre
        num_feats += ["sin_month", "cos_month", "sin_dow", "cos_dow"]

        # Escalado objetivo
        y_scaler = MinMaxScaler()
```

```
y_train = y_scaler.fit_transform(df_train[["precio_neto"]])
y_val = y_scaler.transform(df_val[["precio_neto"]])
y_test = y_scaler.transform(df_test[["precio_neto"]])

# One-hot categoricas si las hay
if len(cat_cols)>0:
    # rarefacci3n top-K por columna, consistente con train
    TOP_K = 50
    def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
    def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")
    top_sets = {c: topk_set(df_train[c], TOP_K) for c in cat_cols}
    for c in cat_cols:
        df_train[c] = to_topk(df_train[c], top_sets[c])
        df_val[c] = to_topk(df_val[c], top_sets[c])
        df_test[c] = to_topk(df_test[c], top_sets[c])

    ohe = OneHotEncoder(handle_unknown="ignore")
    ohe.fit(df_train[cat_cols])

    def _to_dense(X): return X.toarray() if hasattr(X,"toarray") else X
    def make_tab(dpart):
        X_num = dpart[num_feats].to_numpy(dtype=np.float32)
        X_cat = _to_dense(ohe.transform(dpart[cat_cols]))
        return np.hstack([X_num, X_cat]).astype(np.float32)
else:
    def make_tab(dpart):
        return dpart[num_feats].to_numpy(dtype=np.float32)

# Escalado num3rico
X_train_tab = make_tab(df_train)
X_val_tab = make_tab(df_val)
X_test_tab = make_tab(df_test)
n_num = len(num_feats)
num_scaler = StandardScaler().fit(X_train_tab[:, :n_num])
def scale_tab(X):
    Xn = X.copy()
    Xn[:, :n_num] = num_scaler.transform(Xn[:, :n_num])
    return Xn
X_train_tab = scale_tab(X_train_tab)
X_val_tab = scale_tab(X_val_tab)
X_test_tab = scale_tab(X_test_tab)

# Ventanas
Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val])), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test])), LOOKBACK, HORIZON)

# Modelo
model = build_lstm(Xtr.shape[-1], LOOKBACK)
es = callbacks.EarlyStopping(monitor="val_loss", patience=12, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=6, min_lr=1e-5, verbose=0)

_ = model.fit(Xtr, ytr,
              validation_data=(Xva, yva),
              epochs=120, batch_size=64,
              callbacks=[es, rlr],
              verbose=0)

# Predicciones y m3tricas
inv_y = lambda ys: y_scaler.inverse_transform(ys.reshape(-1,1)).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()
yte_true = inv_y(yte.ravel()); pred_te_u = inv_y(pred_te)

# base-line para MASE: serie de train original
mae_ = mean_absolute_error(yte_true, pred_te_u)
rmse_ = rmse(yte_true, pred_te_u)
smap_ = smape(yte_true, pred_te_u)
mase_ = mase(df_train["precio_neto"].values, yte_true, pred_te_u, m=1)

tag = os.path.basename(path)
if lag>0: tag += f" | lagSmurfit={lag}"
results.append({
    "modelo": tag,
    "features": f"cats={cat_cols} | nums={num_feats}",
    "n_train": len(df_train), "n_val": len(df_val), "n_test": len(df_test),
    "MAE": mae_, "RMSE": rmse_, "sMAPE(%)": smap_, "MASE": mase_
})

# (Opcional) Guardar el mejor para graficar paridad luego
best_models_cache[tag] = dict(model=model, Xte=Xte, yte_true=yte_true, pred_te_u=pred_te_u)

# Limpieza
del model, Xtr, ytr, Xva, yva, Xte, yte, X_train_tab, X_val_tab, X_test_tab
gc.collect()

# ----- Resultados -----
if len(results)==0:
    raise SystemExit("No se generaron resultados. Revisa rutas y columnas.")

df_res = pd.DataFrame(results)
df_res = df_res.sort_values(by=["MASE", "sMAPE(%)", "RMSE", "MAE"]).reset_index(drop=True)
display_cols = ["modelo", "n_train", "n_val", "n_test", "MAE", "RMSE", "sMAPE(%)", "MASE", "features"]
print("\n=== LEADERBOARD ===")
print(df_res[display_cols].to_string(index=False))

# Exportar CSV
out_csv = "/content/leaderboard_lstm_bates.csv"
df_res.to_csv(out_csv, index=False)
print(f"\nm3tricas exportadas a: {out_csv}")

# Barras para MASE (m3rys interpretables entre series)
plt.figure(figsize=(10, max(4, 0.4*len(df_res))))
plt.barh(df_res["modelo"], df_res["MASE"])
plt.gca().invert_yaxis()
plt.xlabel("MASE (m3rys bajo es mejor)")
plt.title("Leaderboard por MASE")
plt.grid(axis="x", alpha=0.3)
plt.tight_layout(); plt.show()

# Barras para sMAPE
plt.figure(figsize=(10, max(4, 0.4*len(df_res))))
plt.barh(df_res["modelo"], df_res["sMAPE(%)"])
plt.gca().invert_yaxis()
plt.xlabel("sMAPE % (m3rys bajo es mejor)")
plt.title("Leaderboard por sMAPE")
plt.grid(axis="x", alpha=0.3)
plt.tight_layout(); plt.show()

# Paridad del mejor modelo
best_tag = df_res.iloc[0]["modelo"]
bt = best_models_cache[best_tag]
```

```
y_true = bt["yte_true"]; y_pred = bt["pred_te_u"]
plt.figure(figsize=(5.5,5.5))
plt.scatter(y_true, y_pred, s=12, alpha=0.7)
lims = [min(y_true.min(), y_pred.min()), max(y_true.max(), y_pred.max())]
plt.plot(lims, lims, 'k--', lw=1)
plt.xlim(lims); plt.ylim(lims)
plt.title(f"Paridad (TEST) B· Mejor modelo: {best_tag}")
plt.xlabel("Real (COP)"); plt.ylabel("Predicho (COP)")
plt.grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

"""# **Para mejorar **

Backtesting con ventanas rodantes (entrenar+calibrar+predecir 1 semana, paso semanal).

Ensemble de SARIMAX + LSTM + XGBoost (pesos por desempeño en calibración).

Búsqueda de lags óptimos para TRM/Smurfit en 0-56 semanas (configurable).

Features transformadas: retornos, log-retornos y medias móviles (4/12/26/52 semanas).

Reentrenos semanales.

Intervalos de predicción (Conformal) + opción Bootstrap.

Estabilidad temporal (sMAPE por trimestre) y chequeo de drift (prueba no paramétrica sobre residuos).
"""

!pip -q install lightgbm==4.3.0 openpyxl==3.1.2 matplotlib==3.9.0 pandas==2.2.2 scikit-learn==1.4.2

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import lightgbm as lgb
from sklearn.model_selection import TimeSeriesSplit, RandomizedSearchCV
from sklearn.metrics import mean_absolute_error, r2_score
from scipy.stats import randint, uniform, loguniform
from datetime import datetime

# ----- (A) Carga del archivo desde Drive -----
from google.colab import drive
drive.mount('/content/drive')

# Ruta fija al archivo Excel
fname = "/content/drive/MyDrive/BATES_50KG_DIARIO_COMPLETO_FINAL.xlsx"
print(f"Archivo detectado: {fname}")

# ----- (B) Lectura y saneamiento básico -----
usecols = [
    "Fechadocumento",
    "Precioneto",
    "TRM",
    "Klabin",
    "Mondi",
    "Smurfit",
    "Mes",
    "Alto"
]

df = pd.read_excel(fname, engine="openpyxl")
colset = set(df.columns)
missing = set(usecols) - colset
if missing:
    raise ValueError(f"Faltan columnas en el archivo: {missing}\nColumnas disponibles: {sorted(df.columns)}")

df = df[usecols].copy()

df["Fechadocumento"] = pd.to_datetime(df["Fechadocumento"], dayfirst=True, errors="coerce")
assert df["Fechadocumento"].notna().all(), "Hay fechas no parseables; revisa el formato de 'Fechadocumento'."
df = df.sort_values("Fechadocumento").reset_index(drop=True)

for c in ["Precioneto", "TRM", "Klabin", "Mondi", "Smurfit", "Mes", "Alto"]:
    df[c] = pd.to_numeric(df[c], errors="coerce")

for c in ["TRM", "Klabin", "Mondi", "Smurfit"]:
    df[c] = df[c].ffill().bfill()

# ----- (C) Ingeniería de características -----
def make_features(frame: pd.DataFrame) -> pd.DataFrame:
    df_ = frame.copy()
    df_["date"] = df_["Fechadocumento"]
    df_["dayofweek"] = df_["date"].dt.dayofweek
    df_["quarter"] = df_["date"].dt.quarter
    df_["Mes"] = df_["Mes"].astype(int, errors="ignore")
    df_["Alto"] = df_["Alto"].astype(int, errors="ignore")

    # Lags del objetivo
    for L in [1, 7, 14, 30]:
        df_[f"Precioneto_lag{L}"] = df_["Precioneto"].shift(L)

    # Lags y retornos de exigidas
    for col in ["TRM", "Klabin", "Mondi", "Smurfit"]:
        for L in [1, 5, 22]:
            df_[f"{col}_lag{L}"] = df_[col].shift(L)
            df_[f"{col}_ret1"] = np.log(df_[col] / df_[col].shift(1))
            df_[f"{col}_ret5"] = np.log(df_[col] / df_[col].shift(5))
            df_[f"{col}_ret1_vol10"] = (
                np.log(df_[col] / df_[col].shift(1)).rolling(10, min_periods=5).std()
            )

    df_["Precioneto_ma7"] = df_["Precioneto"].shift(1).rolling(7, min_periods=3).mean()
    df_["Precioneto_ma14"] = df_["Precioneto"].shift(1).rolling(14, min_periods=5).mean()
    df_["Precioneto_delta1"] = df_["Precioneto"].diff(1)
    return df_

df_feat = make_features(df)
df_feat = df_feat.dropna().reset_index(drop=True)

# ----- (D) Split temporal 90%/10% -----
n = len(df_feat)
split_idx = int(n * 0.90)
train = df_feat.iloc[:split_idx].copy()
test = df_feat.iloc[split_idx:].copy()

target_col = "Precioneto"
feature_cols = [c for c in df_feat.columns if c not in ["Fechadocumento", "date", target_col]]

X_train, y_train = train[feature_cols], train[target_col]
X_test, y_test = test[feature_cols], test[target_col]
dates_test = test["date"].copy()

print(f"Total despus de lags: {n}")
print(f"Train: {X_train.shape}, Test: {X_test.shape}")
```

```
# ----- (E) ValidaciIn temporal + RandomizedSearchCV -----
tscv = TimeSeriesSplit(n_splits=3)
lgb_est = lgb.LGBMRegressor(
    objective="mae",
    n_estimators=600,
    boosting_type="gbdt",
    random_state=42,
    n_jobs=-1
)

param_distributions = {
    "num_leaves": randint(15, 256),
    "max_depth": randint(3, 12),
    "learning_rate": loguniform(1e-3, 3e-1),
    "min_child_samples": randint(10, 200),
    "subsample": uniform(0.6, 0.4),
    "colsample_bytree": uniform(0.6, 0.4),
    "reg_alpha": loguniform(1e-8, 1e-1),
    "reg_lambda": loguniform(1e-8, 1e-1),
}

search = RandomizedSearchCV(
    estimator=lgb_est,
    param_distributions=param_distributions,
    n_iter=100,
    scoring="neg_mean_absolute_error",
    cv=tscv,
    verbose=1,
    n_jobs=-1,
    refit=True,
    random_state=42
)

search.fit(X_train, y_train)
best_model = search.best_estimator_
print("\nMejores hiperparYmetros encontrados:")
print(search.best_params_)

# ----- (F) PredicciIn sobre holdout -----
y_pred = best_model.predict(X_test)

# ----- (G) MTricas -----
def mape(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    eps = 1e-8
    return np.mean(np.abs((y_true - y_pred) / np.maximum(np.abs(y_true), eps))) * 100

def smape(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    denom = (np.abs(y_true) + np.abs(y_pred)) / 2.0
    denom = np.where(denom == 0, 1.0, denom)
    return np.mean(np.abs(y_pred - y_true) / denom) * 100

def wape(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.sum(np.abs(y_true - y_pred)) / np.sum(np.abs(y_true)) * 100

mae = mean_absolute_error(y_test, y_pred)
mape_v = mape(y_test, y_pred)
smape_v = smape(y_test, y_pred)
wape_v = wape(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\n=== DesempeIo en el 10% mYs reciente (holdout) ===")
print(f"MAE : {mae:,.4f}")
print(f"MAPE : {mape_v:,.2f}%")
print(f"sMAPE : {smape_v:,.2f}%")
print(f"WAPE : {wape_v:,.2f}%")
print(f"RBI : {r2:,.4f}")

# ----- (H) GrYficas -----
plt.figure(figsize=(12, 5))
plt.plot(dates_test, y_test.values, label="Real (Precioneto)", linewidth=2)
plt.plot(dates_test, y_pred, label="Pronosticado (LightGBM)", linestyle="--")
plt.title("Holdout (10% mYs reciente): Real vs Pronosticado")
plt.xlabel("Fecha")
plt.ylabel("Precio Neto (COP)")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# ----- (I) Importancia de variables -----
imp = pd.Series(best_model.feature_importances_, index=feature_cols)
imp = imp.sort_values(ascending=False).head(20)

plt.figure(figsize=(8, 7))
imp.sort_values().plot(kind="barh")
plt.title("Top 20 Importancias de Variables (LightGBM)")
plt.xlabel("Importancia")
plt.tight_layout()
plt.show()

# ----- (J) Exportar predicciones -----
pred_df = pd.DataFrame({
    "Fechadocumento": dates_test.values,
    "Precioneto_real": y_test.values,
    "Precioneto_pred": y_pred
})
pred_fname = f"pronostico_holdout_{datetime.now().strftime('%Y%m%d_%H%M%S')}.csv"
pred_df.to_csv(pred_fname, index=False, encoding="utf-8")
print(f"\nArchivo con predicciones del holdout guardado como: {pred_fname}")

from google.colab import drive
drive.mount('/content/drive')

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
```



```
# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

# Cambia el PATH segUn tu archivo
PATH = "/content/drive/MyDrive/BATES_50KG_DIARIO_COMPLETO_FINAL.xlsx" # Reemplaza con la ruta de tu archivo
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y renombrar a canFinicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"^[^w\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

# Ajusta segUn las columnas de tu dataset
rename_map = {
    "fechadocumento": "fecha_documento", # Columna de fechas
    "precioneto": "precio_netto", # Variable objetivo
    "trm": "trm", # Variables exFigenas (ajusta segUn tu caso)
    # Agrega mFys columnas exFigenas si es necesario, ej: "klabin": "klabin", "mondi": "mondi"
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) VerificaciUn de columnas =====
required = ["fecha_documento", "precio_netto", "trm"] # Ajusta segUn tus variables exFigenas
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"
df = df[required].copy()

# ===== 4) Tipos, orden temporal y seFtales calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
for c in required[1:]: # Convierte a numFrico todas las columnas excepto la fecha
    df[c] = pd.to_numeric(df[c], errors="coerce")

# Orden y limpieza mF-nima
df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# Relleno simple de exFigenas si hay huecos
for c in required[2:]: # Rellena solo variables exFigenas
    if df[c].isna().any():
        df[c] = df[c].ffill().bfill()

# SeFtales internas (calendario)
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_netto"
num_feats = ["trm", "sin_month", "cos_month", "sin_dow", "cos_dow"] # Ajusta segUn tus variables exFigenas

# ===== 6) Escalado =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

X_train_tab = df_train[num_feats].to_numpy(dtype=np.float32)
X_val_tab = df_val[num_feats].to_numpy(dtype=np.float32)
X_test_tab = df_test[num_feats].to_numpy(dtype=np.float32)

num_scaler = StandardScaler().fit(X_train_tab)
X_train_tab = num_scaler.transform(X_train_tab)
X_val_tab = num_scaler.transform(X_val_tab)
X_test_tab = num_scaler.transform(X_test_tab)

# ===== 7) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1 # Ajusta LOOKBACK segUn ACF/estacionalidad

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]), np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]), np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 8) Modelo LSTM (128n†'64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 9) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)
```

```
# ===== 10) Evaluaci3n y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_tr = model.predict(Xtr, verbose=0).ravel()
pred_va = model.predict(Xva, verbose=0).ravel()
pred_te = model.predict(Xte, verbose=0).ravel()

ytr_true = inv_y(ytr.ravel()); pred_tr_u = inv_y(pred_tr)
yva_true = inv_y(yva.ravel()); pred_va_u = inv_y(pred_va)
yte_true = inv_y(yte.ravel()); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACI3N ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 11) Gr1ficas r1pidas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de p1rdida"); plt.xlabel("T%poca"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

# ===== 12) Gr1ficos con fechas alineadas =====
fechas_tr = df_train["fecha_documento"].iloc[LOOKBACK:].to_numpy()
fechas_va = df_val["fecha_documento"].iloc[1:].to_numpy()
fechas_te = df_test["fecha_documento"].iloc[1:].to_numpy()

assert len(fechas_tr) == len(pred_tr_u), f"TRAIN: {len(fechas_tr)} vs {len(pred_tr_u)}"
assert len(fechas_va) == len(pred_va_u), f"VALID: {len(fechas_va)} vs {len(pred_va_u)}"
assert len(fechas_te) == len(pred_te_u), f"TEST: {len(fechas_te)} vs {len(pred_te_u)}"

fechas_pred = np.concatenate([fechas_tr, fechas_va, fechas_te])
pred_full = np.concatenate([pred_tr_u, pred_va_u, pred_te_u])
fechas_real = df["fecha_documento"].to_numpy()
real_full = df["precio_neto"].to_numpy()
year_min = pd.to_datetime(fechas_real.min()).year
year_max = pd.to_datetime(fechas_real.max()).year

plt.figure(figsize=(14,5))
plt.plot(fechas_real, real_full, label="Real", linewidth=2)
plt.plot(fechas_pred, pred_full, label="LSTM (pred.)", linestyle="--", linewidth=2)
plt.title(f"Precio neto (COP) B" Real vs LSTM {year_min}B" {year_max}")
plt.xlabel("Fecha"); plt.ylabel("Precio neto (COP)")
plt.grid(True, alpha=0.3); plt.legend(); plt.tight_layout(); plt.show()

# --- Panel complementario (cortes + m1tricas anotadas) ---
fig, ax = plt.subplots(figsize=(13,4.5))
ax.plot(df_train["fecha_documento"], df_train["precio_neto"], label="TRAIN (real)", alpha=0.8)
ax.plot(df_val["fecha_documento"], df_val["precio_neto"], label="VAL (real)", alpha=0.8)
ax.plot(df_test["fecha_documento"], df_test["precio_neto"], label="TEST (real)", alpha=0.8)
ax.plot(fechas_tr, pred_tr_u, "--", label="LSTM (Train)")
ax.plot(fechas_va, pred_va_u, "--", label="LSTM (Val)")
ax.plot(fechas_te, pred_te_u, "--", label="LSTM (Test)")

ax.axvline(df_val["fecha_documento"].iloc[0], ls="--", c="gray")
ax.axvline(df_test["fecha_documento"].iloc[0], ls="--", c="gray")
ax.text(df_val["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Validaci3n", va="top", ha="left")
ax.text(df_test["fecha_documento"].iloc[0], ax.get_ylim()[1], "Inicio Test", va="top", ha="left")

ax.set_title("LSTM B· Precio neto (COP) B" Real vs Predicho")
ax.set_ylabel("Precio neto (COP)")
ax.set_xlabel("Fecha")
ax.grid(True, alpha=0.3)
ax.legend(loc="best")

y_true_te = df_test["precio_neto"].iloc[1:].values
mae = mean_absolute_error(y_true_te, pred_te_u)
rmse = math.sqrt(mean_squared_error(y_true_te, pred_te_u))
def mase(ytr, yt, yp, m=1):
    ytr = np.asarray(ytr); scale = np.mean(np.abs(ytr[m:]-ytr[:-m]))
    return np.mean(np.abs(np.asarray(yt)-np.asarray(yp)))/(scale if scale!=0 else 1)
mase_val = mase(df_train["precio_neto"].values, y_true_te, pred_te_u, m=1)
txt = f"TEST B" MAE={mae:.1f} | RMSE={rmse:.1f} | sMAPE={smape(y_true_te, pred_te_u):.2f}% | MASE={mase_val:.2f}"
ax.annotate(txt, xy=(0.01, 0.02), xycoords="axes fraction", fontsize=9,
            bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="gray", alpha=0.7))
plt.tight_layout(); plt.show()

# --- Paridad (Real vs Predicho) en TEST ---
fig, ax = plt.subplots(figsize=(5,5))
ax.scatter(y_true_te, pred_te_u, s=15, alpha=0.7)
lims = [min(y_true_te.min(), pred_te_u.min()), max(y_true_te.max(), pred_te_u.max())]
ax.plot(lims, lims, 'k--', lw=1)
ax.set_xlim(lims); ax.set_ylim(lims)
ax.set_title("Paridad en Test")
ax.set_xlabel("Real (COP)"); ax.set_ylabel("Predicho (COP)")
ax.grid(True, alpha=0.3); plt.tight_layout(); plt.show()

# --- Residuos en TEST ---
res = y_true_te - pred_te_u
fig, ax = plt.subplots(1,2, figsize=(10,3.8))
ax[0].plot(res, lw=1); ax[0].axhline(0, color="k", ls="--", lw=0.8)
ax[0].set_title("Residuos (Test) vs. tiempo"); ax[0].set_xlabel("T1ndice (ventana)"); ax[0].set_ylabel("Error (COP)")
ax[0].grid(True, alpha=0.3)
ax[1].hist(res, bins=30, alpha=0.85)
ax[1].set_title("Histograma de residuos (Test)"); ax[1].set_xlabel("Error (COP)")
ax[1].grid(True, alpha=0.3)
plt.tight_layout(); plt.show()

# ===== 13) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_model.keras"))
import joblib
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# =====
#@title LSTM BASE (con TRM y otros): BATES_50KG_DIARIO_COMPLETO_FINAL.xlsx
# Vars: Proveedor/Centro suministrador, Textobreve, Material,
#       Precioneto, Fechadocumento, TRM, Klabin, Mondi, Smurfit
```



```
# Marco teFirico: normalizaciFin, one-hot con rarefacciFin, split temporal,
# ventanas deslizantes, LSTM (128B+64) + LayerNorm, regularizaciFin y callbacks.
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KG_DIARIO_COMPLETO_FINAL.xlsx"
df = pd.read_excel(PATH)

# ===== 2) Normalizar nombres y mapear a canFinicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"^\w\s/", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "proveedor/centro_suministrador": "proveedor",
    "proveedor_centro_suministrador": "proveedor",
    "textobreve": "texto_breve",
    "precioneto": "precio_netto",
    "fechadocumento": "fecha_documento",
    "material": "material",
    "trm": "trm",
    "klabin": "klabin",
    "mondi": "mondi",
    "smurfit": "smurfit",
    # Mes y Año se ignoran ya que se derivan de la fecha
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) Filtrar estrictamente columnas base =====
required = ["proveedor", "texto_breve", "material", "precio_netto", "fecha_documento", "trm", "klabin", "mondi", "smurfit"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"

df = df[required].copy()

# ===== 4) Parseo de fecha, orden temporal y features de calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# Señales de estacionalidad (opcionales pero Petiles y no exFigenas)
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

TARGET = "precio_netto"
cat_cols = ["proveedor", "texto_breve", "material"]
num_base = ["trm", "klabin", "mondi", "smurfit", "sin_month", "cos_month", "sin_dow", "cos_dow"] # numFóricas internas incluyendo nuevas

# ===== 6) RarefacciFin top-K para categFíricas =====
TOP_K = 50
def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")

top_sets = {c: topk_set(df_train[c], TOP_K) for c in cat_cols}
for c in cat_cols:
    df_train[c] = to_topk(df_train[c], top_sets[c])
    df_val[c] = to_topk(df_val[c], top_sets[c])
    df_test[c] = to_topk(df_test[c], top_sets[c])

# ===== 7) Escalado: X con StandardScaler, y con MinMax =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

# OneHot categFíricas + StandardScaler numFóricas
ohe = OneHotEncoder(handle_unknown="ignore")

# Prepara matrices tabulares para cada split
def make_tab(df_part):
    X_num = df_part[num_base].to_numpy(dtype=np.float32)
    X_cat = ohe.fit_transform(df_train[cat_cols]).toarray() if df_part is df_train else ohe.transform(df_part[cat_cols]).toarray()
    return np.hstack([X_num, X_cat]).astype(np.float32)

# Ajuste del OHE sFílo con TRAIN
ohe.fit(df_train[cat_cols])
X_train_tab = make_tab(df_train)
X_val_tab = make_tab(df_val)
X_test_tab = make_tab(df_test)

# StandardScaler para numFóricas + mantener OHE ya binario
num_scaler = StandardScaler().fit(df_train[num_base].to_numpy(dtype=np.float32))
def scale_tab(X_tab):
    n_num = len(num_base)
    Xn = X_tab.copy()
    Xn[:, :n_num] = num_scaler.transform(Xn[:, :n_num])
    return Xn
```

```
X_train_tab = scale_tab(X_train_tab)
X_val_tab = scale_tab(X_val_tab)
X_test_tab = scale_tab(X_test_tab)

# ===== 8) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 9) Modelo LSTM (128->64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x = LayerNormalization()(x)
x = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x = LayerNormalization()(x)
x = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 10) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 11) Evaluaci3n y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva).ravel()
pred_te = model.predict(Xte).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va); pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}
print("== VALIDACI3N ==", metrics_val)
print("== TEST ==", metrics_test)

# ===== 12) Gr1ficas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de p1rdida"); plt.xlabel("T3poca"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real")
plt.plot(pred_te_u, label="Predicci3n")
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("T3ndice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40)
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 13) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_base_precio_netto_con_trm.keras"))
import joblib
joblib.dump(ohe, os.path.join(SAVE_DIR, "ohe_categorias.joblib"))
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
print("Artefactos guardados en:", SAVE_DIR)

# ===== 14) Notas de tuning r1pido =====
# - Ajusta LOOKBACK (p.ej. 28, 60, 90) seg1en autocorrelaci3n/estacionalidad.
# - Si hay alta cardinalidad en categor1icas: baja TOP_K o filtra categor1as muy raras.
# - Sobreajuste: aumenta dropout/recurrent_dropout o L2; reduce unidades.
# - Infraajuste: aumenta unidades/capas o LOOKBACK.

# =====
#@title LSTM CON VARIABLES EXT1GENAS: BATES_50KG_DIARIO_COMPLETO_FINAL.xlsx
# Vars: Fechadocumento, Proveedor/Centro suministrador, Textobreve, Material,
# Precioneto (TARGET), TRM, Klabin, Mondi, Smurfit, Mes, APto
# Marco te1rico: normalizaci3n, one-hot con rarefaci3n, split temporal,
# ventanas deslizantes, LSTM (128+64) + LayerNorm, regularizaci3n y callbacks.
# NUEVAS VARIABLES EXT1GENAS: TRM y precios competencia (Klabin, Mondi, Smurfit)
# =====

import os, re, math, random, warnings
warnings.filterwarnings("ignore")
import numpy as np
```

```
import pandas as pd
import matplotlib.pyplot as plt

SEED = 42
random.seed(SEED); np.random.seed(SEED)

import tensorflow as tf
tf.random.set_seed(SEED)
from tensorflow.keras import layers, models, callbacks, regularizers, optimizers
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization

from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder
from sklearn.metrics import mean_absolute_error, mean_squared_error

# ===== 1) Cargar datos =====
from google.colab import drive
drive.mount("/content/drive")

PATH = "/content/drive/MyDrive/BATES_50KG_DIARIO_COMPLETO_FINAL.xlsx"
df = pd.read_excel(PATH)

print(f"Dimensiones originales: {df.shape}")
print(f"Columnas: {df.columns.tolist()}")

# ===== 2) Normalizar nombres y mapear a canFinicos =====
def slugify(s):
    s = s.strip()
    s = re.sub(r"^[^w\s/]", "", s, flags=re.UNICODE)
    s = s.replace("/", "_")
    s = re.sub(r"\s+", "_", s)
    return s.lower()

df.columns = [slugify(c) for c in df.columns]

rename_map = {
    "proveedor/centro_suministrador": "proveedor",
    "proveedor_centro_suministrador": "proveedor",
    "textobreve": "texto_breve",
    "precioneto": "precio_netto",
    "fecha_documento": "fecha_documento",
    "fechadocumento": "fecha_documento",
    "material": "material",
    "mes": "mes",
    "año": "anio",
    "ano": "anio",
    "trm": "trm",
    "klabin": "klabin",
    "mondi": "mondi",
    "smurfit": "smurfit"
}
df = df.rename(columns={k:v for k,v in rename_map.items() if k in df.columns})

# ===== 3) Filtrar columnas requeridas =====
required = ["fecha_documento", "proveedor", "texto_breve", "material",
            "precio_netto", "trm", "klabin", "mondi", "smurfit"]
missing = [c for c in required if c not in df.columns]
assert not missing, f"Faltan columnas requeridas: {missing}"

df = df[required].copy()

# ===== 4) Parseo de fecha, orden temporal y features de calendario =====
df["fecha_documento"] = pd.to_datetime(df["fecha_documento"], errors="coerce", infer_datetime_format=True)
df = df.dropna(subset=["fecha_documento", "precio_netto"]).sort_values("fecha_documento").reset_index(drop=True)

# SeTales de estacionalidad
df["dow"] = df["fecha_documento"].dt.dayofweek
df["month"] = df["fecha_documento"].dt.month
df["sin_month"] = np.sin(2*np.pi*df["month"]/12)
df["cos_month"] = np.cos(2*np.pi*df["month"]/12)
df["sin_dow"] = np.sin(2*np.pi*df["dow"]/7)
df["cos_dow"] = np.cos(2*np.pi*df["dow"]/7)

# Convertir variables exTigenas a float (manejar posibles NaN)
exog_vars = ["trm", "klabin", "mondi", "smurfit"]
for var in exog_vars:
    df[var] = pd.to_numeric(df[var], errors="coerce")

# InterpolaciFin lineal para NaN en variables exTigenas
df[exog_vars] = df[exog_vars].interpolate(method='linear', limit_direction='both')
# Rellenar cualquier NaN restante con forward/backward fill
df[exog_vars] = df[exog_vars].fillna(method='ffill').fillna(method='bfill')

print(f"Registros despuTos de limpieza: {len(df)}")
print(f"NaN restantes:\n{df[exog_vars].isnull().sum()}")

# ===== 5) Split temporal =====
train_ratio, val_ratio = 0.7, 0.15
n = len(df); i_train = int(n*train_ratio); i_val = int(n*(train_ratio+val_ratio))
df_train = df.iloc[:i_train].copy()
df_val = df.iloc[i_train:i_val].copy()
df_test = df.iloc[i_val:].copy()

print(f"Train: {len(df_train)} | Val: {len(df_val)} | Test: {len(df_test)}")

TARGET = "precio_netto"
cat_cols = ["proveedor", "texto_breve", "material"]
# IMPORTANTE: Ahora incluimos las variables exTigenas en num_base
num_base = ["sin_month", "cos_month", "sin_dow", "cos_dow",
            "trm", "klabin", "mondi", "smurfit"]

# ===== 6) RarefacciFin top-K para categTiricas =====
TOP_K = 50
def topk_set(s, k): return set(pd.Series(s).astype(str).value_counts().nlargest(k).index)
def to_topk(s, top): return pd.Series(s).astype(str).where(pd.Series(s).astype(str).isin(top), "__OTHER__")

top_sets = {c: topk_set(df_train[c], TOP_K) for c in cat_cols}
for c in cat_cols:
    df_train[c] = to_topk(df_train[c], top_sets[c])
    df_val[c] = to_topk(df_val[c], top_sets[c])
    df_test[c] = to_topk(df_test[c], top_sets[c])

# ===== 7) Escalado: X con StandardScaler, y con MinMax =====
y_scaler = MinMaxScaler()
y_train = y_scaler.fit_transform(df_train[[TARGET]])
y_val = y_scaler.transform(df_val[[TARGET]])
y_test = y_scaler.transform(df_test[[TARGET]])

# OneHot categTiricas + StandardScaler numTiricas
ohe = OneHotEncoder(handle_unknown="ignore")

# Prepara matrices tabulares para cada split
def make_tab(df_part, is_train=False):
    X_num = df_part[num_base].to_numpy(dtype=np.float32)
```

```

    if is_train:
        X_cat = ohe.fit_transform(df_part[cat_cols]).toarray()
    else:
        X_cat = ohe.transform(df_part[cat_cols]).toarray()
    return np.hstack([X_num, X_cat]).astype(np.float32)

# Ajuste del OHE sFilo con TRAIN
X_train_tab = make_tab(df_train, is_train=True)
X_val_tab    = make_tab(df_val)
X_test_tab   = make_tab(df_test)

# StandardScaler para numFOricas (incluye exFigenas)
num_scaler = StandardScaler().fit(df_train[num_base].to_numpy(dtype=np.float32))
def scale_tab(X_tab):
    n_num = len(num_base)
    Xn = X_tab.copy()
    Xn[:, :n_num] = num_scaler.transform(Xn[:, :n_num])
    return Xn

X_train_tab = scale_tab(X_train_tab)
X_val_tab    = scale_tab(X_val_tab)
X_test_tab   = scale_tab(X_test_tab)

print(f"Features totales: {X_train_tab.shape[1]} (num: {len(num_base)}, cat_ohe: {X_train_tab.shape[1]-len(num_base)})")

# ===== 8) Ventanas deslizantes (many-to-one) =====
LOOKBACK, HORIZON = 60, 1

def make_windows(tab_X, y, lookback=LOOKBACK, horizon=HORIZON):
    Xs, ys = [], []
    for t in range(lookback, len(tab_X)-horizon+1):
        Xs.append(tab_X[t-lookback:t, :])
        ys.append(y[t+horizon-1, 0])
    return np.array(Xs, dtype=np.float32), np.array(ys, dtype=np.float32).reshape(-1,1)

Xtr, ytr = make_windows(X_train_tab, y_train, LOOKBACK, HORIZON)
# concatenar cola de train para conservar contexto en val/test
Xva, yva = make_windows(np.vstack([X_train_tab[-LOOKBACK+1:], X_val_tab]),
                        np.vstack([y_train[-LOOKBACK+1:], y_val]), LOOKBACK, HORIZON)
Xte, yte = make_windows(np.vstack([X_val_tab[-LOOKBACK+1:], X_test_tab]),
                        np.vstack([y_val[-LOOKBACK+1:], y_test]), LOOKBACK, HORIZON)

print(f"Shapes -> Xtr:{Xtr.shape} Xva:{Xva.shape} Xte:{Xte.shape} (features={Xtr.shape[-1]})")

# ===== 9) Modelo LSTM (128->64) =====
inp = Input(shape=(LOOKBACK, Xtr.shape[-1]))
x   = LSTM(128, return_sequences=True, dropout=0.2, recurrent_dropout=0.1)(inp)
x   = LayerNormalization()(x)
x   = LSTM(64, return_sequences=False, dropout=0.2, recurrent_dropout=0.1)(x)
x   = LayerNormalization()(x)
x   = Dense(64, activation="relu", kernel_regularizer=regularizers.l2(1e-4))(x)
x   = Dropout(0.2)(x)
out = Dense(1, activation="linear")(x)

model = models.Model(inp, out)
opt = optimizers.Adam(learning_rate=1e-3, clipnorm=1.0)
model.compile(optimizer=opt, loss="mse", metrics=["mae"])
model.summary()

# ===== 10) Entrenamiento =====
es = callbacks.EarlyStopping(monitor="val_loss", patience=15, restore_best_weights=True)
rlr = callbacks.ReduceLROnPlateau(monitor="val_loss", factor=0.5, patience=7, min_lr=1e-5, verbose=1)

history = model.fit(
    Xtr, ytr,
    validation_data=(Xva, yva),
    epochs=200,
    batch_size=64,
    callbacks=[es, rlr],
    verbose=1
)

# ===== 11) EvaluaciFon y desescalado =====
def inv_y(y_scaled): return y_scaler.inverse_transform(y_scaled.reshape(-1,1)).ravel()
pred_va = model.predict(Xva).ravel()
pred_te = model.predict(Xte).ravel()

yva_true = inv_y(yva.ravel()); yte_true = inv_y(yte.ravel())
pred_va_u = inv_y(pred_va);      pred_te_u = inv_y(pred_te)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))
def smape(a,f):
    a = np.array(a); f = np.array(f)
    d = (np.abs(a)+np.abs(f))/2; d[d==0]=1e-8
    return np.mean(np.abs(a-f)/d)*100

metrics_val = {"VAL_MAE": mean_absolute_error(yva_true, pred_va_u),
               "VAL_RMSE": rmse(yva_true, pred_va_u),
               "VAL_sMAPE(%)": smape(yva_true, pred_va_u)}
metrics_test = {"TEST_MAE": mean_absolute_error(yte_true, pred_te_u),
                "TEST_RMSE": rmse(yte_true, pred_te_u),
                "TEST_sMAPE(%)": smape(yte_true, pred_te_u)}

print("\n" + "*"50)
print("== VALIDACIOn ==", metrics_val)
print("== TEST ==", metrics_test)
print("=="50 + "\n")

# ===== 12) GrFficas =====
plt.figure(figsize=(8,4))
plt.plot(history.history["loss"], label="train MSE")
plt.plot(history.history["val_loss"], label="val MSE")
plt.title("Curva de pFrdida"); plt.xlabel("Fpoca"); plt.ylabel("MSE")
plt.legend(); plt.grid(True); plt.show()

plt.figure(figsize=(10,4))
plt.plot(yte_true, label="Real", alpha=0.7)
plt.plot(pred_te_u, label="PredicciFon", alpha=0.7)
plt.title("Precio neto - Test (real vs pred.)")
plt.xlabel("Fndice temporal (ventanas)"); plt.ylabel("Precio neto")
plt.legend(); plt.grid(True); plt.show()

res = yte_true - pred_te_u
plt.figure(figsize=(8,4))
plt.hist(res, bins=40, edgecolor='black')
plt.title("Residuos (Test)"); plt.xlabel("Error"); plt.ylabel("Frecuencia"); plt.grid(True); plt.show()

# ===== 13) Guardar artefactos =====
SAVE_DIR = "/content"
model.save(os.path.join(SAVE_DIR, "lstm_exogenas_precio_netto.keras"))
import joblib
joblib.dump(ohe, os.path.join(SAVE_DIR, "ohe_categorias.joblib"))
joblib.dump(num_scaler, os.path.join(SAVE_DIR, "scaler_numericas.joblib"))
joblib.dump(y_scaler, os.path.join(SAVE_DIR, "scaler_objetivo.joblib"))
```

```
joblib.dump(top_sets, os.path.join(SAVE_DIR, "top_sets_categorias.joblib"))
print("БЪ Artefactos guardados en:", SAVE_DIR)

# ===== 14) Notas de tuning y mejoras =====
print("""
NOTAS DE TUNING:
- Variables exFigenas incluidas: TRM, Klabin, Mondi, Smurfit
- Si TRM es muy volFtil, considera aFtiadir su cambio porcentual o diferencia
- Para precios competencia, podrF-as crear ratios: precio_neto/klabin, etc.
- LOOKBACK=60 asume 2 meses de contexto (datos diarios)
- Si hay fuerte estacionalidad mensual, considera LOOKBACK=90 o 120
- Sobreajuste: aumenta dropout/L2, reduce unidades LSTM
- Infraajuste: aumenta unidades/capas o LOOKBACK
""")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# --- ConfiguraciFon y Montaje de Google Drive ---

# Comentario: Montar Google Drive para acceder al archivo.
# Se le pedirF que autorice el acceso.
try:
    from google.colab import drive
    print("Montando Google Drive...")
    drive.mount('/content/drive')
except ImportError:
    print("No se pudo importar 'google.colab'. Asumiendo entorno local.")

# --- 1. Cargar el Dataset ---
# Ruta especificada en el prompt
FILE_PATH = '/content/drive/MyDrive/BATES_50KG_DIARIO_COMPLETO_FINAL.xlsx'

print(f"\nCargando dataset desde: {FILE_PATH}")
try:
    # Cargar el archivo Excel
    df = pd.read_excel(FILE_PATH)
    print("Dataset cargado exitosamente.")
except FileNotFoundError:
    print(f"ERROR: Archivo no encontrado en la ruta: {FILE_PATH}")
    # Salir si el archivo no existe
    # exit()

# Si el DataFrame estF vacF-o despuF de un error de carga, detenemos el script
if 'df' not in locals() or df.empty:
    print("No se pudo proceder sin datos. Terminando script.")
    # Si desea probar sin Colab/Drive, descomente y cargue un dummy DataFrame aquF-.
    # df = pd.DataFrame(...)
    # exit()

# --- 2. ExploraciFon y Limpieza Inicial de Datos ---

print("\n--- ExploraciFon de Datos ---")
print("Dimensiones del DataFrame:", df.shape)
print("\nPrimeras 5 filas:")
print(df.head())
print("\nTipos de datos y valores no nulos:")
df.info()

# 2.1. Manejo de Valores Nulos
# Comentario: Verificamos los nulos y los imputamos/eliminamos segFen la estrategia.
print("\nConteo de valores nulos por columna:")
print(df.isnull().sum())

# Estrategia de limpieza:
# - Eliminar filas con nulos en la variable objetivo 'Precioneto'.
df.dropna(subset=['Precioneto'], inplace=True)

# - Para variables predictoras (ej: TRM, Klabin, Mondi, Smurfit), imputar con la media o el valor anterior (forward-fill)
# ya que son series temporales. Usaremos forward-fill (ffill).
numeric_cols_to_fill = ['TRM', 'Klabin', 'Mondi', 'Smurfit']
df[numeric_cols_to_fill] = df[numeric_cols_to_fill].fillna(method='ffill')

# Si quedan nulos despuF de ffill (ej: al inicio de la serie), se rellenan con la media.
df[numeric_cols_to_fill] = df[numeric_cols_to_fill].fillna(df[numeric_cols_to_fill].mean())

print("\nConteo de valores nulos despuF de la limpieza:")
print(df.isnull().sum())
print("Filas despuF de la limpieza:", df.shape[0])

# --- 3. ConversiFon de Fecha y Ordenamiento ---

# Comentario: Asegurar que 'Fechadocumento' es un objeto datetime y ordenar.
df['Fechadocumento'] = pd.to_datetime(df['Fechadocumento'])
df.sort_values(by='Fechadocumento', inplace=True)
df.set_index('Fechadocumento', inplace=True)

print("\nDataset ordenado por 'Fechadocumento'.")

# --- 4. SelecciFon y Escalado de Variables ---

# Variables a usar: 'Precioneto' (objetivo) y las variables exFigenas
# Se incluyen todas las columnas relevantes, excepto las categFricas o redundantes.
FEATURES = ['Precioneto', 'TRM', 'Klabin', 'Mondi', 'Smurfit']
data = df[FEATURES].values # Convertir a array de numpy

# Comentario: Escalar los datos para mejorar la estabilidad y convergencia del modelo.
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# Nota: El escalador se entrena con *todos* los datos, pero esto es comFen en modelos
# de series de tiempo donde el rango de valores histFricos es conocido.

# --- 5. CreaciFon de Secuencias Temporales (SecuenciaciFon para LSTM) ---

# HiperparFmetro clave: 'look_back' define cuFntos pasos de tiempo anteriores se usarF
# para predecir el siguiente paso.
LOOK_BACK = 30 # Usar los Feltimos 30 dF-as para predecir el siguiente dF-a (30 timesteps)
TEST_SIZE = 0.2 # 20% para el conjunto de prueba
```



```
def create_sequences(data, look_back=1):
    """
    Funci n para crear conjuntos de datos X (secuencias de entrada) y Y (salida).
    X contendr  `look_back` pasos de tiempo, Y contendr  el siguiente paso de tiempo.
    """
    X, Y = [], []
    for i in range(len(data) - look_back):
        # Secuencia de entrada (X): los 'look_back' valores anteriores (todas las features)
        a = data[i:(i + look_back)]
        X.append(a)
        # Valor de salida (Y): el valor de 'Precioneto' en el siguiente paso de tiempo
        # La variable 'Precioneto' es la primera columna ( ndice 0)
        Y.append(data[i + look_back, 0])
    return np.array(X), np.array(Y)

# Crear las secuencias
X, Y = create_sequences(scaled_data, LOOK_BACK)

print(f"\nForma de las secuencias de entrada (X): {X.shape} (muestras, timesteps, features)")
print(f"Forma de la salida (Y): {Y.shape} (muestras)")

# Dividir en conjuntos de entrenamiento y prueba
# Comentario: En series temporales, la divisi n debe ser *secuencial*, no aleatoria.
train_size = int(len(X) * (1 - TEST_SIZE))
X_train, X_test = X[:train_size], X[train_size:]
Y_train, Y_test = Y[:train_size], Y[train_size:]

print(f"\nTama o de entrenamiento: {len(X_train)} muestras")
print(f"Tama o de prueba: {len(X_test)} muestras")

# --- 6. Definici n, Compilaci n y Entrenamiento del Modelo LSTM ---

# Par metros del modelo
LSTM_UNITS = 50
DROPOUT_RATE = 0.2
EPOCHS = 50 # N mero de  pocas de entrenamiento
BATCH_SIZE = 32

# Comentario: Definir el modelo LSTM secuencial.
model = Sequential()
# Primera capa LSTM: retorna secuencias (return_sequences=True) para la segunda capa LSTM.
# Input_shape: (timesteps, features)
model.add(LSTM(units=LSTM_UNITS, return_sequences=True, input_shape=(LOOK_BACK, X_train.shape[2])))
model.add(Dropout(DROPOUT_RATE))
# Segunda capa LSTM: no retorna secuencias para pasar la salida a la capa densa.
model.add(LSTM(units=LSTM_UNITS, return_sequences=False))
model.add(Dropout(DROPOUT_RATE))
# Capa de salida: una sola neurona para predecir 'Precioneto'
model.add(Dense(units=1))

# Compilaci n
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
print("\n--- Resumen del Modelo LSTM ---")
model.summary()

# Callbacks: Early Stopping para detener el entrenamiento si la p rdida no mejora.
early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1, restore_best_weights=True)

# Entrenamiento
print("\n--- Iniciando Entrenamiento del Modelo ---")
history = model.fit(
    X_train, Y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_split=0.1, # 10% del conjunto de entrenamiento para validaci n
    callbacks=[early_stop],
    verbose=1
)

# Visualizaci n de la p rdida de entrenamiento
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='P rdida de Entrenamiento (MSE)')
plt.plot(history.history['val_loss'], label='P rdida de Validaci n (MSE)')
plt.title('Progreso del Entrenamiento del Modelo LSTM')
plt.xlabel(' poca')
plt.ylabel('P rdida (MSE)')
plt.legend()
plt.show()

# --- 7. Realizar Predicciones y Visualizaci n ---

# Comentario: Realizar predicciones sobre el conjunto de prueba.
Y_pred_scaled = model.predict(X_test)

# Invertir el escalado
# Nota importante: El scaler fue entrenado con 5 columnas. Necesitamos crear un array
# temporal con ceros para las otras 4 columnas y poner la predicci n en la primera columna
# ( ndice 0, correspondiente a 'Precioneto') para poder usar `inverse_transform`.

# 7.1 Invertir el escalado de las Predicciones (Y_pred_scaled)
# Creamos un array temporal lleno de ceros con la forma (muestras, features)
# La forma es (len(Y_pred_scaled), num_features)
dummy_Y_pred = np.zeros((len(Y_pred_scaled), scaled_data.shape[1]))
# Colocamos las predicciones escaladas en la columna de 'Precioneto' ( ndice 0)
dummy_Y_pred[:, 0] = Y_pred_scaled.flatten()
# Invertimos la transformaci n
Y_pred = scaler.inverse_transform(dummy_Y_pred)[:, 0] # Tomamos solo la primera columna (Precioneto)

# 7.2 Invertir el escalado de los Valores Reales (Y_test)
# Hacemos lo mismo para los valores reales (Y_test)
dummy_Y_test = np.zeros((len(Y_test), scaled_data.shape[1]))
dummy_Y_test[:, 0] = Y_test.flatten()
Y_real = scaler.inverse_transform(dummy_Y_test)[:, 0] # Tomamos solo la primera columna (Precioneto)

# 7.3 Visualizaci n
# Obtener las fechas correspondientes a las predicciones para el eje X
dates = df.index[-len(Y_test):] # Fechas desde el inicio del conjunto de prueba

plt.figure(figsize=(14, 7))
plt.plot(dates, Y_real, label='Valor Real (Precioneto)', color='blue')
plt.plot(dates, Y_pred, label='Predicci n LSTM', color='red', linestyle='--')
plt.title('Predicci n de Precioneto vs. Valor Real')
plt.xlabel('Fecha')
plt.ylabel('Precioneto (Valor Original)')
plt.legend()
plt.grid(True)
plt.show()
```

```
# --- 8. Evaluaci3n del Modelo ---

# Comentario: Calcular las m3tricas de evaluaci3n.
mae = mean_absolute_error(Y_real, Y_pred)
rmse = np.sqrt(mean_squared_error(Y_real, Y_pred))
r2 = r2_score(Y_real, Y_pred)

print("\n--- M3tricas de Evaluaci3n del Modelo ---")
print(f"Error Absoluto Medio (MAE): {mae:,.4f}")
print(f"RaI-z del Error Cuadr3tico Medio (RMSE): {rmse:,.4f}")
print(f"Coficiente de Determinaci3n (RBI): {r2:,.4f}")
print("-" * 40)
print(f"Conclusi3n: El modelo LSTM entrenado tiene un RBI de {r2:,.4f} sobre el conjunto de prueba.")

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# --- Configuraci3n y Montaje de Google Drive ---

# Comentario: Montar Google Drive para acceder al archivo.
# Ejecutar esta celda y seguir las instrucciones para la autenticaci3n.
try:
    from google.colab import drive
    print("Montando Google Drive...")
    drive.mount('/content/drive')
except ImportError:
    print("No se pudo importar 'google.colab'. Asumiendo entorno local.")

# --- 1. Cargar el Dataset ---
# **Ruta actualizada** seg3n el requerimiento del usuario.
FILE_PATH = '/content/drive/MyDrive/BATES_50KG_DIARIO_TRM.xlsx'

print(f"\nCargando dataset desde: {FILE_PATH}")
try:
    # Cargar el archivo Excel
    df = pd.read_excel(FILE_PATH)
    print("Dataset cargado exitosamente.")
except FileNotFoundError:
    print(f"ERROR: Archivo no encontrado en la ruta: {FILE_PATH}")
    # Si el archivo no existe, el script no puede continuar.
    if 'df' not in locals() or df.empty:
        print("Terminando script debido a la falta de datos.")
        # Se detiene el script si no hay datos.
        exit()

# --- 2. Exploraci3n y Limpieza Inicial de Datos ---

print("\n--- Exploraci3n y Limpieza de Datos ---")
print("Dimensiones iniciales del DataFrame:", df.shape)

# 2.1. Manejo de Valores Nulos y Tipos
# Comentario: Verificamos los nulos. La variable objetivo ('Pricing') y la ex3gena ('TRM')
# deben estar limpias para la serie temporal.

# Renombrar columnas para evitar problemas con espacios o caracteres especiales si fuera necesario.
df.columns = df.columns.str.replace('[^A-Za-z0-9_]+', '', regex=True)

# Variables a utilizar:
TARGET_COL = 'Pricing'
EXOGENOUS_COL = 'TRM'
DATE_COL = 'FechaDocumento'

# 2.1.1. Conversi3n de Fecha y Ordenamiento (Se ejecuta primero para usar ffill)
df[DATE_COL] = pd.to_datetime(df[DATE_COL])
df.sort_values(by=DATE_COL, inplace=True)
df.set_index(DATE_COL, inplace=True)

# 2.1.2. Imputaci3n de nulos
# Eliminar filas donde falta el valor objetivo.
df.dropna(subset=[TARGET_COL], inplace=True)

# Para 'TRM', usar 'ffill' (forward-fill) para rellenar con el valor conocido m3s reciente.
df[EXOGENOUS_COL] = df[EXOGENOUS_COL].fillna(method='ffill')

# Si quedan nulos (ej: al inicio de la serie), rellenar con la media.
df[EXOGENOUS_COL] = df[EXOGENOUS_COL].fillna(df[EXOGENOUS_COL].mean())

print("\nConteo de nulos despu3s de la limpieza:")
print(df[[TARGET_COL, EXOGENOUS_COL]].isnull().sum())
print("Filas despu3s de la limpieza:", df.shape[0])

# --- 3. Selecci3n y Escalado de Variables Num3ricas ---

# Comentario: Seleccionar las variables para el modelo: objetivo y ex3gena.
FEATURES = [TARGET_COL, EXOGENOUS_COL]
data = df[FEATURES].values # Convertir a array de numpy (2 columnas)

# Comentario: Escalar los datos al rango [0, 1].
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

print(f"\nVariables seleccionadas para el modelo: {FEATURES}")
print(f"Forma de los datos escalados: {scaled_data.shape}")

# --- 4. Creaci3n de Secuencias Temporales (Secuenciaci3n para LSTM) ---

# Hiperpar3metro clave: `look_back` (n3mero de pasos de tiempo anteriores a usar)
LOOK_BACK = 60 # Usamos 60 pasos (ej: dos meses de datos diarios)
TEST_SIZE = 0.2 # 20% para el conjunto de prueba

def create_sequences(data, look_back=1):
    """
    Funci3n para crear conjuntos de datos X (secuencias de entrada) y Y (salida).
    X contendr3 `look_back` pasos de tiempo, Y contendr3 el valor de 'Pricing' en el siguiente paso.
    """
    X, Y = [], []
    # Iteramos hasta el pen3ltimo punto de la serie para asegurar que tenemos un valor Y
    for i in range(len(data) - look_back):
        # X: Secuencia de entrada de 'look_back' d3as, incluyendo todas las features.
        a = data[i:(i + look_back)]
        X.append(a)
        # Y: El valor objetivo 'Pricing' (3ndice 0) en el d3a siguiente.
```

```
        Y.append(data[i + look_back, 0])
    return np.array(X), np.array(Y)

# Crear las secuencias
X, Y = create_sequences(scaled_data, LOOK_BACK)

print(f"\nForma de X (muestras, timesteps, features): {X.shape}")
print(f"Forma de Y (muestras): {Y.shape}")

# División secuencial: los datos de prueba son el período más reciente.
train_size = int(len(X) * (1 - TEST_SIZE))
X_train, X_test = X[:train_size], X[train_size:]
Y_train, Y_test = Y[:train_size], Y[train_size:]

print(f"Tamaño de entrenamiento: {len(X_train)} muestras")
print(f"Tamaño de prueba: {len(X_test)} muestras")

# --- 5. Definición, Compilación y Entrenamiento del Modelo LSTM ---

# Parámetros del modelo
LSTM_UNITS = 64
DROPOUT_RATE = 0.2
EPOCHS = 100
BATCH_SIZE = 32

# Comentario: Definir la arquitectura LSTM.
model = Sequential()
# Primera capa LSTM con input_shape (timesteps, features)
model.add(LSTM(units=LSTM_UNITS, return_sequences=True, input_shape=(LOOK_BACK, X_train.shape[2])))
model.add(Dropout(DROPOUT_RATE))
# Segunda capa LSTM (opcional, para mayor complejidad)
model.add(LSTM(units=LSTM_UNITS, return_sequences=False))
model.add(Dropout(DROPOUT_RATE))
# Capa de salida: una sola neurona para la predicción de un valor continuo.
model.add(Dense(units=1))

# Compilación: Optimizador Adam y función de pérdida MSE para regresión.
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
print("\n--- Resumen del Modelo LSTM ---")
model.summary()

# Callbacks: Early Stopping para evitar el sobreajuste.
early_stop = EarlyStopping(monitor='val_loss', patience=15, verbose=1, restore_best_weights=True)

# Entrenamiento, mostrando el progreso
print("\n--- Iniciando Entrenamiento del Modelo ---")
history = model.fit(
    X_train, Y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_split=0.15, # 15% del entrenamiento para validación interna
    callbacks=[early_stop],
    verbose=1 # Mostrar el progreso por época
)

# Visualización del progreso del entrenamiento
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='Pérdida de Entrenamiento (MSE)')
plt.plot(history.history['val_loss'], label='Pérdida de Validación (MSE)')
plt.title('Progreso del Entrenamiento del Modelo LSTM')
plt.xlabel('Época')
plt.ylabel('Pérdida (MSE)')
plt.legend()
plt.show()

# --- 6. Realizar Predicciones y Visualización ---

# Predicción sobre el conjunto de prueba
Y_pred_scaled = model.predict(X_test)

# Comentario: Invertir el escalado de las predicciones y los valores reales.
# Necesitamos la forma original (muestras, 2 features) para usar `inverse_transform`.

# 6.1 Invertir el escalado de las Predicciones (Y_pred_scaled)
# Crear un array temporal con ceros para la columna 'TRM'
dummy_Y_pred = np.zeros((len(Y_pred_scaled), scaled_data.shape[1]))
# Colocar las predicciones escaladas en la columna de 'Precioneto' (Índice 0)
dummy_Y_pred[:, 0] = Y_pred_scaled.flatten()
# Invertir la transformación y tomar solo la columna de 'Precioneto'
Y_pred = scaler.inverse_transform(dummy_Y_pred)[:, 0]

# 6.2 Invertir el escalado de los Valores Reales (Y_test)
dummy_Y_test = np.zeros((len(Y_test), scaled_data.shape[1]))
dummy_Y_test[:, 0] = Y_test.flatten()
Y_real = scaler.inverse_transform(dummy_Y_test)[:, 0]

# 6.3 Visualización
# Obtener las fechas correspondientes a las predicciones para el eje X
# Las fechas inician después de la ventana LOOK_BACK
dates = df.index[-len(Y_test):]

plt.figure(figsize=(16, 8))
plt.plot(dates, Y_real, label='Valor Real (Precioneto)', color='blue', linewidth=2)
plt.plot(dates, Y_pred, label='Predicción LSTM', color='red', linestyle='--', linewidth=1.5)
plt.title('Predicción de Precioneto vs. Valor Real en el Conjunto de Prueba')
plt.xlabel('Fecha')
plt.ylabel('Precioneto (Valor Original)')
plt.legend()
plt.grid(True, alpha=0.6)
plt.tight_layout()
plt.show()

# --- 7. Evaluación del Modelo ---

# Comentario: Calcular las métricas.
mae = mean_absolute_error(Y_real, Y_pred)
rmse = np.sqrt(mean_squared_error(Y_real, Y_pred))
r2 = r2_score(Y_real, Y_pred)

print("\n--- Métricas de Evaluación del Modelo ---")
print(f"Error Absoluto Medio (MAE): {mae:.4f}")
print(f"Raíz del Error Cuadrático Medio (RMSE): {rmse:.4f}")
print(f"Coefficiente de Determinación (RBI): {r2:.4f}")
print("-" * 50)
print(f"Interpretación: El MAE indica que, en promedio, la predicción se desvía del valor real en {mae:.4f} unidades monetarias.")
```



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# --- Configuraci3n y Montaje de Google Drive ---

# Comentario: Montar Google Drive para acceder al archivo.
# Se le pedir3 que autorice el acceso.
try:
    from google.colab import drive
    print("Montando Google Drive...")
    drive.mount('/content/drive')
except ImportError:
    print("No se pudo importar 'google.colab'. Asumiendo entorno local.")

# --- 1. Cargar el Dataset ---
# Ruta especificada en el prompt
FILE_PATH = '/content/drive/MyDrive/BATES_50KG_DIARIO_COMPLETO_FINAL.xlsx'

print(f"\nCargando dataset desde: {FILE_PATH}")
try:
    # Cargar el archivo Excel
    df = pd.read_excel(FILE_PATH)
    print("Dataset cargado exitosamente.")
except FileNotFoundError:
    print(f"ERROR: Archivo no encontrado en la ruta: {FILE_PATH}")
    # Salir si el archivo no existe
    # exit()

# Si el DataFrame est3 vac3o despu3s de un error de carga, detenemos el script
if 'df' not in locals() or df.empty:
    print("No se pudo proceder sin datos. Terminando script.")
    # Si desea probar sin Colab/Drive, descomente y cargue un dummy DataFrame aqu3-.
    # df = pd.DataFrame(...)
    # exit()

# --- 2. Exploraci3n y Limpieza Inicial de Datos ---

print("\n--- Exploraci3n de Datos ---")
print("Dimensiones del DataFrame:", df.shape)
print("\nPrimeras 5 filas:")
print(df.head())
print("\nTipos de datos y valores no nulos:")
df.info()

# 2.1. Manejo de Valores Nulos
# Comentario: Verificamos los nulos y los imputamos/eliminamos seg3n la estrategia.
print("\nConteo de valores nulos por columna:")
print(df.isnull().sum())

# Estrategia de limpieza:
# - Eliminar filas con nulos en la variable objetivo 'Precioneto'.
df.dropna(subset=['Precioneto'], inplace=True)

# - Para variables predictoras (ej: TRM, Klabin, Mond3, Smurfit), imputar con la media o el valor anterior (forward-fill)
# ya que son series temporales. Usaremos forward-fill (ffill).
numeric_cols_to_fill = ['TRM', 'Klabin', 'Mond3', 'Smurfit']
df[numeric_cols_to_fill] = df[numeric_cols_to_fill].fillna(method='ffill')

# Si quedan nulos despu3s de ffill (ej: al inicio de la serie), se rellenan con la media.
df[numeric_cols_to_fill] = df[numeric_cols_to_fill].fillna(df[numeric_cols_to_fill].mean())

print("\nConteo de valores nulos despu3s de la limpieza:")
print(df.isnull().sum())
print("Filas despu3s de la limpieza:", df.shape[0])

# --- 3. Conversi3n de Fecha y Ordenamiento ---

# Comentario: Asegurar que 'Fechadocumento' es un objeto datetime y ordenar.
df['Fechadocumento'] = pd.to_datetime(df['Fechadocumento'])
df.sort_values(by='Fechadocumento', inplace=True)
df.set_index('Fechadocumento', inplace=True)

print("\nDataset ordenado por 'Fechadocumento'.")

# --- 4. Selecci3n y Escalado de Variables ---

# Variables a usar: 'Precioneto' (objetivo) y las variables ex3genas
# Se incluyen todas las columnas relevantes, excepto las categ3ricas o redundantes.
FEATURES = ['Precioneto', 'TRM', 'Klabin', 'Mond3', 'Smurfit']
data = df[FEATURES].values # Convertir a array de numpy

# Comentario: Escalar los datos para mejorar la estabilidad y convergencia del modelo.
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# Nota: El escalador se entrena con *todos* los datos, pero esto es com3n en modelos
# de series de tiempo donde el rango de valores hist3ricos es conocido.

# --- 5. Creaci3n de Secuencias Temporales (Secuenciaci3n para LSTM) ---

# Hiperpar3metro clave: 'look_back' define cu3ntos pasos de tiempo anteriores se usar3n
# para predecir el siguiente paso.
LOOK_BACK = 30 # Usar los 30 d3as para predecir el siguiente d3a (30 timesteps)
TEST_SIZE = 0.2 # 20% para el conjunto de prueba

def create_sequences(data, look_back=1):
    """
    Funci3n para crear conjuntos de datos X (secuencias de entrada) y Y (salida).
    X contendr3 `look_back` pasos de tiempo, Y contendr3 el siguiente paso de tiempo.
    """
    X, Y = [], []
    for i in range(len(data) - look_back):
        # Secuencia de entrada (X): los 'look_back' valores anteriores (todas las features)
        a = data[i:(i + look_back)]
        X.append(a)
        # Valor de salida (Y): el valor de 'Precioneto' en el siguiente paso de tiempo
        # La variable 'Precioneto' es la primera columna (3ndice 0)
        Y.append(data[i + look_back, 0])
    return np.array(X), np.array(Y)

# Crear las secuencias
X, Y = create_sequences(scaled_data, LOOK_BACK)
```

```
print(f"\nForma de las secuencias de entrada (X): {X.shape} (muestras, timesteps, features)")
print(f"Forma de la salida (Y): {Y.shape} (muestras)")

# Dividir en conjuntos de entrenamiento y prueba
# Comentario: En series temporales, la divisi n debe ser *secuencial*, no aleatoria.
train_size = int(len(X) * (1 - TEST_SIZE))
X_train, X_test = X[:train_size], X[train_size:]
Y_train, Y_test = Y[:train_size], Y[train_size:]

print(f"\nTama o de entrenamiento: {len(X_train)} muestras")
print(f"Tama o de prueba: {len(X_test)} muestras")

# --- 6. Definici n, Compilaci n y Entrenamiento del Modelo LSTM ---

# Par metros del modelo
LSTM_UNITS = 50
DROPOUT_RATE = 0.2
EPOCHS = 50 # N mero de  pocas de entrenamiento
BATCH_SIZE = 32

# Comentario: Definir el modelo LSTM secuencial.
model = Sequential()
# Primera capa LSTM: retorna secuencias (return_sequences=True) para la segunda capa LSTM.
# Input_shape: (timesteps, features)
model.add(LSTM(units=LSTM_UNITS, return_sequences=True, input_shape=(LOOK_BACK, X_train.shape[2])))
model.add(Dropout(DROPOUT_RATE))
# Segunda capa LSTM: no retorna secuencias para pasar la salida a la capa densa.
model.add(LSTM(units=LSTM_UNITS, return_sequences=False))
model.add(Dropout(DROPOUT_RATE))
# Capa de salida: una sola neurona para predecir 'Precioneto'
model.add(Dense(units=1))

# Compilaci n
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
print("\n--- Resumen del Modelo LSTM ---")
model.summary()

# Callbacks: Early Stopping para detener el entrenamiento si la p rdida no mejora.
early_stop = EarlyStopping(monitor='val_loss', patience=10, verbose=1, restore_best_weights=True)

# Entrenamiento
print("\n--- Iniciando Entrenamiento del Modelo ---")
history = model.fit(
    X_train, Y_train,
    epochs=EPOCHS,
    batch_size=BATCH_SIZE,
    validation_split=0.1, # 10% del conjunto de entrenamiento para validaci n
    callbacks=[early_stop],
    verbose=1
)

# Visualizaci n de la p rdida de entrenamiento
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='P rdida de Entrenamiento (MSE)')
plt.plot(history.history['val_loss'], label='P rdida de Validaci n (MSE)')
plt.title('Progreso del Entrenamiento del Modelo LSTM')
plt.xlabel(' poca')
plt.ylabel('P rdida (MSE)')
plt.legend()
plt.show()

# --- 7. Realizar Predicciones y Visualizaci n ---

# Comentario: Realizar predicciones sobre el conjunto de prueba.
Y_pred_scaled = model.predict(X_test)

# Invertir el escalado
# Nota importante: El scaler fue entrenado con 5 columnas. Necesitamos crear un array
# temporal con ceros para las otras 4 columnas y poner la predicci n en la primera columna
# ( ndice 0, correspondiente a 'Precioneto') para poder usar `inverse_transform`.

# 7.1 Invertir el escalado de las Predicciones (Y_pred_scaled)
# Creamos un array temporal lleno de ceros con la forma (muestras, features)
# La forma es (len(Y_pred_scaled), num_features)
dummy_Y_pred = np.zeros((len(Y_pred_scaled), scaled_data.shape[1]))
# Colocamos las predicciones escaladas en la columna de 'Precioneto' ( ndice 0)
dummy_Y_pred[:, 0] = Y_pred_scaled.flatten()
# Invertimos la transformaci n
Y_pred = scaler.inverse_transform(dummy_Y_pred)[:, 0] # Tomamos solo la primera columna (Precioneto)

# 7.2 Invertir el escalado de los Valores Reales (Y_test)
# Hacemos lo mismo para los valores reales (Y_test)
dummy_Y_test = np.zeros((len(Y_test), scaled_data.shape[1]))
dummy_Y_test[:, 0] = Y_test.flatten()
Y_real = scaler.inverse_transform(dummy_Y_test)[:, 0] # Tomamos solo la primera columna (Precioneto)

# 7.3 Visualizaci n
# Obtener las fechas correspondientes a las predicciones para el eje X
dates = df.index[-len(Y_test):] # Fechas desde el inicio del conjunto de prueba

plt.figure(figsize=(14, 7))
plt.plot(dates, Y_real, label='Valor Real (Precioneto)', color='blue')
plt.plot(dates, Y_pred, label='Predicci n LSTM', color='red', linestyle='--')
plt.title('Predicci n de Precioneto vs. Valor Real')
plt.xlabel('Fecha')
plt.ylabel('Precioneto (Valor Original)')
plt.legend()
plt.grid(True)
plt.show()

# --- 8. Evaluaci n del Modelo ---

# Comentario: Calcular las m tricas de evaluaci n.
mae = mean_absolute_error(Y_real, Y_pred)
rmse = np.sqrt(mean_squared_error(Y_real, Y_pred))
r2 = r2_score(Y_real, Y_pred)

print("\n--- M tricas de Evaluaci n del Modelo ---")
print(f"Error Absoluto Medio (MAE): {mae:.4f}")
print(f"Ra z del Error Cuadr tico Medio (RMSE): {rmse:.4f}")
print(f"Coeficiente de Determinaci n (RBI): {r2:.4f}")
print("-" * 40)
print(f"Conclusi n: El modelo LSTM entrenado tiene un RBI de {r2:.4f} sobre el conjunto de prueba.")
```