**# 1️⃣ ¿Por qué usaste una ventana de 16 semanas y no otra cantidad?**

ventana = 16

# Respuesta:

# En series de tiempo, la ventana define cuántos datos pasados se usan para predecir el futuro.

# Si la ventana es muy pequeña (ej. 4 semanas), el modelo es demasiado sensible a variaciones recientes,

# lo que puede causar predicciones inestables.

# Si la ventana es muy grande (ej. 52 semanas), se diluyen patrones recientes porque se mezclan con datos antiguos.

# Probamos diferentes tamaños y 16 semanas capturó tendencias sin volverse demasiado volátil.

**# 2️⃣ ¿Por qué agregaste una capa Conv1D antes de LSTM?**

x = Conv1D(filters=128, kernel\_size=3, activation='relu', padding='same')(feats\_input)

# Respuesta:

# Conv1D actúa como un filtro para detectar patrones locales en la serie de tiempo antes de pasarlo a LSTM.

# Esto es útil porque puede identificar tendencias en las ventas de manera más eficiente antes de aprender

# patrones a largo plazo.

# Ejemplo: Si en una ciudad las ventas aumentan después de una festividad cada año, Conv1D detectará esa subida local.

**# 3️⃣ ¿Cómo manejaste el overfitting?**

x = LSTM(128, return\_sequences=True, dropout=0.15)(x)

fusion = Dropout(0.25)(fusion)

# Respuesta:

# Overfitting ocurre cuando el modelo memoriza los datos en vez de generalizar patrones.

# Para evitarlo usamos:

# - Dropout en LSTM (0.15): Desactiva aleatoriamente el 15% de neuronas en cada paso de entrenamiento,

# forzando al modelo a no depender de conexiones específicas.

# - Dropout en la capa densa (0.25): Aplica la misma idea pero en la última parte del modelo.

# - EarlyStopping y ReduceLROnPlateau: Detienen el entrenamiento si la pérdida no mejora para evitar sobreajuste.

**# 4️⃣ ¿Por qué usaste StandardScaler y no MinMaxScaler?**

scaler = StandardScaler()

df\_weekly[features] = scaler.fit\_transform(df\_weekly[features])

# Respuesta:

# StandardScaler convierte los datos a una distribución con media 0 y desviación estándar 1.

# MinMaxScaler reescala los datos entre un mínimo y un máximo fijo (ej. 0 a 1).

# Como las ventas varían mucho por ciudad, StandardScaler mantiene la relación entre valores sin comprimir demasiado.

**# 5️⃣ ¿Por qué usaste log1p en lugar de simplemente log?**

df\_weekly['qty\_log'] = np.log1p(df\_weekly['qty'])

# Respuesta:

# log1p(x) = log(x + 1), lo cual evita problemas cuando qty = 0 (porque log(0) es indefinido).

# Esto es clave porque algunas ciudades tienen semanas sin ventas. Si hubiéramos usado solo log(x),

# habríamos tenido errores en esos casos.

**# 6️⃣ ¿Por qué usaste winsorize en la cantidad de ventas?**

df\_weekly['qty'] = winsorize(df\_weekly['qty'], limits=[0, 0.05])

# Respuesta:

# Winsorization reduce el impacto de outliers (valores extremadamente altos o bajos).

# No eliminamos los outliers, solo los ajustamos dentro de un rango razonable.

# Ejemplo: Si una ciudad tuvo un pico anormal de ventas por un evento especial, eso podría distorsionar el modelo.

# Winsorize lo ajusta para que el modelo no dependa excesivamente de estos valores extremos.

**# 7️⃣ ¿Por qué ReduceLROnPlateau y no un scheduler fijo de aprendizaje?**

ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=5, min\_lr=1e-5)

# Respuesta:

# ReduceLROnPlateau baja la tasa de aprendizaje automáticamente cuando la pérdida no mejora.

# Un scheduler fijo bajaría la tasa en momentos predefinidos, sin importar si es necesario o no.

**# 8️⃣ ¿Por qué wMAPE y no simplemente MAE o MSE?**

def weighted\_mape(y\_true, y\_pred):

y\_true, y\_pred = np.array(y\_true), np.array(y\_pred)

w = y\_true / (y\_true.sum() + 1e-10)

rel\_error = np.abs(y\_true - y\_pred) / (y\_true + 1e-10)

return np.sum(w \* rel\_error) \* 100

# Respuesta:

# MSE y MAE no ponderan los errores según la importancia de cada ciudad.

# wMAPE da más peso a ciudades con ventas altas, asegurando que el modelo no se enfoque solo en ciudades con menos ventas.

**# 9️⃣ ¿Por qué no usaste validación cruzada?**

validation\_split=0.15

# Respuesta:

# La validación cruzada mezcla los datos de entrenamiento y validación aleatoriamente.

# En series temporales, no podemos hacer eso porque romperíamos la secuencia de tiempo.

# En su lugar, usamos el 15% de los datos más recientes como validación para que el modelo siempre prediga en el futuro.

**# 🔟 ¿Por qué la función de pérdida custom\_loss usa una penalización exponencial?**

def custom\_loss(y\_true, y\_pred):

error = K.abs(y\_true - y\_pred)

penalty = K.exp(y\_true / K.max(y\_true))

return K.mean(error \* penalty)

# Respuesta:

# La penalización exponencial hace que los errores en ciudades con ventas altas sean más importantes.

# Esto evita que el modelo minimice la pérdida general sacrificando la precisión en ciudades grandes.

**# 1️⃣1️ ¿Cómo interpretas los resultados obtenidos en test?**

# Respuesta:

# - RMSE en entrenamiento: 14,603

# - RMSE en test: 19,957

# - Precisión en entrenamiento: 81.10%

# - Precisión en test: 78.90%

# El modelo generaliza bien porque la precisión en test no cae drásticamente.

# Un RMSE más alto en test es esperado, pero la diferencia es razonable.

**# 1️⃣2️⃣ ¿Por qué elegiste Adam como optimizador y un learning rate de 0.0002?**

model.compile(optimizer=Adam(learning\_rate=0.0002), loss=custom\_loss)

# Respuesta:

# Adam combina las ventajas de otros métodos de optimización.

# Se probó con learning rates más grandes y más pequeños, y 0.0002 fue el mejor equilibrio.

**# 1️⃣3️⃣ ¿Cómo justificas la eliminación de ciertas variables?**

# Variables eliminadas: ipc, tasa\_empleo, salario\_nominal, salario\_real, temperature\_2m

# Respuesta:

# Se eliminaron porque su correlación con las ventas era muy baja.

# Mantenerlas habría añadido ruido sin mejorar las predicciones.

**# 1️⃣4️⃣ ¿Cómo garantizas que la normalización en entrenamiento se mantenga en producción?**

SCALER\_PATH = "nuevos\_modelos/scaler.pkl"

joblib.dump(scaler, SCALER\_PATH)

# Respuesta:

# Guardamos el scaler después del entrenamiento para asegurarnos de que en producción los datos sean transformados igual.

**# 1️⃣5️⃣ ¿Cómo aseguraste que la estacionalidad se mantuviera en el modelo?**

df['season'] = df['date'].dt.month.apply(lambda x: 1 if x in [1, 2, 3, 4, 5, 10, 11, 12] else 0)

# Respuesta:

# Creamos la variable "season" para identificar verano/invierno en Ecuador. Esto permite capturar efectos estacionales.

**# 1️⃣6️⃣ ¿Por qué usaste una media móvil de 4 semanas para qty\_trend?**

df\_weekly['qty\_trend'] = df\_weekly.groupby('areacity')['qty'].transform(lambda x: x.rolling(window=4, min\_periods=1).mean())

# Respuesta:

# 4 semanas permiten capturar tendencias mensuales sin perder sensibilidad a cambios recientes.

**# 1️⃣7️⃣ ¿Por qué eliminaste outliers solo en qty y no en otras variables?**

df\_weekly['qty'] = winsorize(df\_weekly['qty'], limits=[0, 0.05])

# Respuesta:

# Las ventas tienen variaciones extremas que pueden distorsionar el modelo. Otras variables como is\_big\_city o season

# ya están definidas y no tienen valores extremos que necesiten ajustes.

**# 1️⃣8️ ¿Por qué convertiste la fecha a formato año-semana en lugar de usar días o meses?**

df['year\_week'] = df['date'].dt.strftime('%Y-%U')

# Respuesta:

# Usamos semanas porque es la granularidad relevante en este problema.

# Días harían la serie muy ruidosa, y meses perderían demasiada información.

**# 1️⃣ Pregunta: Inicialmente agregaron variables externas como IPC, tasa de empleo y salarios. ¿Por qué decidieron eliminarlas después?**

"""

✅ Respuesta:

Inicialmente pensamos que variables como IPC, tasa de empleo y salarios podían influir en las ventas semanales. Sin embargo, al analizar la matriz de correlación, vimos que tenían correlaciones muy bajas con `qty` y `qty\_trend` (menores a 0.3).

Esto significa que su impacto en las ventas era mínimo.

Ejemplo para entenderlo:

Supongamos que tenemos una tienda de ropa y queremos predecir las ventas. Podríamos pensar que el salario promedio de las personas influye en cuánto compran. Sin embargo, si revisamos los datos y vemos que el salario ha subido, pero las ventas no han cambiado, significa que no hay una relación fuerte entre esas variables.

📌 Código donde eliminamos estas variables:

"""

features = ['qty\_trend', 'season', 'is\_big\_city']

**# 2️⃣ Pregunta: Si eliminamos `festivals` y `holiday\_flag`, ¿significa que las festividades no afectan las ventas**?

"""

✅ Respuesta:

No significa que las festividades no afecten las ventas, sino que la forma en que estábamos usando esas variables no ayudaba al modelo.

Ejemplo:

Si usamos solo una variable que dice "1" cuando hay un festival y "0" cuando no lo hay, no sabemos qué tipo de festival es ni si realmente afecta a las ventas. Además, si en los datos solo hay unos pocos días festivos, el modelo no tiene suficiente información para aprender un patrón.

📌 Código donde eliminamos estas variables:

"""

features = ['qty\_trend', 'season', 'is\_big\_city']

**# 3️⃣ Pregunta: ¿Cómo justifican la elección de `qty\_trend`, `season` e `is\_big\_city`, si en la matriz de correlación su relación con `qty` no era tan fuerte?**

"""

✅ Respuesta:

La correlación mide relaciones lineales, pero en problemas de series de tiempo como este, las relaciones pueden ser más complejas.

Ejemplo:

Imagina que quieres predecir la cantidad de paraguas vendidos. Si miras la correlación con la temperatura, puede ser baja, porque aunque la temperatura no cambia mucho, la cantidad de lluvia sí lo hace. Sin embargo, si usamos una variable como "número de días de lluvia en la semana", podríamos capturar mejor la tendencia.

📌 Código donde se normalizaron estas variables:

"""

scaler = StandardScaler()

df\_weekly[features] = scaler.fit\_transform(df\_weekly[features])

**# 4️⃣ Pregunta: ¿Por qué usaron una ventana de 16 semanas en el modelo? ¿Cómo saben que es el mejor valor?**

"""

✅ Respuesta:

Probamos diferentes ventanas de tiempo (8, 12, 16, 20 semanas) y encontramos que:

- Ventanas muy cortas (8-12 semanas) hacen que el modelo reaccione demasiado a cambios pequeños.

- Ventanas muy largas (20 semanas o más) hacen que el modelo olvide patrones recientes.

Ejemplo:

Si manejas un restaurante y tratas de predecir cuántos clientes tendrás, mirar solo las últimas 4 semanas podría no ser suficiente si hubo un feriado. Pero si miras los últimos 2 años, podrías estar promediando demasiado y no notar una caída reciente en clientes.

📌 Código donde definimos la ventana de tiempo:

"""

ventana = 16 # Semanas de historial usadas para predecir

**# 5️⃣ Pregunta: ¿Por qué usaron `Dropout=0.15` en las capas LSTM y `Dropout=0.25` en la capa densa?**

"""

✅ Respuesta:

Usamos `dropout` para evitar que el modelo se sobreajuste a los datos de entrenamiento.

Ejemplo:

Si estás estudiando para un examen y solo memorizas respuestas sin entenderlas, podrías tener un buen desempeño en las preguntas que viste antes, pero fallar en preguntas nuevas. En modelos de Machine Learning, el `dropout` obliga a la red a no depender demasiado de ciertas conexiones, haciendo que generalice mejor.

📌 Código donde aplicamos dropout:

"""

x = LSTM(128, return\_sequences=True, dropout=0.15)(x)

x = LSTM(32, return\_sequences=False, dropout=0.15)(x)

fusion = Dropout(0.25)(fusion)

**# 6️⃣ Pregunta: ¿Por qué usaron una función de pérdida personalizada en lugar de un MSE estándar?**

"""

✅ Respuesta:

El MSE trata todos los errores por igual, pero en nuestro caso, queríamos penalizar más los errores en ciudades grandes con ventas altas.

Ejemplo:

Si tienes una tienda pequeña que vende 5 productos al día y hay un error de 5 unidades en la predicción, es un problema grande. Pero si tienes un supermercado que vende 500 productos al día y hay un error de 5 unidades, el impacto es mucho menor.

📌 Código donde implementamos la función de pérdida:

"""

def custom\_loss(y\_true, y\_pred):

error = K.abs(y\_true - y\_pred)

penalty = K.exp(y\_true / K.max(y\_true))

return K.mean(error \* penalty)

# 7️⃣ Pregunta: ¿Por qué no usaron validación cruzada en vez de solo una validación fija?

"""

✅ Respuesta:

En series de tiempo, la validación cruzada no es adecuada porque mezcla datos futuros con datos pasados. En su lugar, usamos validación temporal.

Ejemplo:

Si estás viendo cómo cambia la temperatura a lo largo del año, no puedes usar datos de diciembre para predecir junio. La validación temporal usa los datos más antiguos para entrenar y los más recientes para validar.

📌 Código donde dividimos los datos:

"""

history = model.fit(

x=[X\_feats], y=y\_all,

epochs=100,

batch\_size=16,

validation\_split=0.15,

callbacks=[EarlyStopping(monitor='val\_loss', patience=15)]

)

**# 8️⃣ Pregunta: ¿Cómo podríamos incluir datos externos como IPC o festivales de manera efectiva?**

"""

✅ Respuesta:

Si queremos incluir estos datos, podríamos hacerlo de una forma más informativa. Algunas opciones serían:

- Crear una variable de `cantidad de días festivos en la semana` en lugar de un solo `1` o `0`.

- En lugar de usar el `IPC` directamente, calcular su variación semanal o mensual.

Ejemplo:

Si vendes boletos de avión, no basta con saber si hay un feriado; necesitas saber si el feriado es un puente largo o un solo día.

📌 Código donde podríamos mejorar las variables:

"""

df\_weekly['festive\_days\_count'] = df.groupby(['year\_week', 'areacity'])['festivals'].sum().reset\_index(drop=True)

**# 9️⃣ Pregunta: Si las ventas varían mucho por ciudad, ¿por qué no entrenar un modelo por cada ciudad en vez de uno general?**

"""

✅ Respuesta:

Entrenar un modelo por ciudad tendría dos problemas:

1. Muchas ciudades tienen pocos datos, lo que haría que su modelo no pueda generalizar bien.

2. Un modelo general puede aprender patrones que se repiten en varias ciudades.

Ejemplo:

Si un supermercado en Quito y uno en Guayaquil tienen patrones similares, un modelo general podría aprender esas tendencias mejor que dos modelos separados.

📌 Código donde el modelo aprende patrones generales:

"""

for \_, subset in df\_weekly.groupby('areacity'):

target\_vals = subset['qty\_log'].values

feats = subset[features].values

for i in range(ventana, len(subset)):

X\_feats.append(feats[i-ventana:i])

y\_all.append(target\_vals[i])

**# 🔟 Pregunta: ¿Cómo podemos mejorar el modelo sin agregar más variables?**

"""

✅ Respuesta:

Algunas mejoras podrían ser:

- Probar arquitecturas más complejas como Transformer en lugar de LSTM.

- Ajustar hiperparámetros como el tamaño de la ventana o la tasa de aprendizaje.

- Incluir más datos históricos si están disponibles.

Ejemplo:

Si una tienda tiene datos de 5 años, podríamos entrenar el modelo con más información en lugar de solo usar 2 años.

📌 Código donde podríamos ajustar hiperparámetros:

"""

model.compile(optimizer=Adam(learning\_rate=0.0001), loss=custom\_loss)