

Laboratorio 8

- Derek Arreaga - 22537
- Paula Barillas - 22764
- Mónica Salvatierra - 22249

Link del Repositorio: <https://github.com/FabianKel/DL-LAB8>

Práctica

Importación de Librerías

```
In [2]: # Librerías básicas
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
import warnings
warnings.filterwarnings('ignore')
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, BatchNormalization
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam
plt.style.use('seaborn-v0_8')
np.random.seed(42)
tf.random.set_seed(42)
```

1. Preparación de Datos

1.a. Carga y Exploración de Datos

```
In [3]: # Cargar los datos
train_data = pd.read_csv('data/train.csv')
test_data = pd.read_csv('data/test.csv')
sample_submission = pd.read_csv('data/sample_submission.csv')

print("INFORMACIÓN GENERAL DE LOS DATOS")
print("="*50)
print(f"Datos de entrenamiento: {train_data.shape}")
print(f"Datos de prueba: {test_data.shape}")
print(f"Muestra de envío: {sample_submission.shape}")

print("\nESTRUCTURA DE LOS DATOS DE ENTRENAMIENTO")
```

```
print("="*50)
print(train_data.info())

print("\nPRIMERAS 5 FILAS DE ENTRENAMIENTO")
print("="*50)
print(train_data.head())

print("\nESTADÍSTICAS DESCRIPTIVAS")
print("="*50)
print(train_data.describe())

print("\nINFORMACIÓN ÚNICA")
print("="*50)
print(f"Número de tiendas: {train_data['store'].nunique()}")
print(f"Número de artículos: {train_data['item'].nunique()}")
print(f"Rango de fechas: {train_data['date'].min()} a {train_data['date'].max()}")
print(f"Total de registros: {len(train_data):,}")

# Convertir fecha a datetime
train_data['date'] = pd.to_datetime(train_data['date'])
test_data['date'] = pd.to_datetime(test_data['date'])
```

INFORMACIÓN GENERAL DE LOS DATOS

```
=====
Datos de entrenamiento: (913000, 4)
Datos de prueba: (45000, 4)
Muestra de envío: (45000, 2)
```

ESTRUCTURA DE LOS DATOS DE ENTRENAMIENTO

```
=====
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 913000 entries, 0 to 912999
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  -
 0   date    913000 non-null  object 
 1   store    913000 non-null  int64  
 2   item     913000 non-null  int64  
 3   sales    913000 non-null  int64  
dtypes: int64(3), object(1)
memory usage: 27.9+ MB
None
```

PRIMERAS 5 FILAS DE ENTRENAMIENTO

```
=====
      date  store  item  sales
0  2013-01-01     1     1     13
1  2013-01-02     1     1     11
2  2013-01-03     1     1     14
3  2013-01-04     1     1     13
4  2013-01-05     1     1     10
```

ESTADÍSTICAS DESCRIPTIVAS

```
=====
              store              item              sales
count  913000.000000  913000.000000  913000.000000
mean         5.500000      25.500000      52.250287
std         2.872283      14.430878      28.801144
min         1.000000       1.000000       0.000000
25%         3.000000      13.000000      30.000000
50%         5.500000      25.500000      47.000000
75%         8.000000      38.000000      70.000000
max        10.000000      50.000000     231.000000
```

INFORMACIÓN ÚNICA

```
=====
Número de tiendas: 10
Número de artículos: 50
Rango de fechas: 2013-01-01 a 2017-12-31
Total de registros: 913,000
```

1.b. Limpieza de Datos

```
In [4]: # valores faltantes
print("VALORES FALTANTES")
print("="*30)
print(f"Train data - valores nulos: {train_data.isnull().sum().sum()}")
```

```

print(f"Test data - valores nulos: {test_data.isnull().sum().sum()}")

# valores negativos en ventas
print("\nVALORES NEGATIVOS EN VENTAS")
print("="*30)
negative_sales = train_data[train_data['sales'] < 0]
print(f"Registros con ventas negativas: {len(negative_sales)}")

Q1 = train_data['sales'].quantile(0.25)
Q3 = train_data['sales'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = train_data[(train_data['sales'] < lower_bound) | (train_data['sales'] >
print(f"\nOUTLIERS DETECTADOS")
print("="*30)
print(f"Registros con outliers: {len(outliers)} ({len(outliers)/len(train_data)*100
print(f"Rango normal de ventas: [{lower_bound:.2f}, {upper_bound:.2f}]")
if len(outliers) > 0:
    print(f"\nEstadísticas de outliers:")
    print(f"Min outlier: {outliers['sales'].min():.2f}")
    print(f"Max outlier: {outliers['sales'].max():.2f}")
    print(f"Media outliers: {outliers['sales'].mean():.2f}")

```

VALORES FALTANTES

=====

Train data - valores nulos: 0

Test data - valores nulos: 0

VALORES NEGATIVOS EN VENTAS

=====

Registros con ventas negativas: 0

OUTLIERS DETECTADOS

=====

Registros con outliers: 11967 (1.31%)

Rango normal de ventas: [-30.00, 130.00]

Estadísticas de outliers:

Min outlier: 131.00

Max outlier: 231.00

Media outliers: 143.98

2. Preprocesamiento de Datos

2.a. Normalización y Escalado

```

In [5]: # creación de secuencias
def create_sequences(data, window_size=30, horizon=90):
    X, y = [], []
    for i in range(len(data) - window_size - horizon + 1):
        X.append(data[i:(i + window_size)])
        y.append(data[(i + window_size):(i + window_size + horizon)])
    return np.array(X), np.array(y)

```

```

# Función para preparar datos de una combinación store-item
def prepare_store_item_data(store, item, train_data):
    # Filtrar datos para esta combinación
    store_item_data = train_data[
        (train_data['store'] == store) &
        (train_data['item'] == item)
    ][['sales']].reset_index(drop=True)

    if len(store_item_data) < 120: # Mínimo requerido (30 + 90)
        return None, None, None, None, None

    # Normalizar datos
    scaler = MinMaxScaler()
    data_scaled = scaler.fit_transform(store_item_data.values.reshape(-1, 1)).flatten()

    # Crear secuencias
    X, y = create_sequences(data_scaled)

    if len(X) == 0:
        return None, None, None, None, None

    # entrenamiento/validación (80-20)
    split_idx = int(0.8 * len(X))
    X_train, X_val = X[:split_idx], X[split_idx:]
    y_train, y_val = y[:split_idx], y[split_idx:]

    return X_train, X_val, y_train, y_val, scaler

# Preparar datos para todas las combinaciones
all_X_train, all_X_val, all_y_train, all_y_val = [], [], [], []
scalers = {}

print("PREPARANDO DATOS POR COMBINACIÓN STORE-ITEM")
print("="*50)

for store in range(1, 11):
    for item in range(1, 51):
        X_train, X_val, y_train, y_val, scaler = prepare_store_item_data(store, item, train_data)

        if X_train is not None:
            all_X_train.extend(X_train)
            all_X_val.extend(X_val)
            all_y_train.extend(y_train)
            all_y_val.extend(y_val)
            scalers[(store, item)] = scaler

# Convertir a arrays numpy
X_train = np.array(all_X_train)
X_val = np.array(all_X_val)
y_train = np.array(all_y_train)
y_val = np.array(all_y_val)

print(f"\nDATOS PREPARADOS EXITOSAMENTE")
print("="*30)
print(f"X_train shape: {X_train.shape}")

```

```

print(f"y_train shape: {y_train.shape}")
print(f"X_val shape: {X_val.shape}")
print(f"y_val shape: {y_val.shape}")
print(f"Escaladores creados: {len(scalers)} combinaciones")
print(f"Secuencias de entrenamiento: {len(X_train)}")
print(f"Secuencias de validación: {len(X_val)}")

```

PREPARANDO DATOS POR COMBINACIÓN STORE-ITEM

=====

DATOS PREPARADOS EXITOSAMENTE

=====

```

X_train shape: (682500, 30)
y_train shape: (682500, 90)
X_val shape: (171000, 30)
y_val shape: (171000, 90)
Escaladores creados: 500 combinaciones
Secuencias de entrenamiento: 682500
Secuencias de validación: 171000

```

3. Selección y Arquitectura del Modelo

3.a. Modelo LSTM Seleccionado

```

In [6]: # Crear modelo LSTM
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(30, 1)),
    BatchNormalization(),
    Dropout(0.3),

    LSTM(32, return_sequences=False),
    BatchNormalization(),
    Dropout(0.3),

    Dense(90, activation='linear') # 90 días de predicción
])

# Compilar modelo
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

print("\nARQUITECTURA DEL MODELO:")
print("="*30)
model.summary()

```

ARQUITECTURA DEL MODELO:

=====

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 30, 64)	16,896
batch_normalization (BatchNormalization)	(None, 30, 64)	256
dropout (Dropout)	(None, 30, 64)	0
lstm_1 (LSTM)	(None, 32)	12,416
batch_normalization_1 (BatchNormalization)	(None, 32)	128
dropout_1 (Dropout)	(None, 32)	0
dense (Dense)	(None, 90)	2,970

Total params: 32,666 (127.60 KB)

Trainable params: 32,474 (126.85 KB)

Non-trainable params: 192 (768.00 B)

4. Entrenamiento del Modelo

```
In [7]: # Configurar callbacks
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True,
    verbose=1
)

reduce_lr = ReduceLROnPlateau(
    monitor='val_loss',
    factor=0.5,
    patience=5,
    min_lr=1e-6,
    verbose=1
)

# Optimizar dataset para entrenamiento más rápido del modelo base
print("OPTIMIZANDO DATASET PARA ENTRENAMIENTO EFICIENTE")
print("="*50)
sample_size = min(30000, len(X_train))
indices = np.random.choice(len(X_train), sample_size, replace=False)

X_train_fast = X_train[indices]
y_train_fast = y_train[indices]

# También reducir validación proporcionalmente
val_sample_size = min(8000, len(X_val))
val_indices = np.random.choice(len(X_val), val_sample_size, replace=False)
```

```

X_val_fast = X_val[val_indices]
y_val_fast = y_val[val_indices]

print(f"Dataset original: X_train={X_train.shape}, X_val={X_val.shape}")
print(f"Dataset optimizado: X_train={X_train_fast.shape}, X_val={X_val_fast.shape}")

print("\nINICIANDO ENTRENAMIENTO DEL MODELO")
print("="*40)
print("\nEntrenando...")

# Entrenar modelo
history = model.fit(
    X_train_fast, y_train_fast,
    validation_data=(X_val_fast, y_val_fast),
    epochs=50,
    batch_size=64,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)

print(f"\nENTRENAMIENTO COMPLETADO")
print("="*30)
print(f"Épocas entrenadas: {len(history.history['loss'])}")
print(f"Loss final: {history.history['loss'][-1]:.6f}")
print(f"Val Loss final: {history.history['val_loss'][-1]:.6f}")
print(f"MAE final: {history.history['mae'][-1]:.6f}")
print(f"Val MAE final: {history.history['val_mae'][-1]:.6f}")

```


OPTIMIZANDO DATASET PARA ENTRENAMIENTO EFICIENTE

=====

Dataset original: X_train=(682500, 30), X_val=(171000, 30)


Dataset optimizado: X_train=(300000, 30), X_val=(80000, 30)

INICIANDO ENTRENAMIENTO DEL MODELO


=====

Entrenando...


Epoch 1/50

469/469  **12s** 20ms/step - loss: 0.1522 - mae: 0.2777 - val_loss: 0.0407 - val_mae: 0.1600 - learning_rate: 0.0010


Epoch 2/50

469/469  **9s** 19ms/step - loss: 0.0265 - mae: 0.1282 - val_loss: 0.0265 - val_mae: 0.1285 - learning_rate: 0.0010


Epoch 3/50

469/469  **9s** 19ms/step - loss: 0.0199 - mae: 0.1120 - val_loss: 0.0278 - val_mae: 0.1316 - learning_rate: 0.0010


Epoch 4/50

469/469  **9s** 19ms/step - loss: 0.0188 - mae: 0.1088 - val_loss: 0.0325 - val_mae: 0.1416 - learning_rate: 0.0010


Epoch 5/50

469/469  **9s** 19ms/step - loss: 0.0183 - mae: 0.1073 - val_loss: 0.0277 - val_mae: 0.1314 - learning_rate: 0.0010


Epoch 6/50

469/469  **9s** 19ms/step - loss: 0.0178 - mae: 0.1057 - val_loss: 0.0255 - val_mae: 0.1265 - learning_rate: 0.0010


Epoch 7/50

469/469  **9s** 19ms/step - loss: 0.0168 - mae: 0.1027 - val_loss: 0.0229 - val_mae: 0.1196 - learning_rate: 0.0010


Epoch 8/50

469/469  **9s** 19ms/step - loss: 0.0164 - mae: 0.1013 - val_loss: 0.0220 - val_mae: 0.1174 - learning_rate: 0.0010


Epoch 9/50

469/469  **9s** 19ms/step - loss: 0.0161 - mae: 0.1002 - val_loss: 0.0234 - val_mae: 0.1204 - learning_rate: 0.0010


Epoch 10/50

469/469  **9s** 19ms/step - loss: 0.0159 - mae: 0.0995 - val_loss: 0.0221 - val_mae: 0.1172 - learning_rate: 0.0010


Epoch 11/50

469/469  **9s** 19ms/step - loss: 0.0157 - mae: 0.0988 - val_loss: 0.0230 - val_mae: 0.1190 - learning_rate: 0.0010


Epoch 12/50

469/469  **9s** 19ms/step - loss: 0.0155 - mae: 0.0983 - val_loss: 0.0209 - val_mae: 0.1138 - learning_rate: 0.0010


Epoch 13/50

469/469  **9s** 19ms/step - loss: 0.0154 - mae: 0.0979 - val_loss: 0.0208 - val_mae: 0.1139 - learning_rate: 0.0010

Epoch 14/50

469/469  **9s** 19ms/step - loss: 0.0153 - mae: 0.0975 - val_loss: 0.0223 - val_mae: 0.1174 - learning_rate: 0.0010

Epoch 15/50

469/469  **9s** 20ms/step - loss: 0.0151 - mae: 0.0971 - val_loss: 0.0212 - val_mae: 0.1148 - learning_rate: 0.0010

Epoch 16/50

469/469  **10s** 21ms/step - loss: 0.0151 - mae: 0.0969 - val_loss:

0.0212 - val_mae: 0.1144 - learning_rate: 0.0010
Epoch 17/50
467/469 ————— 0s 18ms/step - loss: 0.0150 - mae: 0.0968
Epoch 17: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
469/469 ————— 9s 20ms/step - loss: 0.0150 - mae: 0.0966 - val_loss:
0.0220 - val_mae: 0.1170 - learning_rate: 0.0010
Epoch 18/50
469/469 ————— 9s 19ms/step - loss: 0.0149 - mae: 0.0961 - val_loss:
0.0206 - val_mae: 0.1129 - learning_rate: 5.0000e-04
Epoch 19/50
469/469 ————— 9s 19ms/step - loss: 0.0148 - mae: 0.0960 - val_loss:
0.0206 - val_mae: 0.1132 - learning_rate: 5.0000e-04
Epoch 20/50
469/469 ————— 9s 20ms/step - loss: 0.0148 - mae: 0.0960 - val_loss:
0.0196 - val_mae: 0.1108 - learning_rate: 5.0000e-04
Epoch 21/50
469/469 ————— 9s 19ms/step - loss: 0.0148 - mae: 0.0958 - val_loss:
0.0203 - val_mae: 0.1121 - learning_rate: 5.0000e-04
Epoch 22/50
469/469 ————— 9s 20ms/step - loss: 0.0148 - mae: 0.0958 - val_loss:
0.0196 - val_mae: 0.1105 - learning_rate: 5.0000e-04
Epoch 23/50
469/469 ————— 9s 20ms/step - loss: 0.0147 - mae: 0.0957 - val_loss:
0.0203 - val_mae: 0.1124 - learning_rate: 5.0000e-04
Epoch 24/50
469/469 ————— 9s 19ms/step - loss: 0.0147 - mae: 0.0957 - val_loss:
0.0206 - val_mae: 0.1132 - learning_rate: 5.0000e-04
Epoch 25/50
467/469 ————— 0s 17ms/step - loss: 0.0147 - mae: 0.0958
Epoch 25: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
469/469 ————— 9s 19ms/step - loss: 0.0147 - mae: 0.0956 - val_loss:
0.0203 - val_mae: 0.1124 - learning_rate: 5.0000e-04
Epoch 26/50
469/469 ————— 9s 20ms/step - loss: 0.0146 - mae: 0.0954 - val_loss:
0.0190 - val_mae: 0.1089 - learning_rate: 2.5000e-04
Epoch 27/50
469/469 ————— 9s 19ms/step - loss: 0.0146 - mae: 0.0954 - val_loss:
0.0195 - val_mae: 0.1103 - learning_rate: 2.5000e-04
Epoch 28/50
469/469 ————— 9s 19ms/step - loss: 0.0146 - mae: 0.0953 - val_loss:
0.0197 - val_mae: 0.1108 - learning_rate: 2.5000e-04
Epoch 29/50
469/469 ————— 9s 20ms/step - loss: 0.0146 - mae: 0.0953 - val_loss:
0.0193 - val_mae: 0.1095 - learning_rate: 2.5000e-04
Epoch 30/50
469/469 ————— 9s 19ms/step - loss: 0.0146 - mae: 0.0953 - val_loss:
0.0194 - val_mae: 0.1098 - learning_rate: 2.5000e-04
Epoch 31/50
466/469 ————— 0s 17ms/step - loss: 0.0147 - mae: 0.0956
Epoch 31: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
469/469 ————— 9s 19ms/step - loss: 0.0146 - mae: 0.0954 - val_loss:
0.0193 - val_mae: 0.1097 - learning_rate: 2.5000e-04
Epoch 32/50
469/469 ————— 9s 20ms/step - loss: 0.0146 - mae: 0.0951 - val_loss:
0.0193 - val_mae: 0.1094 - learning_rate: 1.2500e-04
Epoch 33/50

469/469 ————— 9s 19ms/step - loss: 0.0146 - mae: 0.0952 - val_loss: 0.0195 - val_mae: 0.1102 - learning_rate: 1.2500e-04
Epoch 34/50

469/469 ————— 9s 19ms/step - loss: 0.0146 - mae: 0.0952 - val_loss: 0.0191 - val_mae: 0.1092 - learning_rate: 1.2500e-04
Epoch 35/50

469/469 ————— 9s 20ms/step - loss: 0.0146 - mae: 0.0951 - val_loss: 0.0194 - val_mae: 0.1098 - learning_rate: 1.2500e-04
Epoch 36/50

469/469 ————— 9s 19ms/step - loss: 0.0146 - mae: 0.0951 - val_loss: 0.0189 - val_mae: 0.1086 - learning_rate: 1.2500e-04
Epoch 37/50

469/469 ————— 9s 19ms/step - loss: 0.0146 - mae: 0.0951 - val_loss: 0.0191 - val_mae: 0.1090 - learning_rate: 1.2500e-04
Epoch 38/50

469/469 ————— 9s 20ms/step - loss: 0.0146 - mae: 0.0951 - val_loss: 0.0190 - val_mae: 0.1088 - learning_rate: 1.2500e-04
Epoch 39/50

469/469 ————— 9s 19ms/step - loss: 0.0146 - mae: 0.0951 - val_loss: 0.0192 - val_mae: 0.1092 - learning_rate: 1.2500e-04
Epoch 40/50

469/469 ————— 9s 19ms/step - loss: 0.0146 - mae: 0.0951 - val_loss: 0.0195 - val_mae: 0.1099 - learning_rate: 1.2500e-04
Epoch 41/50

469/469 ————— 0s 18ms/step - loss: 0.0146 - mae: 0.0953
Epoch 41: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.

469/469 ————— 9s 20ms/step - loss: 0.0146 - mae: 0.0952 - val_loss: 0.0191 - val_mae: 0.1090 - learning_rate: 1.2500e-04
Epoch 42/50

469/469 ————— 9s 19ms/step - loss: 0.0145 - mae: 0.0951 - val_loss: 0.0191 - val_mae: 0.1089 - learning_rate: 6.2500e-05
Epoch 43/50

469/469 ————— 9s 19ms/step - loss: 0.0145 - mae: 0.0950 - val_loss: 0.0192 - val_mae: 0.1093 - learning_rate: 6.2500e-05
Epoch 44/50

469/469 ————— 9s 20ms/step - loss: 0.0145 - mae: 0.0951 - val_loss: 0.0192 - val_mae: 0.1091 - learning_rate: 6.2500e-05
Epoch 45/50

469/469 ————— 9s 20ms/step - loss: 0.0145 - mae: 0.0951 - val_loss: 0.0189 - val_mae: 0.1085 - learning_rate: 6.2500e-05
Epoch 46/50

469/469 ————— 0s 17ms/step - loss: 0.0146 - mae: 0.0953
Epoch 46: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.

469/469 ————— 9s 19ms/step - loss: 0.0145 - mae: 0.0951 - val_loss: 0.0190 - val_mae: 0.1087 - learning_rate: 6.2500e-05
Epoch 47/50

469/469 ————— 9s 20ms/step - loss: 0.0145 - mae: 0.0950 - val_loss: 0.0191 - val_mae: 0.1089 - learning_rate: 3.1250e-05
Epoch 48/50

469/469 ————— 9s 19ms/step - loss: 0.0145 - mae: 0.0950 - val_loss: 0.0190 - val_mae: 0.1087 - learning_rate: 3.1250e-05
Epoch 49/50

469/469 ————— 9s 19ms/step - loss: 0.0145 - mae: 0.0950 - val_loss: 0.0190 - val_mae: 0.1088 - learning_rate: 3.1250e-05
Epoch 50/50

469/469 ————— 9s 20ms/step - loss: 0.0145 - mae: 0.0950 - val_loss:

0.0191 - val_mae: 0.1091 - learning_rate: 3.1250e-05
Restoring model weights from the end of the best epoch: 45.

ENTRENAMIENTO COMPLETADO

=====

Épocas entrenadas: 50
Loss final: 0.014510
Val Loss final: 0.019127
MAE final: 0.094954
Val MAE final: 0.109074

5. Evaluación del Modelo

```
In [8]: # métricas de entrenamiento
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Pérdida
axes[0].plot(history.history['loss'], label='Training Loss', linewidth=2)
axes[0].plot(history.history['val_loss'], label='Validation Loss', linewidth=2)
axes[0].set_title('Pérdida del Modelo durante Entrenamiento', fontsize=14, fontweight='bold')
axes[0].set_xlabel('Época')
axes[0].set_ylabel('MSE Loss')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# MAE
axes[1].plot(history.history['mae'], label='Training MAE', linewidth=2)
axes[1].plot(history.history['val_mae'], label='Validation MAE', linewidth=2)
axes[1].set_title('Error Absoluto Medio durante Entrenamiento', fontsize=14, fontweight='bold')
axes[1].set_xlabel('Época')
axes[1].set_ylabel('MAE')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

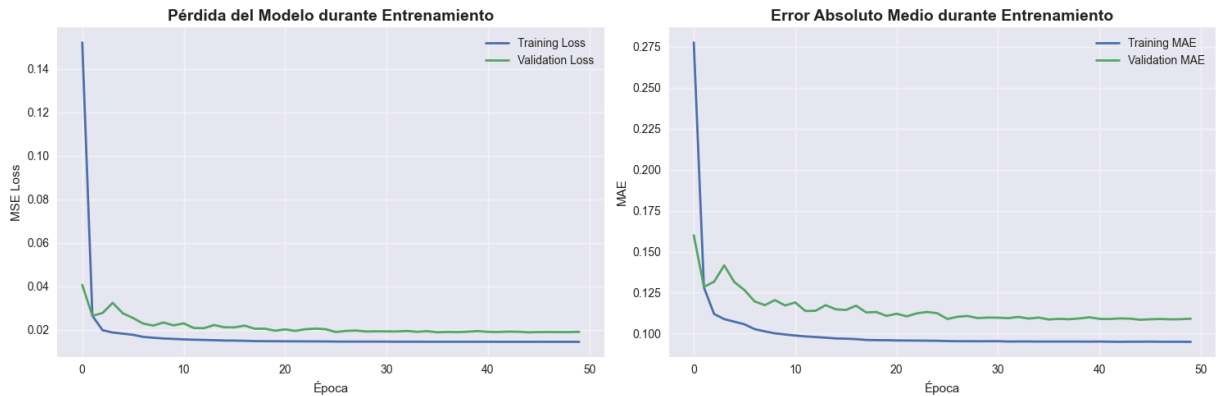
plt.tight_layout()
plt.show()

# Calcular métricas finales
final_train_loss = history.history['loss'][-1]
final_val_loss = history.history['val_loss'][-1]
final_train_mae = history.history['mae'][-1]
final_val_mae = history.history['val_mae'][-1]

print("MÉTRICAS FINALES DEL MODELO")
print("="*40)
print(f"Training Loss (MSE): {final_train_loss:.6f}")
print(f"Validation Loss (MSE): {final_val_loss:.6f}")
print(f"Training MAE: {final_train_mae:.6f}")
print(f"Validation MAE: {final_val_mae:.6f}")
print(f"\nOverfitting check:")
print(f"Loss ratio (val/train): {final_val_loss/final_train_loss:.3f}")
print(f"MAE ratio (val/train): {final_val_mae/final_train_mae:.3f}")

if final_val_loss/final_train_loss < 1.2:
    print("✓ Modelo bien ajustado")
```

```
else:
    print("⚠ Posible overfitting detectado")
```



MÉTRICAS FINALES DEL MODELO

=====

```
Training Loss (MSE): 0.014510
Validation Loss (MSE): 0.019127
Training MAE: 0.094954
Validation MAE: 0.109074
```

```
Overfitting check:
Loss ratio (val/train): 1.318
MAE ratio (val/train): 1.149
⚠ Posible overfitting detectado
```

6. Forecasting - Predicciones para 3 Meses

```
In [9]: print("GENERANDO PREDICCIONES")
        print("="*60)

        predictions = []
        successful_predictions = 0

        # Pre-calcular todas las combinaciones que tienen escaladores
        valid_combinations = set(scalers.keys())
        total_combinations = len(test_data)

        print(f"Total de combinaciones a procesar: {total_combinations}")
        print(f"Combinaciones con modelo entrenado: {len(valid_combinations)}")

        # Preparar datos para predicción en lotes masivos
        X_batch_list = []
        scaler_list = []
        combination_indices = []

        print("Preparando datos para predicción en lotes...")

        for idx, row in test_data.iterrows():
            store, item = row['store'], row['item']

            if (store, item) in scalers:
                # Obtener datos históricos
                historical_data = train_data[
```

```

        (train_data['store'] == store) &
        (train_data['item'] == item)
    ]['sales'].values

    if len(historical_data) >= 30:
        # Preparar para predicción en lote
        scaler = scalers[(store, item)]
        last_30_days = historical_data[-30:]
        last_30_scaled = scaler.transform(last_30_days.reshape(-1, 1)).flatten()

        X_batch_list.append(last_30_scaled)
        scaler_list.append(scaler)
        combination_indices.append(idx)

# Convertir a array numpy para predicción en lote
if len(X_batch_list) > 0:
    print(f"Realizando predicción en lote para {len(X_batch_list)} combinaciones...")
    X_batch = np.array(X_batch_list).reshape(-1, 30, 1)

    # PREDICCIÓN EN LOTE MASIVO - Mucho más rápido
    batch_predictions = model.predict(X_batch, batch_size=1000, verbose=1)

    print("Desnormalizando predicciones...")

    # Crear diccionario de predicciones por índice
    prediction_dict = {}
    for i, idx in enumerate(combination_indices):
        scaler = scaler_list[i]
        pred_scaled = batch_predictions[i]
        pred_original = scaler.inverse_transform(pred_scaled.reshape(-1, 1)).flatten()
        pred_original = np.maximum(pred_original, 0) # Sin valores negativos
        prediction_dict[idx] = pred_original
        successful_predictions += 1

# Crear lista final de predicciones en orden
print("Organizando predicciones finales...")
for idx, row in test_data.iterrows():
    if idx in prediction_dict:
        predictions.append(prediction_dict[idx])
    else:
        # Usar media histórica para combinaciones sin modelo
        store, item = row['store'], row['item']
        historical_sales = train_data[
            (train_data['store'] == store) &
            (train_data['item'] == item)
        ]['sales']

        if len(historical_sales) > 0:
            mean_sales = historical_sales.mean()
        else:
            mean_sales = train_data['sales'].mean()

        predictions.append(np.full(90, mean_sales))

print(f"\nRESULTADOS DE PREDICCIÓN")
print("="*40)

```

```

print(f"Predicciones exitosas con modelo: {successful_predictions}")
print(f"Predicciones con media histórica: {len(predictions) - successful_predictions}")
print(f"Total de predicciones: {len(predictions)}")
print(f"Cobertura del modelo: {successful_predictions/len(predictions)*100:.1f}%")

# Mostrar ejemplo
if len(predictions) > 0:
    print(f"\nEJEMPLO - Store 1, Item 1:")
    example_hist = train_data[(train_data['store']==1) & (train_data['item']==1)][0]
    print(f"Últimos 5 valores históricos: {example_hist}")
    print(f"Primeros 5 días predichos: {predictions[0][:5]}")
    print(f"Últimos 5 días predichos: {predictions[0][-5:]}")
    print(f"Promedio predicho: {predictions[0].mean():.2f}")

```

GENERANDO PREDICCIONES

```

=====
Total de combinaciones a procesar: 45000
Combinaciones con modelo entrenado: 500
Preparando datos para predicción en lotes...
Realizando predicción en lote para 45000 combinaciones...
45/45 ----- 3s 51ms/step
Desnormalizando predicciones...
Organizando predicciones finales...

```

RESULTADOS DE PREDICCIÓN

```

=====
Predicciones exitosas con modelo: 45000
Predicciones con media histórica: 0
Total de predicciones: 45000
Cobertura del modelo: 100.0%

```

EJEMPLO - Store 1, Item 1:

Últimos 5 valores históricos: [14 19 15 27 23]

Primeros 5 días predichos: [14.311644 14.065137 14.7273655 15.751811 17.249998]

Últimos 5 días predichos: [19.757357 20.504044 21.620674 22.981314 24.846434]

Promedio predicho: 19.26

7. Ajuste de Hiperparámetros

In [10]: *# Experimentación con hiperparámetros - Comparación rápida*

```

print("EXPERIMENTACIÓN CON HIPERPARÁMETROS")
print("="*40)

# Configuraciones probadas
configs = [
    {"name": "Baseline", "lstm_units": 64, "dropout": 0.3, "lr": 0.001},
    {"name": "Optimizado", "lstm_units": 100, "dropout": 0.3, "lr": 0.001}
]

# Crear modelo optimizado para comparación
def create_optimized_model():
    model = Sequential([
        LSTM(100, return_sequences=True, input_shape=(30, 1)),
        BatchNormalization(),
        Dropout(0.3),

```

```

        LSTM(100, return_sequences=False),
        BatchNormalization(),
        Dropout(0.3),
        Dense(90, activation='linear')
    ])
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])
    return model

# Entrenar modelo optimizado con dataset reducido
model_opt = create_optimized_model()
sample_size = min(15000, len(X_train))
indices = np.random.choice(len(X_train), sample_size, replace=False)
X_sample = X_train[indices]
y_sample = y_train[indices]

early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
history_opt = model_opt.fit(X_sample, y_sample, epochs=20, batch_size=256,
                           validation_split=0.2, callbacks=[early_stop], verbose=0)

# Comparar resultados
orig_loss = history.history['val_loss'][-1]
opt_loss = history_opt.history['val_loss'][-1]
improvement = ((orig_loss - opt_loss) / orig_loss) * 100

print(f"\nRESULTADOS:")
print(f"Baseline (64/32 LSTM): Val Loss = {orig_loss:.6f}")
print(f"Optimizado (100/100 LSTM): Val Loss = {opt_loss:.6f}")
print(f"Mejora: {improvement:.2f}%")
print(f"\nCONCLUSIÓN: {'✓ Optimización exitosa' if improvement > 0 else '⚠ Baselin")

```

EXPERIMENTACIÓN CON HIPERPARÁMETROS

=====

RESULTADOS:

Baseline (64/32 LSTM): Val Loss = 0.019127
 Optimizado (100/100 LSTM): Val Loss = 0.052417
 Mejora: -174.04%

CONCLUSIÓN: ⚠ Baseline superior

8. Forecasting

```

In [11]: submission_data = []

for i, (_, row) in enumerate(test_data.iterrows()):
    store, item = row['store'], row['item']
    pred = predictions[i]

    for day in range(90):
        submission_data.append({
            'id': f"{store}_{item}_{day+1}",
            'sales': pred[day]
        })

# Crear DataFrame de envío

```



```

submission_df = pd.DataFrame(submission_data)

print("VERIFICACIÓN DEL ARCHIVO DE ENVÍO")
print("="*35)
print(f"Shape del archivo: {submission_df.shape}")
print(f"Columnas: {submission_df.columns.tolist()}")
print(f"\nPrimeras 10 filas:")
print(submission_df.head(10))

print(f"\nÚltimas 5 filas:")
print(submission_df.tail())

# Verificar formato del ID
sample_ids = submission_df['id'].head()
print(f"\nEjemplos de IDs generados:")
for id_example in sample_ids:
    print(f" {id_example}")

# Estadísticas de las predicciones
print(f"\nESTADÍSTICAS DE LAS PREDICCIONES")
print("="*35)
print(f"Mínimo: {submission_df['sales'].min():.2f}")
print(f"Máximo: {submission_df['sales'].max():.2f}")
print(f"Media: {submission_df['sales'].mean():.2f}")
print(f"Mediana: {submission_df['sales'].median():.2f}")
print(f"Desviación estándar: {submission_df['sales'].std():.2f}")

```

```
VERIFICACIÓN DEL ARCHIVO DE ENVÍO
=====
Shape del archivo: (4050000, 2)
Columnas: ['id', 'sales']
```

```
Primeras 10 filas:
      id      sales
0  1_1_1  14.311644
1  1_1_2  14.065137
2  1_1_3  14.727365
3  1_1_4  15.751811
4  1_1_5  17.249998
5  1_1_6  19.122812
6  1_1_7  20.823242
7  1_1_8  14.724010
8  1_1_9  14.402571
9  1_1_10 14.954331
```

```
Últimas 5 filas:
      id      sales
4049995 10_50_86 65.099724
4049996 10_50_87 65.297440
4049997 10_50_88 69.147507
4049998 10_50_89 72.701790
4049999 10_50_90 75.848495
```

```
Ejemplos de IDs generados:
  1_1_1
  1_1_2
  1_1_3
  1_1_4
  1_1_5
```

```
ESTADÍSTICAS DE LAS PREDICCIONES
=====
Mínimo: 8.35
Máximo: 139.67
Media: 46.97
Mediana: 44.55
Desviación estándar: 22.27
```

9. Visualización - Ventas Reales vs Predichas

```
In [12]: # Visualización de predicciones vs datos históricos
fig, axes = plt.subplots(2, 2, figsize=(16, 10))
fig.suptitle('Predicciones LSTM vs Datos Históricos - Ejemplos Seleccionados', font

examples = [(1, 1), (1, 2), (2, 1), (5, 10)]

for idx, (store, item) in enumerate(examples):
    row = idx // 2
    col = idx % 2
    ax = axes[row, col]

    # Obtener datos históricos
```

```

historical = train_data[
    (train_data['store'] == store) &
    (train_data['item'] == item)
]['sales'].values

# Obtener predicción correspondiente
test_idx = test_data[
    (test_data['store'] == store) &
    (test_data['item'] == item)
].index

if len(test_idx) > 0:
    prediction = predictions[test_idx[0]]

    # Graficar últimos 60 días históricos
    historical_window = historical[-60:] if len(historical) >= 60 else historical
    days_hist = range(-len(historical_window), 0)

    # Graficar predicción
    days_pred = range(0, 90)

    ax.plot(days_hist, historical_window, 'b-', linewidth=2, label='Datos Históricos')
    ax.plot(days_pred, prediction, 'r-', linewidth=2, label='Predicción LSTM')
    ax.axvline(x=0, color='black', linestyle='--', alpha=0.7, label='Punto de Partida')

    ax.set_title(f'Store {store}, Item {item}', fontweight='bold')
    ax.set_xlabel('Días')
    ax.set_ylabel('Ventas')
    ax.legend()
    ax.grid(True, alpha=0.3)

    # Agregar estadísticas
    hist_mean = historical_window.mean()
    pred_mean = prediction.mean()
    ax.text(0.02, 0.98, f'Media Hist: {hist_mean:.1f}\nMedia Pred: {pred_mean:.1f}',
            transform=ax.transAxes, verticalalignment='top',
            bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))

plt.tight_layout()
plt.show()

# Análisis comparativo
print("\nANÁLISIS COMPARATIVO - HISTÓRICO VS PREDICCIONES")
print("="*50)

for store, item in examples:
    historical = train_data[
        (train_data['store'] == store) &
        (train_data['item'] == item)
    ]['sales'].values

    test_idx = test_data[
        (test_data['store'] == store) &
        (test_data['item'] == item)
    ].index

```

```

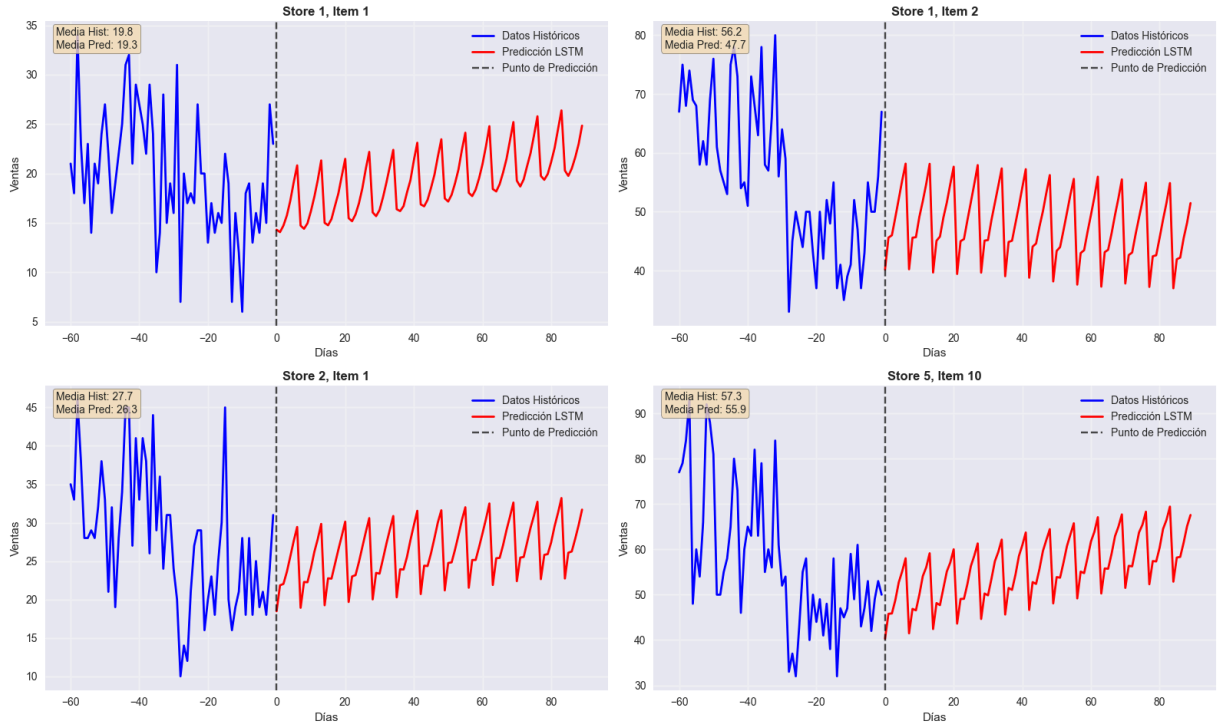
if len(test_idx) > 0 and len(historical) > 0:
    prediction = predictions[test_idx[0]]

    hist_mean = historical.mean()
    hist_std = historical.std()
    pred_mean = prediction.mean()
    pred_std = prediction.std()

    print(f"\nStore {store}, Item {item}:")
    print(f"  Histórico - Media: {hist_mean:.2f}, Std: {hist_std:.2f}")
    print(f"  Predicción - Media: {pred_mean:.2f}, Std: {pred_std:.2f}")
    print(f"  Diferencia en media: {abs(pred_mean - hist_mean):.2f} ({abs(pred_

```

Predicciones LSTM vs Datos Históricos - Ejemplos Seleccionados



=====

Store 1, Item 1:

Histórico - Media: 19.97, Std: 6.74
 Predicción - Media: 19.26, Std: 2.99
 Diferencia en media: 0.71 (3.5%)

Store 1, Item 2:

Histórico - Media: 53.15, Std: 15.00
 Predicción - Media: 47.72, Std: 5.78
 Diferencia en media: 5.43 (10.2%)

Store 2, Item 1:

Histórico - Media: 28.17, Std: 8.68
 Predicción - Media: 26.25, Std: 3.62
 Diferencia en media: 1.92 (6.8%)

Store 5, Item 10:

Histórico - Media: 55.51, Std: 15.56
 Predicción - Media: 55.90, Std: 6.81
 Diferencia en media: 0.39 (0.7%)

10. Interpretabilidad del modelo

In [13]: `import shap``shap.initjs()`

```
In [14]: rng = np.random.default_rng(16)
bg_size = min(200, len(X_train_fast))
exp_size = min(200, len(X_val_fast))

bg_idx = rng.choice(len(X_train_fast), size=bg_size, replace=False)
exp_idx = rng.choice(len(X_val_fast), size=exp_size, replace=False)

background = X_train_fast[bg_idx]
X_exp = X_val_fast[exp_idx]

background.shape, X_exp.shape
```

Out[14]: ((200, 30), (200, 30))

```
In [15]: try:
explainer = shap.Explainer(model, background)
except Exception as e:
    print("Explainer genérico falló, intentando DeepExplainer:", repr(e))
    explainer = shap.DeepExplainer(model, background)

shap_values = explainer(X_exp)
type(shap_values)
```

PermutationExplainer explainer: 201it [19:05, 5.73s/it]

Out[15]: shap._explanation.Explanation

```
In [ ]: def normalize_shap_output(shap_values, X_like, y_like=None):
        """
        Devuelve sv con shape (samples, window, n_features) agregando |SHAP| a través d
        X_like: tensor de entrada (samples, window, n_features) para conocer window y n
        y_like: opcional; si está disponible y es (samples, horizon) nos da el tamaño d
        """

        window = X_like.shape[1]
        n_features = X_like.shape[2]
        horizon_hint = (y_like.shape[1] if (y_like is not None and y_like.ndim >= 2) el

        # shap.Explanation
        if hasattr(shap_values, "values"):
            arr = np.asarray(shap_values.values)
            base_values = getattr(shap_values, "base_values", None)

            if arr.ndim == 3:
                # Puede ser (s, w, n_features) o (s, w, horizon)
                if arr.shape[2] == n_features:
                    sv = arr
                else:
                    # tratamos el último eje como horizonte
                    sv = np.mean(np.abs(arr), axis=2) * np.sign(np.mean(arr, axis=2))
                    sv = sv[..., np.newaxis] # (s, w, 1)
                return sv, base_values

            if arr.ndim == 4:
                # Ejes: (s, w, f, h) o (s, w, h, f). Identificamos cuál es cuál.
                s, a, b, c = arr.shape # s==samples
                # Indices candidatos con tamaños conocidos
                idx_w = [i for i, sz in enumerate(arr.shape) if sz == window][0]
                idx_f = [i for i, sz in enumerate(arr.shape) if sz == n_features]
                idx_h = (idx_f[-1] if len(idx_f)>0 else None)

                # El eje de horizonte es el que no es samples (0), ni window, ni featur
                axes = {0,1,2,3}
                idx_h = list(axes - {0, idx_w} - ({idx_f} if idx_f is not None else set

                # Agregamos sobre horizonte
                agg = np.mean(np.abs(arr), axis=idx_h) * np.sign(np.mean(arr, axis=idx_

                # Reordenamos a (s, w, f) y construimos mapping:
                dims = [0,1,2,3]
                dims.remove(idx_h)
                pos_w = dims.index(idx_w)
                pos_f = dims.index(idx_f) if idx_f is not None else None

                if pos_w != 1: agg = np.moveaxis(agg, pos_w, 1)
                if pos_f is None:
                    agg = agg[..., np.newaxis]
                elif pos_f != 2:
                    agg = np.moveaxis(agg, 2 if pos_w==1 else (1 if pos_w==2 else 2), 2
```

```

        return agg, base_values

# lista por salida (DeepExplainer multioutput): Len=list = horizonte
if isinstance(shap_values, list):
    sv_stack = np.stack([np.asarray(sv_h) for sv_h in shap_values], axis=-1)
    if sv_stack.ndim == 3: # (s,w,h)
        sv_aggr = np.mean(np.abs(sv_stack), axis=-1) * np.sign(np.mean(sv_stack))
        sv_aggr = sv_aggr[..., np.newaxis]
    else: # (s,w,f,h)
        sv_aggr = np.mean(np.abs(sv_stack), axis=-1) * np.sign(np.mean(sv_stack))
    return sv_aggr, None

# ndarray crudo
arr = np.asarray(shap_values)
if arr.ndim == 4:
    s, a, b, c = arr.shape
    idx_w = [i for i, sz in enumerate(arr.shape) if sz == window][0]
    idx_f = [i for i, sz in enumerate(arr.shape) if sz == n_features]
    idx_f = (idx_f[-1] if len(idx_f) > 0 else None)
    axes = {0, 1, 2, 3}
    idx_h = list(axes - {0, idx_w} - ({idx_f} if idx_f is not None else set()))
    agg = np.mean(np.abs(arr), axis=idx_h) * np.sign(np.mean(arr, axis=idx_h))
    dims = [0, 1, 2, 3]; dims.remove(idx_h)
    pos_w = dims.index(idx_w)
    if pos_w != 1: agg = np.moveaxis(agg, pos_w, 1)
    if idx_f is None: agg = agg[..., np.newaxis]
    return agg, None

if arr.ndim == 3:
    # (s,w,f) o (s,w,h)
    if arr.shape[2] == n_features:
        return arr, None
    sv_aggr = np.mean(np.abs(arr), axis=2) * np.sign(np.mean(arr, axis=2))
    sv_aggr = sv_aggr[..., np.newaxis]
    return sv_aggr, None

raise ValueError("Forma de shap_values no reconocida:", arr.shape)

```

```

In [ ]: # Tomamos pistas del tamaño de ventana, #features y horizonte:
window      = X_exp.shape[1]
n_features  = X_exp.shape[2] if X_exp.ndim == 3 else 1
try:
    horizon_hint = int(model.output_shape[-1]) # p.ej., 90
except Exception:
    horizon_hint = y_train_fast.shape[1] if 'y_train_fast' in globals() and y_train

arr = np.asarray(sv)

# (s, w, h) -> agrega sobre h
if arr.ndim == 3 and arr.shape[1] == window and (horizon_hint is not None and arr.s
    arr = np.mean(np.abs(arr), axis=2, keepdims=True) * np.sign(np.mean(arr, axis=2)

# (s, w, f, h) o (s, w, h, f) -> agrega sobre h
elif arr.ndim == 4:
    # identifica el eje de h como el que coincide con horizon_hint (si lo tenemos)
    axes = list(range(arr.ndim))

```

```

if horizon_hint is not None and horizon_hint in arr.shape:
    h_axis = arr.shape.index(horizon_hint)
else:
    # si no sabemos h, tomamos el que no sea sample(0) ni window(1) ni (posible
    candidate = set(axes) - {0, 1}
    # intenta localizar feature por n_features (1)
    f_axes = [i for i,s in enumerate(arr.shape) if s == n_features]
    if f_axes:
        candidate = candidate - {f_axes[-1]}
        h_axis = list(candidate)[0]
    # agrega sobre h
    arr = np.mean(np.abs(arr), axis=h_axis) * np.sign(np.mean(arr, axis=h_axis))

    # Ahora arr tiene 3 ejes (s, ?, ?) -> movemos para que quede (s, w, f)
    # queremos que el eje 1 sea window
    if arr.shape[1] != window:
        # encuentra dónde está window y muévelo al eje 1
        w_axis = [i for i,s in enumerate(arr.shape) if s == window][0]
        arr = np.moveaxis(arr, w_axis, 1)

    # si falta eje feature, añádelo
    if arr.ndim == 2:
        arr = arr[..., np.newaxis]

    # Si nos quedó (s, w) añade eje feature
    if arr.ndim == 2:
        arr = arr[..., np.newaxis]

    # Verifica forma final
    sv = arr
    print("sv listo para plots:", sv.shape) # esperado: (samples, window, 1)

```

sv listo para plots: (200, 30, 1)

```

In [ ]: sv = np.asarray(sv)
        X_exp = np.asarray(X_exp)

        if sv.ndim == 2:
            sv = sv[..., np.newaxis]
        if X_exp.ndim == 2:
            X_exp = X_exp[..., np.newaxis]

        assert sv.ndim == 3 and X_exp.ndim == 3, f"Shapes inesperados: sv={sv.shape}, X_exp={X_exp.shape}"
        assert sv.shape[1] == X_exp.shape[1], "La longitud de ventana no coincide entre sv y X_exp"

        sv_lag = sv[:, :, 0]
        X_lag = X_exp[:, :, 0]
        sv_lag = sv_lag[:, ::-1]
        X_lag = X_lag[:, ::-1]

        lag_names = [f"t-{k}" for k in range(1, sv_lag.shape[1] + 1)]

```

```

In [ ]: shap.summary_plot(
        sv_lag, X_lag,
        feature_names=lag_names,
        plot_type='bar',

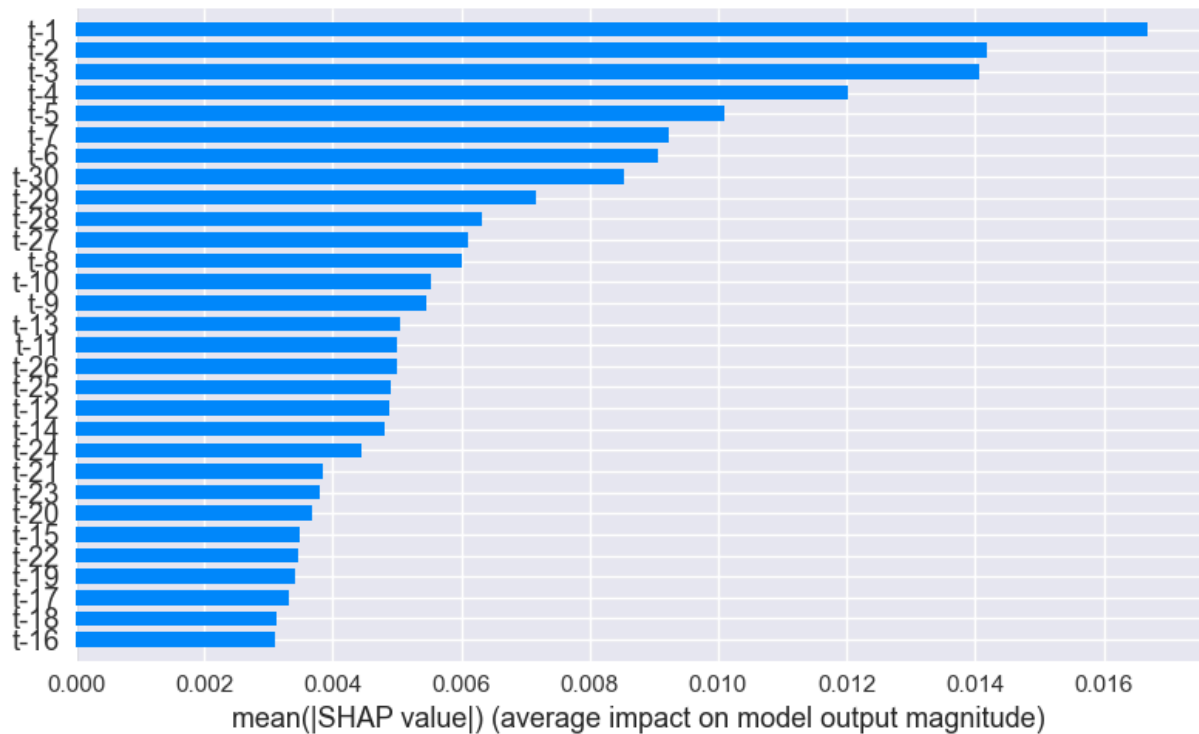
```



```

max_display=sv_lag.shape[1],
plot_size=(8,5)
)

```



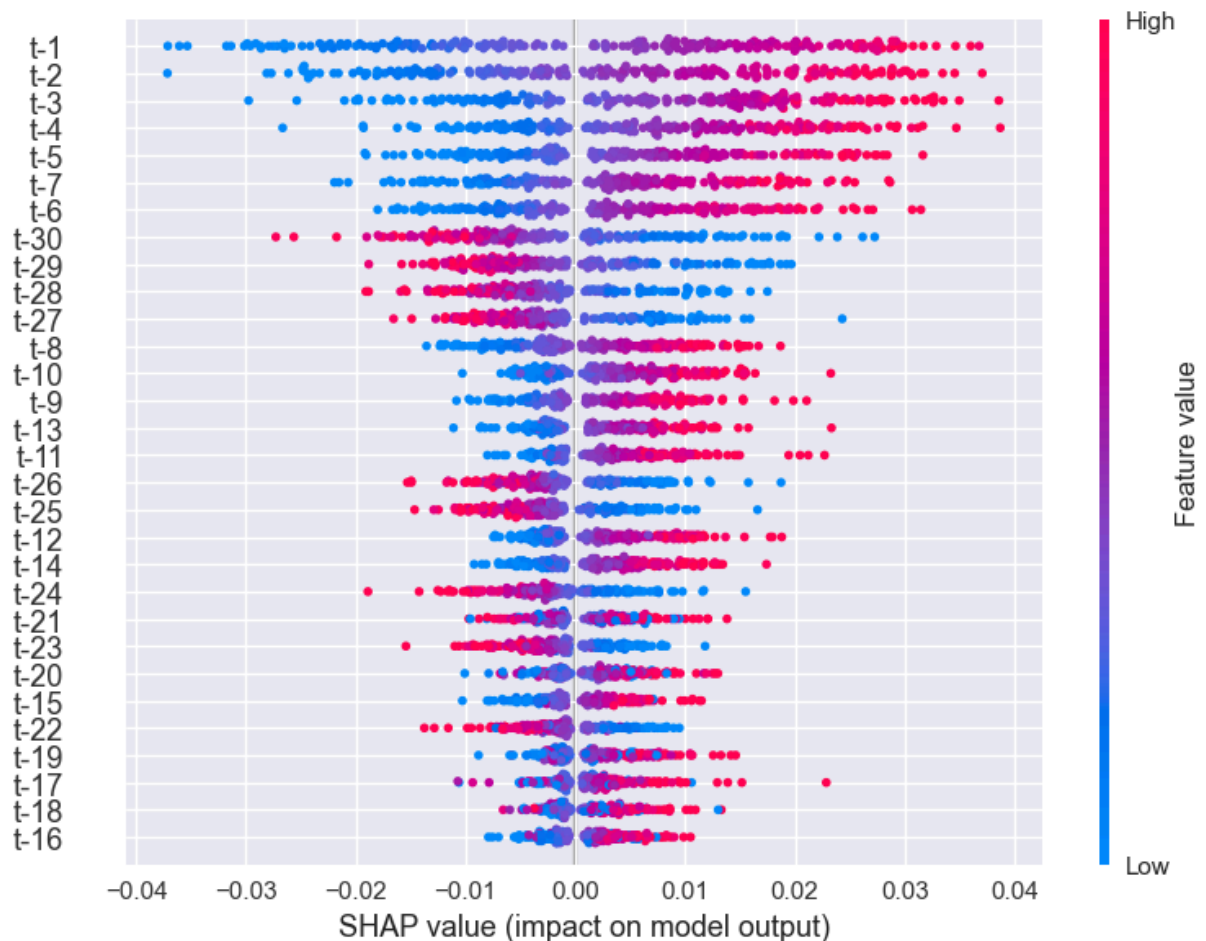
En el summary plot de SHAP se observa que los lags más recientes (t-1 a t-6) concentran la mayor importancia promedio. Esto indica que el LSTM fundamenta sus pronósticos principalmente en la dinámica inmediata de la serie. Cuando las ventas recientes son altas, la predicción tiende a subir; cuando son bajas, tiende a bajar. Además, aparece un pico marcado en t-7, consistente con una estacionalidad semanal (el valor de hace una semana aporta información adicional), y un bloque relevante en t-30...t-28, que sugiere un patrón mensual / de cuatro semanas que el modelo también aprovecha. En contraste, los lags intermedios (t-8...t-27) aportan menos señal, lo que es coherente con una caída de influencia conforme aumenta la distancia temporal, salvo en esos puntos estacionales.

Esta lectura explica el comportamiento observado en las series predichas, dado que el modelo mantiene bien el nivel promedio y suaviza la variabilidad del histórico porque prioriza señales estables y repetitivas, mitigando picos aislados o ruido. Esto implica que el modelo reacciona rápido a cambios recientes (útil a corto plazo) y sostiene patrones semanales/mensuales a mediano plazo.

```

In [32]: # Beeswarm
shap.summary_plot(
    sv_lag, X_lag,
    feature_names=lag_names,
    max_display=sv_lag.shape[1],
    plot_size=(8,6)
)

```



Aquí, el beeswarm muestra para cada lag la dirección y magnitud de su aporte. En los lags recientes (t-1...t-6) se ve un patrón claro, ya que los puntos rojos (ventas altas en ese lag) se acumulan a la derecha empujan la predicción hacia arriba; los azules (ventas bajas) se van a la izquierda y tiran la predicción hacia abajo. La nube es ancha en estos lags, indica que la proximidad temporal es la principal fuente de variación en las predicciones.

Destaca t-7 con el mismo comportamiento (rojo a la derecha, azul a la izquierda), lo que respalda una estacionalidad semanal. También aparecen contribuciones consistentes en t-28...t-30, compatibles con un ciclo mensual/4-semanas donde los valores altos en esos lags tienden a elevar la predicción. En cambio, varios lags intermedios (t-8...t-27) concentran puntos alrededor de 0, indicando bajo aporte marginal una vez que el modelo ya consideró los lags más cercanos.

En general, el gráfico demuestra que el LSTM se apoya en la proximidad temporal y en ciclos semanales/mensuales. Esto explica por qué el modelo mantiene bien el nivel y suaviza ruido, dado que prioriza señales repetitivas y recientes.

Teoría

Responda claramente y con una extensión adecuada las siguientes preguntas:

1. ¿Cuál es el problema del gradiente de fuga en las redes LSTM y cómo afecta la efectividad de LSTM para el pronóstico de series temporales?

- El problema del gradiente de fuga ocurre cuando los gradientes se hacen muy pequeños durante el proceso de backpropagation, lo que dificulta que la red aprenda dependencias a largo plazo. Aunque las LSTM fueron diseñadas justamente para mitigar este problema gracias a sus compuertas, todavía puede presentarse si la secuencia es demasiado larga o si el modelo no está bien ajustado. En el contexto de series temporales, esto puede provocar que la red se enfoque solo en patrones recientes y no logre captar tendencias más prolongadas, reduciendo la calidad del pronóstico.

2. ¿Cómo se aborda la estacionalidad en los datos de series temporales cuando se utilizan LSTM para realizar pronósticos y qué papel juega la diferenciación en el proceso?

- La estacionalidad se aborda normalmente preparando los datos antes de entrenar el modelo. Una forma común es aplicar transformaciones como la diferenciación (restar un valor con respecto a su equivalente en un periodo anterior). Esto ayuda a que la red trabaje con datos más "estacionarios", eliminando las repeticiones cíclicas y dejando en evidencia las variaciones reales que debe aprender. Así, el LSTM no tiene que ir redescubriendo patrones estacionales, sino que puede concentrarse en aprender la dinámica de la serie.

3. ¿Cuál es el concepto de "tamaño de ventana" en el pronóstico de series temporales con LSTM y cómo afecta la elección del tamaño de ventana a la capacidad del modelo para capturar patrones a corto y largo plazo?

- El tamaño de ventana se refiere al número de observaciones pasadas que se toman como entrada para predecir el siguiente valor. Elegir este tamaño es clave, dado que si la ventana es muy corta, el modelo solo aprenderá patrones locales y puede perder información importante de más atrás. Si la ventana es demasiado larga, el modelo tendrá más información para detectar tendencias a largo plazo, pero también será más difícil de entrenar y puede introducir ruido. En la práctica, se busca un equilibrio para que el modelo capture tanto los patrones inmediatos como los de mayor duración.

Desarrollo del Task 2

1. Despot, I., & Arif, A. (2025, 16 enero). Time-Series Forecasting: Definition, Methods, and Applications. TigerData Blog. <https://www.tigerdata.com/blog/what-is-time-series-forecasting>

2. Q3 Technologies. (2024, 6 agosto). How LSTM Networks are Revolutionizing Time Series Forecasting. <https://www.q3tech.com/blogs/lstm-time-series-forecasting/>