

# APRENDIZAJE PROFUNDO Y SERIES TEMPORALES

por: Marco Cajamarca, Adrian Campoverde y Pablo Bravo

A continuación, se desarrollará una función que utiliza una para predecir la demanda de productos a partir de datos históricos reales. Este enfoque permitirá anticipar las necesidades de inventario y tomar decisiones informadas basadas en el comportamiento temporal de las ventas.

```
source ~/tf-env/bin/activate mlflow ui --backend-store-uri  
file:mlruns
```

```
In [ ]: # Imports  
import pandas as pd  
from pathlib import Path  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.metrics import mean_squared_error, mean_absolute_error  
import matplotlib.pyplot as plt  
import numpy as np  
from datetime import datetime, timedelta  
from sklearn.model_selection import train_test_split  
from sklearn.preprocessing import LabelEncoder  
import tensorflow as tf  
from tensorflow import keras  
from tensorflow.keras.layers import Embedding, Concatenate  
from IPython.display import display  
import mlflow  
import joblib  
import os
```

## Configuracion de mlflow

```
In [ ]: notebook_dir = os.path.dirname(os.path.abspath(__file__)) if '__file__'  
mlflow_tracking_dir = os.path.join(notebook_dir, 'mlruns')  
  
os.makedirs(mlflow_tracking_dir, exist_ok=True)  
  
mlflow.set_tracking_uri(f'file://{mlflow_tracking_dir}')  
  
experiment_name = 'stock_demand_forecasting'  
mlflow.set_experiment(experiment_name)  
  
# Obtener información del experimento  
experiment = mlflow.get_experiment_by_name(experiment_name)  
print(f"Experimento: {experiment_name}")  
print(f"Tracking URI: {mlflow.get_tracking_uri()}")  
print(f"Directorio MLflow: {mlflow_tracking_dir}")
```

```
print(f"Experiment ID: {experiment.experiment_id}")

mlflow.keras.autolog(log_models=True, log_datasets=True, disable=False,

print("\n✅ MLflow configurado correctamente")
print("💡 Para ver la UI, ejecuta en terminal: mlflow ui --backend-store
```

## configuracion básica de tensorflow para uso optimo de gpu

```
In [ ]: gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for g in gpus:
            tf.config.experimental.set_memory_growth(g, True)
    except Exception as e:
        print("GPU config error:", e)

# Habilitar XLA JIT para acelerar operaciones
tf.config.optimizer.set_jit(True)

# Ajustar hilos para aprovechar CPU en preprocesamiento
ncpu = os.cpu_count() or 4
tf.config.threading.set_intra_op_parallelism_threads(ncpu)
tf.config.threading.set_inter_op_parallelism_threads(ncpu)
```

## Fase 1:

### Análisis y Preparación del Dataset

Objetivo: Cargar, explorar y preparar el dataset para el entrenamiento de modelos de aprendizaje profundo. En esta fase se realiza una inspección inicial de los datos, identificación de valores faltantes, análisis de las variables y creación de características temporales relevantes para el modelado de series temporales.

```
In [ ]: # Cargar el archivo CSV real
ruta = Path('practica_completo.csv')
df = pd.read_csv(ruta)

# Mostrar columnas y primeras filas
print("Columnas:", df.columns.tolist())
display(df.head())

# Tipos de datos y valores faltantes
print("\nTipos de datos:")
display(df.dtypes)
print("\nValores faltantes por columna:")
display(df.isnull().sum())

# Mostrar rango de fechas del dataset
print("Primer registro:", df['created_at'].min())
print("Último registro:", df['created_at'].max())
```

```
In [ ]: # Eliminar filas con valores nulos críticos
df = df.dropna(subset=['created_at', 'product_id', 'salida'])

# Eliminar duplicados si existen
df = df.drop_duplicates()

# Convertir fechas a datetime
df['created_at'] = pd.to_datetime(df['created_at'])
df['last_order_date'] = pd.to_datetime(df['last_order_date'], errors='coerce')

# Revisar valores extremos en ventas
print("Ventas (salida) - resumen estadístico:")
display(df['salida'].describe())
```

```
In [ ]: # Crear variables temporales
df['dia_semana'] = df['created_at'].dt.dayofweek
df['mes'] = df['created_at'].dt.month
df['fin_semana'] = df['dia_semana'].isin([5,6]).astype(int)

# Feriados (ajusta según tu país)
feriados = [
    '2024-01-01', '2024-02-12', '2024-02-13', '2024-03-29', '2024-05-01',
    '2024-05-24', '2024-08-10', '2024-10-09', '2024-11-02', '2024-11-03'
]
df['fecha'] = df['created_at'].dt.date.astype(str)
df['feriado'] = df['fecha'].isin(feriados).astype(int)

# Antigüedad del producto (en días)
df['antigüedad_producto'] = (df['created_at'] - df.groupby('product_id')['created_at'].transform('min')).dt.days

# Ratio vendida/stock (si aplica)
df['ratio_vendida_stock'] = df['salida'] / (df['quantity_on_hand'] + 1e-6)

# Mostrar ejemplo
display(df.head())
```

```
In [ ]: # Variables a usar en la secuencia
features = ['salida', 'dia_semana', 'mes', 'fin_semana', 'feriado', 'quantity_on_hand']
ventana = 7 # días de historial

def crear_ventanas_con_producto(df, features, ventana=7):
    """Crea secuencias manteniendo el product_id y prediciendo stock"""
    X_seq, y_stock, product_ids, fechas = [], [], [], []

    for product_id, grupo in df.groupby('product_id'):
        grupo = grupo.sort_values('created_at')
        datos = grupo[features].values
        stock = grupo['quantity_on_hand'].values
        fechas_grupo = grupo['created_at'].values

        for i in range(len(datos) - ventana):
            X_seq.append(datos[i:i+ventana])
            y_stock.append(stock[i+ventana])
            product_ids.append(product_id)
            fechas.append(fechas_grupo[i+ventana])

    return np.array(X_seq), np.array(y_stock), np.array(product_ids), np.array(fechas)

X_seq, y_seq, product_ids, fechas = crear_ventanas_con_producto(df, features, ventana=7)
```

```
print("Forma de X_seq:", X_seq.shape)
print("Forma de y_seq (stock):", y_seq.shape)
print("Productos únicos:", len(np.unique(product_ids)))
```

## Fase 2:

### Desarrollo y Entrenamiento del Modelo


#### Objetivo:

Implementar un modelo de Deep Learning usando RNN (por ejemplo, LSTM o GRU) para predecir la demanda futura.

Limpieza y normalización de datos

```
In [ ]: from sklearn.preprocessing import RobustScaler
        from scipy import stats

n_samples, ventana, n_features = X_seq.shape

# Filtrar productos con stock muy bajo o cero
print(f"

```

```

outliers_removed = len(y_train_log_raw) - len(y_train_log)
print(f"\n📊 Limpieza de outliers:")
print(f"    Outliers removidos: {outliers_removed} ({outliers_removed/len(y_train_log_raw):.1f})")
print(f"    Train: {len(y_train_log)}, Val: {len(y_val_log)}")
print(f"    Stock (escala log) - Min: {y_train_log.min():.3f}, Max: {y_train_log.max():.3f}")
print(f"    Stock (escala real) - Min: {np.expml(y_train_log).min():.1f}, Max: {np.expml(y_train_log).max():.1f}")

```

## Creación del modelo con embedding de productos

```

In [ ]: n_productos = len(df['product_id'].unique())
        embedding_dim = 32 # Baseline dimension

# Input de secuencia temporal
seq_input = keras.layers.Input(shape=(X_train.shape[1], X_train.shape[2]))
product_input = keras.layers.Input(shape=(1,), name='product_id')

# Embedding del producto
product_embedding = Embedding(
    input_dim=n_productos + 1,
    output_dim=embedding_dim,
    name='product_embedding'
)(product_input)
product_embedding = keras.layers.Flatten()(product_embedding)

# Procesamiento de secuencia temporal - arquitectura baseline simplificada
x = keras.layers.Bidirectional(keras.layers.GRU(
    128,
    return_sequences=True,
    kernel_regularizer=keras.regularizers.l2(0.0001)
))(seq_input)
x = keras.layers.BatchNormalization()(x)
x = keras.layers.Dropout(0.3)(x)

x = keras.layers.Bidirectional(keras.layers.GRU(
    64,
    return_sequences=False,
    kernel_regularizer=keras.regularizers.l2(0.0001)
))(x)
x = keras.layers.BatchNormalization()(x)
x = keras.layers.Dropout(0.2)(x)

# Combinar información de producto + secuencia temporal
combined = Concatenate()([x, product_embedding])

# Capas densas
x = keras.layers.Dense(128, activation='relu', kernel_regularizer=keras.regularizers.l2(0.0001))(combined)
x = keras.layers.BatchNormalization()(x)
x = keras.layers.Dropout(0.2)(x)
x = keras.layers.Dense(64, activation='relu')(x)

# Output en escala logarítmica
output = keras.layers.Dense(1, name='stock_log_predicho')(x)

model = keras.Model(inputs=[seq_input, product_input], outputs=output)

```

```

# Custom weighted loss que da más importancia a valores altos de stock
@tf.function
def weighted_huber_loss(y_true, y_pred, delta=1.0, high_stock_threshold=5.3):
    """
    Huber loss con pesos adaptativos basados en el valor del stock (en escala log)
    high_stock_threshold en escala log:  $\log_{10}(200) \approx 5.3$ 
    """

    # Calcular error
    error = y_true - y_pred
    abs_error = tf.abs(error)

    # Huber loss básico
    quadratic = tf.minimum(abs_error, delta)
    linear = abs_error - quadratic
    loss = 0.5 * quadratic**2 + delta * linear

    # Calcular pesos adaptativos: mayor peso para stock alto
    # Stock alto (log >= 5.3): peso 2.5x
    # Stock medio (log 3.9-5.3): peso 1.5x
    # Stock bajo (log < 3.9): peso 1.0x
    weights = tf.where(
        y_true >= high_stock_threshold,
        2.5, # Stock >= 200 unidades ( $\log_{10}(200) \approx 5.3$ )
        tf.where(
            y_true >= 3.9, # Stock >= 50 unidades ( $\log_{10}(50) \approx 3.9$ )
            1.5,
            1.0
        )
    )

    # Aplicar pesos a la loss
    weighted_loss = loss * weights

    return tf.reduce_mean(weighted_loss)

# Learning rate schedule más conservador
initial_learning_rate = 0.001
lr_schedule = keras.optimizers.schedules.CosineDecayRestarts(
    initial_learning_rate=initial_learning_rate,
    first_decay_steps=100,
    t_mul=2.0,
    m_mul=0.9,
    alpha=0.0001
)

optimizer = keras.optimizers.Adam(
    learning_rate=lr_schedule,
    clipnorm=1.0
)

# Compilar con loss personalizada
model.compile(
    optimizer=optimizer,
    loss=weighted_huber_loss,
    metrics=['mae', keras.metrics.RootMeanSquaredError(name='rmse')]
)

# Callbacks
early_stopping = keras.callbacks.EarlyStopping(
    monitor='val_loss',

```

```

        patience=25,
        restore_best_weights=True,
        verbose=1,
        min_delta=1e-5,
        mode='min'
    )

    model_checkpoint = keras.callbacks.ModelCheckpoint(
        'best_model_v2.keras',
        monitor='val_loss',
        save_best_only=True,
        save_weights_only=False,
        mode='min',
        verbose=1
    )

    tensorboard = keras.callbacks.TensorBoard(
        log_dir='./logs',
        histogram_freq=1,
        write_graph=True,
        update_freq='epoch'
    )

    run_params = {
        'ventana': ventana,
        'n_features': n_features,
        'n_productos': n_productos,
        'embedding_dim': embedding_dim,
        'architecture': 'Baseline_BiGRU_WeightedLoss',
        'total_params': model.count_params(),
        'optimizer': 'adam_with_cosine_decay',
        'initial_learning_rate': initial_learning_rate,
        'loss_function': 'weighted_huber',
        'loss_weights': 'high_stock_2.5x_medium_1.5x_low_1.0x',
        'target_transform': 'loglp',
        'output_activation': 'linear',
        'scaler': 'RobustScaler',
        'outlier_removal': 'z_score_3.5',
        'min_stock_threshold': 10,
        'outliers_removed': outliers_removed,
        'batch_size': 256,
        'epochs': 150,
        'prediction_target': 'quantity_on_hand_log',
        'enhancements': 'weighted_loss_for_high_stock+batch_norm'
    }

    with mlflow.start_run(run_name=f'stock_weighted_{datetime.now().strftime("%Y-%m-%d_%H-%M-%S")}') as run:
        mlflow.log_params(run_params)
        mlflow.log_param('train_samples', len(X_train))
        mlflow.log_param('val_samples', len(X_val))

        model_summary = []
        model.summary(print_fn=lambda x: model_summary.append(x))
        model_summary_str = '\n'.join(model_summary)

        with open('model_summary.txt', 'w') as f:
            f.write(model_summary_str)
        mlflow.log_artifact('model_summary.txt')

        print(f"\nRun ID: {run.info.run_id}")

```

```

print(f"Estrategia: Weighted Loss - Stock alto (>=200) peso 2.5x, m

history = model.fit(
    [X_train, pid_train],
    y_train_log,
    validation_data=([X_val, pid_val], y_val_log),
    epochs=run_params['epochs'],
    batch_size=run_params['batch_size'],
    callbacks=[early_stopping, model_checkpoint, tensorboard],
    verbose=1,
    shuffle=True
)

scaler_path = 'scaler.joblib'
joblib.dump(scaler, scaler_path)
mlflow.log_artifact(scaler_path)
mlflow.log_artifact('product_encoder.joblib')

if os.path.exists('best_model_v2.keras'):
    mlflow.log_artifact('best_model_v2.keras')

fig, axes = plt.subplots(2, 2, figsize=(15, 10))

axes[0, 0].plot(history.history['loss'], label='Train Loss', linewidth=2)
axes[0, 0].plot(history.history['val_loss'], label='Val Loss', linewidth=2)
axes[0, 0].set_title('Model Loss (Weighted Huber en escala log)')
axes[0, 0].set_xlabel('Epoch')
axes[0, 0].set_ylabel('Loss')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

axes[0, 1].plot(history.history['mae'], label='Train MAE', linewidth=2)
axes[0, 1].plot(history.history['val_mae'], label='Val MAE', linewidth=2)
axes[0, 1].set_title('Mean Absolute Error (escala log)')
axes[0, 1].set_xlabel('Epoch')
axes[0, 1].set_ylabel('MAE')
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3)

axes[1, 0].plot(history.history['rmse'], label='Train RMSE', linewidth=2)
axes[1, 0].plot(history.history['val_rmse'], label='Val RMSE', linewidth=2)
axes[1, 0].set_title('RMSE (escala log)')
axes[1, 0].set_xlabel('Epoch')
axes[1, 0].set_ylabel('RMSE')
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

axes[1, 1].text(0.5, 0.5,
                f'GRU Bidireccional Baseline\n{n_productos} productos',
                ha='center', va='center', transform=axes[1, 1].trans)
axes[1, 1].set_title('Configuración del Modelo')
axes[1, 1].axis('off')

plt.tight_layout()
plt.savefig('training_history.png', dpi=150, bbox_inches='tight')
mlflow.log_artifact('training_history.png')
plt.show()

```

```

In [ ]: # Evaluar el modelo
print("🐼 Evaluando modelo en conjunto de validación...")

```

```

y_pred_log = model.predict([X_val, pid_val], batch_size=256)
y_pred_log = np.asarray(y_pred_log).ravel().astype(np.float32)

# Convertir de escala logarítmica a escala real
y_pred = np.expml(y_pred_log) # Inversa de loglp
y_val_real = np.expml(y_val_log)

# Asegurar que no hay predicciones negativas
y_pred = np.maximum(y_pred, 0)

# Calcular métricas en escala real
rmse = np.sqrt(mean_squared_error(y_val_real, y_pred))
mae = mean_absolute_error(y_val_real, y_pred)
median_ae = np.median(np.abs(y_val_real - y_pred))

# MAPE solo para stock >= 10
threshold = 10
mask_mape = y_val_real >= threshold
if np.sum(mask_mape) > 0:
    mape = np.mean(np.abs((y_val_real[mask_mape] - y_pred[mask_mape]) /
else:
    mape = 0.0

# R²
r2 = 1 - (np.sum((y_val_real - y_pred)**2) / np.sum((y_val_real - y_val

# Métricas por rangos de stock
mask_bajo = (y_val_real >= 10) & (y_val_real < 50)
mask_medio = (y_val_real >= 50) & (y_val_real < 200)
mask_alto = y_val_real >= 200

print(f"\n📊 Métricas Finales (Predicción de Stock):")
print(f"  RMSE: {rmse:.2f}")
print(f"  MAE: {mae:.2f}")
print(f"  Median AE: {median_ae:.2f}")
print(f"  MAPE (stock >= {threshold}): {mape:.2f}%")
print(f"  R²: {r2:.4f}")
print(f"\n📈 Precisión por rango:")
if np.sum(mask_bajo) > 0:
    mape_bajo = np.mean(np.abs((y_val_real[mask_bajo] - y_pred[mask_bajo]
    print(f"  Stock bajo (10-50): MAPE={mape_bajo:.2f}% | N={np.sum(masl
if np.sum(mask_medio) > 0:
    mape_medio = np.mean(np.abs((y_val_real[mask_medio] - y_pred[mask_me
    print(f"  Stock medio (50-200): MAPE={mape_medio:.2f}% | N={np.sum(r
if np.sum(mask_alto) > 0:
    mape_alto = np.mean(np.abs((y_val_real[mask_alto] - y_pred[mask_alto
    print(f"  Stock alto (>=200): MAPE={mape_alto:.2f}% | N={np.sum(masl

print(f"\n🔍 Análisis de predicciones:")
print(f"  Rango predicciones: [{y_pred.min():.1f}, {y_pred.max():.1f}]")
print(f"  Rango reales: [{y_val_real.min():.1f}, {y_val_real.max():.1f}]")
print(f"  Media predicciones: {y_pred.mean():.1f}")
print(f"  Media reales: {y_val_real.mean():.1f}")

try:
    with mlflow.start_run(run_id=run.info.run_id):
        mlflow.log_metric('final_rmse', float(rmse))
        mlflow.log_metric('final_mae', float(mae))
        mlflow.log_metric('final_median_ae', float(median_ae))
        mlflow.log_metric('final_mape', float(mape))

```

```

mlflow.log_metric('final_r2', float(r2))

fig, axes = plt.subplots(2, 2, figsize=(16, 10))

# Predicción vs Real
axes[0, 0].plot(y_val_real[:100], 'o-', label='Real', alpha=0.7)
axes[0, 0].plot(y_pred[:100], 's-', label='Predicción', alpha=0.7)
axes[0, 0].set_title('Predicción vs Valor Real (Stock) - Primer Ejemplo')
axes[0, 0].set_xlabel('Ejemplo')
axes[0, 0].set_ylabel('Stock (unidades)')
axes[0, 0].legend()
axes[0, 0].grid(True, alpha=0.3)

# Scatter plot
axes[0, 1].scatter(y_val_real, y_pred, alpha=0.3, s=10)
axes[0, 1].plot([y_val_real.min(), y_val_real.max()], [y_val_real.min(), y_val_real.max()], 'r--', lw=2, label='Predicción Perfecta')
axes[0, 1].set_xlabel('Stock Real')
axes[0, 1].set_ylabel('Stock Predicho')
axes[0, 1].set_title(f'Correlación (R²={r2:.3f}, MAPE={mape:.1f}%)')
axes[0, 1].legend()
axes[0, 1].grid(True, alpha=0.3)

# Distribución de residuos
residuos = y_val_real - y_pred
axes[1, 0].hist(residuos, bins=50, edgecolor='black', alpha=0.7)
axes[1, 0].axvline(x=0, color='r', linestyle='--', linewidth=2, label=f'Mediana=0')
axes[1, 0].axvline(x=np.median(residuos), color='green', linestyle='--', linewidth=2, label=f'Mediana={np.median(residuos):.1f}')
axes[1, 0].set_title(f'Distribución de Residuos (μ={residuos.mean():.1f})')
axes[1, 0].set_xlabel('Residuo (Real - Predicción)')
axes[1, 0].set_ylabel('Frecuencia')
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

# Error porcentual absoluto
error_pct = np.abs((y_val_real - y_pred) / (y_val_real + 1)) * 100
error_pct_filtrado = error_pct[error_pct < 100]
axes[1, 1].hist(error_pct_filtrado, bins=50, edgecolor='black', alpha=0.7)
axes[1, 1].axvline(x=np.median(error_pct_filtrado), color='darkblue', linestyle='--', linewidth=2, label=f'Mediana={np.median(error_pct_filtrado):.1f}')
axes[1, 1].axvline(x=np.mean(error_pct_filtrado), color='darkblue', linestyle='--', linewidth=2, label=f'Media={np.mean(error_pct_filtrado):.1f}')
axes[1, 1].set_title(f'Error Porcentual Absoluto (< 100%)')
axes[1, 1].set_xlabel('Error %')
axes[1, 1].set_ylabel('Frecuencia')
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('prediction_analysis.png', dpi=150, bbox_inches='tight')
mlflow.log_artifact('prediction_analysis.png')
plt.show()

print(f"\n✅ Métricas guardadas en MLflow (Run ID: {run.info.run_id})")

except Exception as e:
    print(f"⚠️ Error al registrar métricas: {e}")

```

# Función de Predicción para Uso en Producción

```
In [ ]: import numpy as np
import pandas as pd
from datetime import datetime, timedelta

def predecir_stock_futuro(model, scaler, label_encoder, df, product_id,
    """
    Predice el stock para un producto en una fecha específica

    Args:
        model: Modelo entrenado (predice en escala log)
        scaler: RobustScaler ajustado
        label_encoder: LabelEncoder para product_ids
        df: DataFrame con datos históricos
        product_id: ID del producto
        fecha_objetivo: Fecha para la predicción (datetime o string)
        ventana: Días de historial necesarios (default: 7)

    Returns:
        dict con product_id, fecha, stock_predicho, stock_actual
    """
    if isinstance(fecha_objetivo, str):
        fecha_objetivo = pd.to_datetime(fecha_objetivo)

    # Filtrar datos del producto
    df_producto = df[df['product_id'] == product_id].sort_values('created_at')

    if len(df_producto) == 0:
        raise ValueError(f"Producto {product_id} no encontrado")

    # Obtener últimos 'ventana' días antes de la fecha objetivo
    datos_recientes = df_producto[df_producto['created_at'] < fecha_objetivo]

    if len(datos_recientes) < ventana:
        raise ValueError(f"No hay suficiente historial para producto {product_id}")

    # Preparar input de secuencia
    X_input = datos_recientes[features].values.reshape(1, ventana, -1)
    X_scaled = scaler.transform(X_input.reshape(-1, len(features))).reshape(1, ventana, -1)

    # Preparar input de product_id
    try:
        product_id_encoded = label_encoder.transform([product_id])[0]
    except ValueError:
        raise ValueError(f"Producto {product_id} no fue visto durante el entrenamiento")

    product_id_array = np.array([[product_id_encoded]])

    # Predecir en escala logarítmica y convertir a escala real
    stock_log_predicho = model.predict([X_scaled, product_id_array], verbose=0)
    stock_predicho = np.expml(stock_log_predicho) # Inversa de loglp

    return {
        'product_id': product_id,
        'fecha': fecha_objetivo,
        'stock_predicho': stock_predicho,
        'stock_actual': None
    }
```

```

        'stock_predicho': max(0, float(stock_predicho)),
        'stock_actual': float(datos_recientes['quantity_on_hand'].iloc[
        'ultima_fecha_datos': datos_recientes['created_at'].iloc[-1]
    }

# Ejemplo de uso
print("\n" + "="*60)
print("EJEMPLO DE PREDICCIÓN")
print("="*60)

# Seleccionar un producto aleatorio
producto_ejemplo = df['product_id'].value_counts().index[0]
fecha_futura = datetime.now() + timedelta(days=3)

try:
    resultado = predecir_stock_futuro(
        model=model,
        scaler=scaler,
        label_encoder=le,
        df=df,
        product_id=producto_ejemplo,
        fecha_objetivo=fecha_futura,
        ventana=ventana
    )

    print(f"\n📦 Producto ID: {resultado['product_id']}")
    print(f"📅 Fecha de predicción: {resultado['fecha'].strftime('%Y-%m-%d')}")
    print(f"📊 Stock actual (último dato): {resultado['stock_actual']:.0f}")
    print(f"🔮 Stock predicho: {resultado['stock_predicho']:.0f} unidades")
    print(f"📅 Última fecha con datos: {resultado['ultima_fecha_datos'].strftime('%Y-%m-%d')}")

    diferencia = resultado['stock_predicho'] - resultado['stock_actual']
    if diferencia > 0:
        print(f"📈 Tendencia: +{diferencia:.0f} unidades")
    else:
        print(f"📉 Tendencia: {diferencia:.0f} unidades")

except Exception as e:
    print(f"⚠️ Error: {e}")

```

## Predicción para Múltiples Productos

```

In [ ]: def predecir_stock_multiple(model, scaler, label_encoder, df, product_ids):
    """
    Predice stock para múltiples productos

    Args:
        product_ids: Lista de product_ids
        fecha_objetivo: Fecha para predicción

    Returns:
        DataFrame con predicciones
    """
    resultados = []

    for pid in product_ids:
        try:
            res = predecir_stock_futuro(model, scaler, label_encoder, df,

```

```

        resultados.append(res)
    except Exception as e:
        print(f"⚠ Error con producto {pid}: {e}")

    return pd.DataFrame(resultados)

# Ejemplo: predecir para los 5 productos más vendidos
top_productos = df['product_id'].value_counts().head(5).index.tolist()
fecha_prediccion = datetime.now() + timedelta(days=7)

print(f"\n🧠 Predicción de stock para {len(top_productos)} productos")
print(f"📅 Fecha objetivo: {fecha_prediccion.strftime('%Y-%m-%d')}")
print("-" * 80)

predicciones_df = predecir_stock_multiple(
    model=model,
    scaler=scaler,
    label_encoder=le,
    df=df,
    product_ids=top_productos,
    fecha_objetivo=fecha_prediccion,
    ventana=7
)

display(predicciones_df)

# Visualización
if len(predicciones_df) > 0:
    fig, ax = plt.subplots(figsize=(12, 6))

    x = np.arange(len(predicciones_df))
    width = 0.35

    ax.bar(x - width/2, predicciones_df['stock_actual'], width, label='Actual')
    ax.bar(x + width/2, predicciones_df['stock_predicho'], width, label='Predicho')

    ax.set_xlabel('Producto ID')
    ax.set_ylabel('Stock')
    ax.set_title(f'Predicción de Stock para {fecha_prediccion.strftime("%Y-%m-%d")}')
    ax.set_xticks(x)
    ax.set_xticklabels(predicciones_df['product_id'], rotation=45)
    ax.legend()
    ax.grid(True, alpha=0.3)

    plt.tight_layout()
    plt.savefig('multi_product_prediction.png', dpi=150, bbox_inches='tight')
    plt.show()

```