Códigos Fuente del Proyecto

Sistema de Predicción de Demanda con LSTM

Compendio de Scripts Python

November 2025

Índice de Scripts

1. Script Principal (Conversión del Cuaderno)

Lógica principal de entrenamiento, optimización y generación de modelos.

`Archivo: fase_final_red_neuronal_converted.py`

```
# !pip install keras_tuner

# @title 1- Cargue de libreias
import matplotlib
matplotlib.use(&#x27;Agg')  # Use non-interactive backend
import tensorflow as tf
import datetime
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import plotly.express as px
from scipy import stats
from scipy.signal import periodogram
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from scipy.stats import entropy
from sklearn.metrics import r2_score
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import grangercausalitytests
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_predict

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow import keras
from keras import layers
import keras_tuner as kt
import math



# Limpiar directorio de resultados
import shutil
shutil.rmtree(&#x27;tuning_results/hyperband_tuning', ignore_errors=True)
shutil.rmtree(&#x27;tuning_results/random_search_tuning', ignore_errors=True)
shutil.rmtree(&#x27;tuning_results/bayesian_optimization_tuning', ignore_errors=True)
shutil.rmtree(&#x27;my_dir', ignore_errors=True)
```

```
shutil.rmtree(&#x27;my_dir_hyperband', ignore_errors=True)

# @title 2- Cargar data set limpia
url = "https://github.com/OscarT231/Proyecto-deep-/raw/refs/heads/main/Base_filtrada.xlsx"
df = pd.read_excel(url)



# @title 2.1 Eliminar columnas
df.columns = df.columns.astype(str).str.strip()
columnas_deseadas = [
    "bodega", "producto", "calificacion_abc",
    "2024-09-01 00:00:00","2024-10-01 00:00:00","2024-11-01 00:00:00","2024-12-01 00:00:00",
    "2025-01-01 00:00:00","2025-02-01 00:00:00","2025-03-01 00:00:00","2025-04-01 00:00:00",
    "2025-05-01 00:00:00","2025-06-01 00:00:00","2025-07-01 00:00:00","2025-08-01 00:00:00"
]

df_sugerido = df[[col for col in columnas_deseadas if col in df.columns]].copy()
df_sugerido = df_sugerido[~df_sugerido["calificacion_abc"].isin(["O", "N"])].copy()
df_sugerido.head()

df= df_sugerido

#@title 2.2 Ajustar el dataset para formato LSTM
# Identificar columnas de fecha (todas excepto bodega, producto, calificación)
id_cols = ["bodega", "producto", "calificacion_abc"]
date_cols = [c for c in df.columns if c not in id_cols]

# Convertir wide → long
df_long = df.melt(id_vars=id_cols,
                  value_vars=date_cols,
                  var_name="fecha",
                  value_name="stock_solicitado")

# Convertir fecha a datetime (formato dd/mm/yyyy)
df_long[&#x27;fecha'] = pd.to_datetime(df_long['fecha'])

# Ordenar correctamente
df_long = df_long.sort_values(["bodega", "producto", "fecha"])

df_long.head()

"""
- Se crearán dos bases de datos para los productos top y media categoría
- Se analizarán las bodegas como una sola para después de entrenar y tener el modelo
    utilizar probabilidad (muestreo estratificado) para decidir cuantas unidades van a cada bodega
"""
# @title 2.3 creación de los dos dataset
#df del producto A
```

```
df_A = df_long[df_long[&#x27;producto'].isin(['P9933'])].dropna().reset_index(drop=True)



#df del producto B
df_B = df_long[df_long[&#x27;producto'].isin(['P2417'])]



df_A["fecha"] = pd.to_datetime(df_A["fecha"])
df_B["fecha"] = pd.to_datetime(df_B["fecha"])

df_A.head()

#@title 3- Creación de diccionarios
#DICCIONARIO A
dict_A = {}

for b in df_A["bodega"].unique():
    df_b = df_A[df_A["bodega"] == b][["fecha", "stock_solicitado"]].copy()
    df_b = df_b.sort_values("fecha")

    # Saltar bodegas sin registros útiles
    if df_b["stock_solicitado"].sum() == 0:
        continue

    dict_A[b] = df_b

# DICCIONARIO B
dict_B = {}

for b in df_B["bodega"].unique():
    df_b = df_B[df_B["bodega"] == b][["fecha", "stock_solicitado"]].copy()
    df_b = df_b.sort_values("fecha")

    if df_b["stock_solicitado"].sum() == 0:
        continue

    dict_B[b] = df_b

#@title 4- Normalizar diccionario
#DICCIONARIO A
from sklearn.preprocessing import MinMaxScaler

dict_A_norm = {}
dict_A_scalers = {}

for b, df_b in dict_A.items():
```

```
    scaler = MinMaxScaler()

    df_b_norm = df_b.copy()
    df_b_norm["stock_solicitado_s"] = scaler.fit_transform(df_b[["stock_solicitado"]])

    dict_A_norm[b] = df_b_norm
    dict_A_scalers[b] = scaler

#DICCIONARIO B
dict_B_norm = {}
dict_B_scalers = {}

for b, df_b in dict_B.items():
    scaler = MinMaxScaler()

    df_b_norm = df_b.copy()
    df_b_norm["stock_solicitado_s"] = scaler.fit_transform(df_b[["stock_solicitado"]])

    dict_B_norm[b] = df_b_norm
    dict_B_scalers[b] = scaler

dict_A_norm["BDG-19GNI"].head(15)

#@title 5- Crear Ventanas
from dateutil.relativedelta import relativedelta

def crear_ventanas_para_df(df_b, fecha_col="fecha", valor_col="stock_solicitado_s",
                           ventana=6, horizonte=1):
    """
    Crea ventanas (X, y) para un DataFrame de una sola bodega.
    - df_b: df con columnas fecha (datetime) y valor_col (normalizado).
    - Devuelve dict con X_seq (n_samples, ventana, 1), y (n_samples, horizonte),
      fecha_end (n_samples,) y n_original (n registros originales).
    - Si no hay suficientes observaciones, devuelve None.
    """
    # Asegurar orden por fecha
    df_b = df_b.sort_values(fecha_col).reset_index(drop=True)
    vals = df_b[valor_col].values
    fechas = pd.to_datetime(df_b[fecha_col].values)

    n = len(df_b)
    min_req = ventana + horizonte
    if n < min_req:
        return None

    X_list, y_list, fecha_end = [], [], []

    # sliding window
```

```
    for i in range(n - min_req + 1):
        start = i
        end = i + ventana  # exclusive index for the input window
        target_start = end
        target_end = end + horizonte

        X = vals[start:end].reshape(ventana, 1)        # (ventana, 1)
        y = vals[target_start:target_end].reshape(horizonte,)  # (horizonte,)

        X_list.append(X)
        y_list.append(y)
        fecha_end.append(fechas[end-1])  # fecha del último timestep de la ventana

    return {
        "X_seq": np.array(X_list, dtype=float),
        "y": np.array(y_list, dtype=float),
        "fecha_end": np.array(fecha_end, dtype=&#x27;datetime64[ns]'),
        "n_original": n
    }


def crear_ventanas_por_diccionario(dict_norm, ventana=6, horizonte=1,
                                   fecha_col="fecha", valor_col="stock_solicitado_s",
                                   min_samples=None):
    """
    Crea un nuevo diccionario con ventanas por bodega.
    - dict_norm: {bodega: df_b}
    - min_samples: umbral mínimo (si None, se asume ventana+horizonte)
    Devuelve:
      seq_dict: {bodega: {"X_seq", "y", "fecha_end", "n_original"}}
    """
    seq_dict = {}
    for bodega, df_b in dict_norm.items():
        out = crear_ventanas_para_df(df_b, fecha_col=fecha_col, valor_col=valor_col,
                                     ventana=ventana, horizonte=horizonte)
        if out is None:
            # opcional: loggear que la bodega se saltó
            # print(f"Saltada bodega {bodega}: menos de {ventana+horizonte} registros.")
            continue

        # chequeo de min_samples si aplica
        if min_samples is not None and out["X_seq"].shape[0] < min_samples:
            # print(f"Saltada bodega {bodega}: menos de {min_samples} muestras (ventanas).")
            continue

        seq_dict[bodega] = out

    return seq_dict
```

```
def concatenar_secuencias(seq_dict, ordenar_por_fecha_end=True):
    """
    Concatena todas las ventanas de seq_dict en arrays globales.
    Devuelve X_seq_all, X_bodega_idx, y_all, fecha_ends_all
      - X_seq_all: (N_total, ventana, 1)
      - X_bodega_idx: (N_total,) indices (or labels) of bodegas for each sample
      - y_all: (N_total, horizonte)
      - fecha_ends_all: (N_total,)
    Si ordenar_por_fecha_end True -> ordena globalmente por fecha_end asc.
    """
    b_list = []
    X_list, y_list, f_list = [], [], []

    for b, out in seq_dict.items():
        n = out["X_seq"].shape[0]
        X_list.append(out["X_seq"])
        y_list.append(out["y"])
        f_list.append(out["fecha_end"])
        b_list.extend([b]*n)

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

    X_all = np.concatenate(X_list, axis=0)
    y_all = np.concatenate(y_list, axis=0)
    f_all = np.concatenate(f_list, axis=0)
    b_all = np.array(b_list)

    if ordenar_por_fecha_end:
        order = np.argsort(f_all)
        X_all = X_all[order]
        y_all = y_all[order]
        f_all = f_all[order]
        b_all = b_all[order]

    return X_all, b_all, y_all, f_all

# Paraemetros
ventana = 6
horizonte = 1

#  crear ventanas por bodega para A y B
seq_A = crear_ventanas_por_diccionario(dict_A_norm, ventana=ventana, horizonte=horizonte)
seq_B = crear_ventanas_por_diccionario(dict_B_norm, ventana=ventana, horizonte=horizonte)

print("Bodegas con secuencias en A:", len(seq_A))
```

```
print("Bodegas con secuencias en B:", len(seq_B))


# @title 6- Crear split
"""
el split temporal por bodega sirve para separar los datos de Cada bodega en train / val / test
respetando su propia línea de tiempo, no la del dataset completo.
"""


#Split temporal por una bodega
from dateutil.relativedelta import relativedelta
def temporal_split_por_bodega(fecha_ends, test_months=2, val_months=2):
    """
    Dado un array de fechas (fecha_end) de UNA bodega,
    devuelve los boolean masks para train / val / test.
    """
    fecha_ends = pd.to_datetime(fecha_ends)
    max_fecha = fecha_ends.max()

    # Cortes temporales para esta bodega
    test_start = max_fecha - relativedelta(months=test_months)
    val_start = max_fecha - relativedelta(months=(test_months + val_months))

    # Máscaras
    mask_test  = fecha_ends > test_start
    mask_val   = (fecha_ends > val_start) & (fecha_ends <= test_start)
    mask_train = fecha_ends <= val_start

    return mask_train, mask_val, mask_test

# Split para todo el diccionario

def split_por_bodega(seq_dict, test_months=2, val_months=2):
    """
    Aplica el split temporal por bodega a todo el diccionario de secuencias.
    Retorna un nuevo diccionario con los splits por bodega.
    """
    split_dict = {}

    for bodega, datos in seq_dict.items():

        fechas = datos["fecha_end"]
        X = datos["X_seq"]
        y = datos["y"]

        # Obtener máscaras específicas para esta bodega
        mask_train, mask_val, mask_test = temporal_split_por_bodega(
```

```
            fechas,
            test_months=test_months,
            val_months=val_months
        )

        # Guardar las particiones
        split_dict[bodega] = {
            "X_train":  X[mask_train],
            "y_train":  y[mask_train],
            "fechas_train": fechas[mask_train],

            "X_val":    X[mask_val],
            "y_val":    y[mask_val],
            "fechas_val": fechas[mask_val],

            "X_test":   X[mask_test],
            "y_test":   y[mask_test],
            "fechas_test": fechas[mask_test],
        }

    return split_dict

seq_A = crear_ventanas_por_diccionario(dict_A_norm, ventana=6, horizonte=1)
seq_B = crear_ventanas_por_diccionario(dict_B_norm, ventana=6, horizonte=1)

split_A = split_por_bodega(seq_A, test_months=2, val_months=2)
split_B = split_por_bodega(seq_B, test_months=2, val_months=2)

# @title 7- Contrucción del modelo keras Tuner
def build_lstm_model(hp, input_shape):
    model = keras.Sequential()

    # Número de unidades LSTM
    units = hp.Int("units", min_value=16, max_value=128, step=16)

    model.add(layers.LSTM(units, return_sequences=False, input_shape=input_shape))

    # Densa final
    model.add(layers.Dense(1))

    # Learning rate
    lr = hp.Choice("lr", values=[1e-4, 5e-4, 1e-3])

    model.compile(
        optimizer=keras.optimizers.Adam(learning_rate=lr),
        loss="mse",
        metrics=["mae"]
    )
```

```
    return model

# @title 8- Entrenamiento automático
import os
from keras.callbacks import EarlyStopping, ModelCheckpoint

# Crear carpeta de modelos
os.makedirs("modelos_entrenados", exist_ok=True)

# Diccionarios a recorrer - CORRECTED to use split_A and split_B
dicts = {
    "A": split_A,
    "B": split_B
}

for nombre_modelo, dataset_dict in dicts.items():

    print(f"\n============================")
    print(f" ENTRENANDO MODELO {nombre_modelo}")
    print(f"============================\n")

    for bodega, splits in dataset_dict.items():

        print(f"\n--> Entrenando para bodega {bodega}")

        X_train = splits["X_train"]
        y_train = splits["y_train"]
        X_val   = splits["X_val"]
        y_val   = splits["y_val"]

        input_shape = (X_train.shape[1], X_train.shape[2])

        # ----------------------------
        # KERAS TUNER
        # ----------------------------
        tuner = kt.RandomSearch(
            lambda hp: build_lstm_model(hp, input_shape),
            objective="val_loss",
            max_trials=8,
            directory="tuner_results",
            project_name=f"tuner_{nombre_modelo}_{bodega}"
        )

        tuner.search(
            X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=50,
```

```
            callbacks=[
                EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
            ],
            verbose=0
        )

        # Mejor hiperparámetros
        best_hp = tuner.get_best_hyperparameters(1)[0]
        model = tuner.hypermodel.build(best_hp)

        # ----------------------------
        # ENTRENAMIENTO FINAL
        # ----------------------------
        ruta_guardado = f"modelos_entrenados/modelo_{nombre_modelo}_bodega_{bodega}.keras"

        checkpoint = ModelCheckpoint(
            ruta_guardado,
            monitor="val_loss",
            save_best_only=True,
            mode="min"
        )

        history = model.fit(
            X_train, y_train,
            validation_data=(X_val, y_val),
            epochs=100,
            callbacks=[
                EarlyStopping(monitor="val_loss", patience=12, restore_best_weights=True),
                checkpoint
            ],
            verbose=0
        )

        print(f"   ✓ Modelo guardado como: {ruta_guardado}")

#@title 9- Evaluar modelos

def evaluar_modelos(dict_modelos, nombre_diccionario, carpeta="modelos_entrenados"):
    """
    dict_modelos: seq_A o seq_B
    nombre_diccionario: "A" o "B"
    """

    resultados = []

    for bodega, splits in dict_modelos.items():

        ruta = os.path.join(carpeta, f"modelo_{nombre_diccionario}_bodega_{bodega}.keras")
```

```
        if not os.path.exists(ruta):
            print(f"No existe modelo para bodega {bodega}")
            continue

        # Cargar modelo
        model = tf.keras.models.load_model(ruta)

        # Evaluar
        X_test = splits["X_test"]
        y_test = splits["y_test"]

        loss, mae = model.evaluate(X_test, y_test, verbose=0)

        # Extraer hiperparámetros limpios directamente
        hp_clean = {}
        try:
            model_architecture_config = model.get_config()
            # Search for LSTM layer to get units
            for layer_config in model_architecture_config["layers"]:
                if layer_config["class_name"] == "LSTM":
                    hp_clean["lstm_units"] = layer_config["config"]["units"]
                    # Dropout is not tuned, defaults to 0 if not present
                    hp_clean["lstm_dropout"] = layer_config["config"].get("dropout", 0.0)
                    break
        except Exception as e:
            print(f"Error extracting LSTM units for {bodega}: {e}")

        try:
            # Get learning rate from the loaded model's optimizer
            if hasattr(model, &#x27;optimizer') and hasattr(model.optimizer, 'learning_rate'):
                # Keras 3 learning_rate can be a callable schedule
                if callable(model.optimizer.learning_rate):
                    # Pass optimizer.iterations to get current LR if it's a schedule
                    hp_clean["learning_rate"] =
float(model.optimizer.learning_rate(model.optimizer.iterations).numpy())
                else:
                    hp_clean["learning_rate"] = float(model.optimizer.learning_rate.numpy())
            else:
                hp_clean["learning_rate"] = None
        except Exception as e:
            print(f"Error extracting learning rate for {bodega}: {e}")
            hp_clean["learning_rate"] = None

        resultados.append({
            "diccionario": nombre_diccionario,
            "bodega": bodega,
            "loss": loss,
```

```
            "mae": mae,
            "hp": hp_clean # Store the cleaned HPs directly here
        })

    return resultados

resultados_A = evaluar_modelos(split_A, "A")
resultados_B = evaluar_modelos(split_B, "B")

df_res_A = pd.DataFrame(resultados_A).sort_values("mae")
df_res_B = pd.DataFrame(resultados_B).sort_values("mae")

print("TOP MODELOS A:")
print(df_res_A.head(28))

print("\nTOP MODELOS B:")
print(df_res_B.head(24))

#@ 10- Extraer hiperparámetros
def extraer_hp(model_config):
    """Devuelve solo lo importante del modelo."""
    if model_config is None:
        return {}

    hp = {}

    # Buscar capa LSTM
    for layer in model_config["layers"]:
        if "class_name" in layer and layer["class_name"] == "LSTM":
            hp["lstm_units"] = layer["config"]["units"]
            hp["lstm_dropout"] = layer["config"]["dropout"]

    # Buscar optimizador
    opt = model_config["compile_config"]["optimizer_config"]["config"]
    hp["learning_rate"] = opt.get("learning_rate", None)

    return hp

df_res_A["hp_clean"] = df_res_A["hp"]
df_res_B["hp_clean"] = df_res_B["hp"]

df_res_A[["bodega", "mae", "hp_clean"]].head()

# @title 10- Exportar a un csv
# The 'hp' column now contains the cleaned hyperparameters after re-running the evaluation cells.
df_res_A["hp_clean"] = df_res_A["hp"]
df_res_B["hp_clean"] = df_res_B["hp"]
```

```
# Seleccionar columnas importantes
cols = ["diccionario", "bodega", "mae", "loss", "hp_clean"]

# Exportar modelos del diccionario A
df_res_A[cols].to_csv("mejores_modelos_A.csv", index=False)

# Exportar modelos del diccionario B
df_res_B[cols].to_csv("mejores_modelos_B.csv", index=False)

print("Archivos exportados:\n- mejores_modelos_A.csv\n- mejores_modelos_B.csv")

#@title 11- Predicciones futuras
def predecir_futuro_por_bodega(
        dict_splits,        # seq_A o seq_B
        dict_series,        # dict_A_series o dict_B_series
        dict_scalers,       # scalers_A o scalers_B
        nombre_diccionario, # "A" o "B"
        carpeta_modelos="modelos_entrenados"
    ):

    resultados = []

    for bodega in dict_splits.keys():

        # -------------------
        # 1. Cargar modelo
        # -------------------
        ruta_modelo = os.path.join(carpeta_modelos, f"modelo_{nombre_diccionario}_bodega_{bodega}.keras")
        if not os.path.exists(ruta_modelo):
            print(f"Modelo NO encontrado para bodega {bodega} ({nombre_diccionario})")
            continue

        model = tf.keras.models.load_model(ruta_modelo)

        # -------------------
        # 2. Obtener última ventana de esa bodega
        # -------------------
        serie = dict_series[bodega]["stock_solicitado_s"].values # Corrected column name
        ventana_size = list(dict_splits[bodega]["X_test"].shape)[1]  # tamaño de ventana usado

        if len(serie) < ventana_size:
            print(f"Bodega {bodega}: serie muy corta para predicción futura.")
            continue

        ultima_ventana = serie[-ventana_size:].reshape(1, ventana_size, 1)

        # -------------------
        # 3. Predecir futuro
```

```
        # -------------------
        pred_scaled = model.predict(ultima_ventana, verbose=0)[0]

        # -------------------
        # 4. Invertir escala
        # -------------------
        scaler = dict_scalers[bodega]
        pred_real = scaler.inverse_transform(pred_scaled.reshape(-1, 1)).flatten()[0]

        # -------------------
        # 5. Guardar resultado
        # -------------------
        resultados.append({
            "diccionario": nombre_diccionario,
            "bodega": bodega,
            "pred_scaled": float(pred_scaled[0]),
            "pred_real": float(pred_real)
        })

    return pd.DataFrame(resultados)

pred_A = predecir_futuro_por_bodega(
    dict_splits=split_A,
    dict_series=dict_A_norm,
    dict_scalers=dict_A_scalers,
    nombre_diccionario="A"
)

pred_B = predecir_futuro_por_bodega(
    dict_splits=split_B,
    dict_series=dict_B_norm,
    dict_scalers=dict_B_scalers,
    nombre_diccionario="B"
)

#@title 12- graficar predicció vs historia
import matplotlib.pyplot as plt
import pandas as pd

def graficar_bodega(
        dict_series,     # dict_A_series o dict_B_series
        df_pred,         # pred_A o pred_B
        bodega,
        titulo_prefix=""
    ):

    if bodega not in dict_series:
        print(f"Bodega {bodega} no existe en el diccionario de series.")
```

```
        return

    # Serie histórica real
    serie = dict_series[bodega].copy()

    # Agregar predicción futura
    try:
        pred = df_pred[df_pred["bodega"] == bodega]["pred_real"].values[0]
    except:
        print(f"No hay predicción para bodega {bodega}")
        return

    # Crear fila futura
    fecha_futura = serie["fecha"].max() + pd.DateOffset(months=1)

    df_plot = serie.copy()
    df_plot["tipo"] = "historia"

    df_fut = pd.DataFrame({
        "fecha": [fecha_futura],
        "stock_solicitado": [pred],
        "serie_s": [None],
        "tipo": ["predicción"]
    })

    df_plot = pd.concat([df_plot, df_fut], ignore_index=True)

    # Graficar
    plt.figure(figsize=(12, 5))
    plt.plot(
        df_plot[df_plot["tipo"] == "historia"]["fecha"],
        df_plot[df_plot["tipo"] == "historia"]["stock_solicitado"],
        label="Historia",
        linewidth=2
    )
    plt.scatter(
        df_plot[df_plot["tipo"] == "predicción"]["fecha"],
        df_plot[df_plot["tipo"] == "predicción"]["stock_solicitado"],
        color="red",
        label="Predicción Próximo Mes",
        s=100
    )

    plt.title(f"{titulo_prefix} Bodega {bodega} — Predicción vs Historia")
    plt.xlabel("Fecha")
    plt.ylabel("Stock Solicitado")
    plt.legend()
    plt.grid(True)
```

```
    plt.tight_layout()
    # Save plot instead of showing
    filename = f"plot_{titulo_prefix.replace(&#x27; ', '_')}_{bodega}.png"
    plt.savefig(filename, dpi=100, bbox_inches=&#x27;tight')
    plt.close()
    print(f"Gráfico guardado: {filename}")


graficar_bodega(dict_series=dict_A_norm, df_pred=pred_A, bodega="BDG-19GNI", titulo_prefix="Producto A")

graficar_bodega(dict_series=dict_B_norm, df_pred=pred_B, bodega="BDG-19GNI", titulo_prefix="Producto B")

for b in pred_A["bodega"].unique():
    graficar_bodega(dict_series=dict_A_norm, df_pred=pred_A, bodega=b, titulo_prefix="Producto P9933")

for b in pred_B["bodega"].unique():
    graficar_bodega(dict_series=dict_B_norm, df_pred=pred_B, bodega=b, titulo_prefix="Producto P2417")

#@13- Guardar el mejor modelo
os.makedirs("modelos_A", exist_ok=True)
os.makedirs("modelos_B", exist_ok=True)

# Cargar modelos para guardarlos
best_models_A = {}
for bodega in df_res_A[&#x27;bodega']:
    path = f"modelos_entrenados/modelo_A_bodega_{bodega}.keras"
    if os.path.exists(path):
        best_models_A[bodega] = tf.keras.models.load_model(path)

best_models_B = {}
for bodega in df_res_B[&#x27;bodega']:
    path = f"modelos_entrenados/modelo_B_bodega_{bodega}.keras"
    if os.path.exists(path):
        best_models_B[bodega] = tf.keras.models.load_model(path)

# best_models_A[bodega]
# best_models_B[bodega]

def guardar_modelos(diccionario_modelos, ruta_base):
    """
    Guarda cada modelo del diccionario en una carpeta individual.
    """
    for bodega, modelo in diccionario_modelos.items(): # Changed 'info' to 'modelo' here

        # Crear ruta del modelo por bodega
        ruta_bodega = os.path.join(ruta_base, f"bodega_{bodega}")
        os.makedirs(ruta_bodega, exist_ok=True)
```

```
        # Nombre del archivo
        ruta_modelo = os.path.join(ruta_bodega, "best_model.keras")

        # Guardar modelo
        modelo.save(ruta_modelo)

        print(f"✓ Modelo guardado para bodega {bodega} en {ruta_modelo}")

guardar_modelos(best_models_A, "modelos_A")
guardar_modelos(best_models_B, "modelos_B")
```

2. Análisis Completo de Datos

Script para generar estadísticas globales y análisis masivo de bodegas.

`Archivo: analisis_completo_todos_los_datos.py`

```
# -*- coding: utf-8 -*-
"""
ANALISIS COMPLETO DEL CONJUNTO DE DATOS
========================================
Análisis exhaustivo de todos los productos y bodegas
"""

import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import json
import warnings
warnings.filterwarnings(&#x27;ignore')

print("="*80)
print("ANALISIS COMPLETO - SISTEMA DE PREDICCION DE DEMANDA")
print("="*80)

# ============================================
# CARGAR DATOS REALES
# ============================================
print("\n[1/6] Cargando datos del archivo Excel...")
url = "https://github.com/OscarT231/Proyecto-deep-/raw/refs/heads/main/Base_filtrada.xlsx"
df = pd.read_excel(url)

df.columns = df.columns.astype(str).str.strip()
columnas = [
    "bodega", "producto", "calificacion_abc",
    "2024-09-01 00:00:00","2024-10-01 00:00:00","2024-11-01 00:00:00","2024-12-01 00:00:00",
    "2025-01-01 00:00:00","2025-02-01 00:00:00","2025-03-01 00:00:00","2025-04-01 00:00:00",
    "2025-05-01 00:00:00","2025-06-01 00:00:00","2025-07-01 00:00:00","2025-08-01 00:00:00"
]

df = df[[col for col in columnas if col in df.columns]].copy()
df = df[~df["calificacion_abc"].isin(["O", "N"])].copy()
```

```
# Formato long
id_cols = ["bodega", "producto", "calificacion_abc"]
date_cols = [c for c in df.columns if c not in id_cols]
df_long = df.melt(id_vars=id_cols, value_vars=date_cols, var_name="fecha", value_name="stock")
df_long[&#x27;fecha'] = pd.to_datetime(df_long['fecha'])
df_long = df_long.sort_values(["bodega", "producto", "fecha"])

print(f"   Total de registros cargados: {len(df_long):,}")
print(f"   Productos unicos: {df_long[&#x27;producto'].nunique()}")
print(f"   Bodegas uniques: {df_long[&#x27;bodega'].nunique()}")

# ============================================
# ANALISIS PRODUCTO A (P9933)
# ============================================
print("\n[2/6] Analizando Producto P9933 (Categoria A)...")
df_P9933 = df_long[df_long[&#x27;producto'] == 'P9933']

resultados_A = []
for bodega in df_P9933[&#x27;bodega'].unique():
    datos = df_P9933[df_P9933[&#x27;bodega'] == bodega]
    if len(datos) >= 6:
        try:
            ultimos_6 = datos[&#x27;stock'].tail(6).values
            scaler = MinMaxScaler()
            scaler.fit(ultimos_6.reshape(-1, 1))
            norm = scaler.transform(ultimos_6.reshape(-1, 1))

            modelo = tf.keras.models.load_model(f&#x27;modelos_A/bodega_{bodega}/best_model.keras')
            pred = modelo.predict(norm.reshape(1, 6, 1), verbose=0)
            pred_real = scaler.inverse_transform(pred.reshape(-1, 1))[0][0]

            resultados_A.append({
                &#x27;bodega': bodega,
                &#x27;demanda_promedio': ultimos_6.mean(),
                &#x27;demanda_min': ultimos_6.min(),
                &#x27;demanda_max': ultimos_6.max(),
                &#x27;feb_2025': ultimos_6[-1],
                &#x27;mar_2025_pred': pred_real,
                &#x27;cambio_abs': pred_real - ultimos_6[-1],
                &#x27;cambio_pct': ((pred_real - ultimos_6[-1]) / ultimos_6[-1]) * 100
            })
        except Exception as e:
            pass

df_A = pd.DataFrame(resultados_A)
print(f"   Bodegas analizadas: {len(df_A)}")
print(f"   Demanda total predicha: {df_A[&#x27;mar_2025_pred'].sum():.0f} unidades")
```

```
# ============================================
# ANALISIS PRODUCTO B (P2417)
# ============================================
print("\n[3/6] Analizando Producto P2417 (Categoria B)...")
df_P2417 = df_long[df_long[&#x27;producto'] == 'P2417']

resultados_B = []
for bodega in df_P2417[&#x27;bodega'].unique():
    datos = df_P2417[df_P2417[&#x27;bodega'] == bodega]
    if len(datos) >= 6:
        try:
            ultimos_6 = datos[&#x27;stock'].tail(6).values
            scaler = MinMaxScaler()
            scaler.fit(ultimos_6.reshape(-1, 1))
            norm = scaler.transform(ultimos_6.reshape(-1, 1))

            modelo = tf.keras.models.load_model(f&#x27;modelos_B/bodega_{bodega}/best_model.keras')
            pred = modelo.predict(norm.reshape(1, 6, 1), verbose=0)
            pred_real = scaler.inverse_transform(pred.reshape(-1, 1))[0][0]

            resultados_B.append({
                &#x27;bodega': bodega,
                &#x27;demanda_promedio': ultimos_6.mean(),
                &#x27;demanda_min': ultimos_6.min(),
                &#x27;demanda_max': ultimos_6.max(),
                &#x27;feb_2025': ultimos_6[-1],
                &#x27;mar_2025_pred': pred_real,
                &#x27;cambio_abs': pred_real - ultimos_6[-1],
                &#x27;cambio_pct': ((pred_real - ultimos_6[-1]) / ultimos_6[-1]) * 100
            })
        except Exception as e:
            pass

df_B = pd.DataFrame(resultados_B)
print(f"   Bodegas analizadas: {len(df_B)}")
print(f"   Demanda total predicha: {df_B[&#x27;mar_2025_pred'].sum():.0f} unidades")

# ============================================
# METRICAS DE RENDIMIENTO
# ============================================
print("\n[4/6] Analizando metricas de rendimiento...")
metricas_A = pd.read_csv(&#x27;mejores_modelos_A.csv')
metricas_B = pd.read_csv(&#x27;mejores_modelos_B.csv')

mae_promedio_A = metricas_A[&#x27;mae'].mean()
mae_promedio_B = metricas_B[&#x27;mae'].mean()

print(f"   MAE promedio Producto A: {mae_promedio_A:.6f}")
```

```
print(f"   MAE promedio Producto B: {mae_promedio_B:.6f}")

# ============================================
# ESTADISTICAS GLOBALES
# ============================================
print("\n[5/6] Calculando estadisticas globales...")

estadisticas = {
    &#x27;producto_A': {
        &#x27;bodegas': int(len(df_A)),
        &#x27;demanda_total_feb': float(df_A['feb_2025'].sum()),
        &#x27;demanda_total_mar_pred': float(df_A['mar_2025_pred'].sum()),
        &#x27;demanda_promedio': float(df_A['demanda_promedio'].mean()),
        &#x27;crecimiento_total': float(df_A['mar_2025_pred'].sum() - df_A['feb_2025'].sum()),
        &#x27;mae_promedio': float(mae_promedio_A)
    },
    &#x27;producto_B': {
        &#x27;bodegas': int(len(df_B)),
        &#x27;demanda_total_feb': float(df_B['feb_2025'].sum()),
        &#x27;demanda_total_mar_pred': float(df_B['mar_2025_pred'].sum()),
        &#x27;demanda_promedio': float(df_B['demanda_promedio'].mean()),
        &#x27;crecimiento_total': float(df_B['mar_2025_pred'].sum() - df_B['feb_2025'].sum()),
        &#x27;mae_promedio': float(mae_promedio_B)
    },
    &#x27;global': {
        &#x27;total_bodegas': int(len(df_A) + len(df_B)),
        &#x27;total_modelos': 52,
        &#x27;registros_analizados': int(len(df_long)),
        &#x27;demanda_total_predicha': float(df_A['mar_2025_pred'].sum() + df_B['mar_2025_pred'].sum())
    }
}

# ============================================
# EXPORTAR RESULTADOS
# ============================================
print("\n[6/6] Exportando resultados...")

# Guardar DataFrames
df_A.to_csv(&#x27;analisis_completo_producto_A.csv', index=False)
df_B.to_csv(&#x27;analisis_completo_producto_B.csv', index=False)

# Guardar estadísticas
with open(&#x27;estadisticas_globales.json', 'w', encoding='utf-8') as f:
    json.dump(estadisticas, f, indent=2, ensure_ascii=False)

print("\n" + "="*80)
print("RESUMEN EJECUTIVO")
print("="*80)
```

```
print(f"\nProducto P9933 (Categoria A):")
print(f"  - Bodegas analizadas: {estadisticas[&#x27;producto_A'][&#x27;bodegas']}")
print(f"  - Demanda Feb 2025: {estadisticas[&#x27;producto_A'][&#x27;demanda_total_feb']:.0f} unidades")
print(f"  - Prediccion Mar 2025: {estadisticas[&#x27;producto_A'][&#x27;demanda_total_mar_pred']:.0f} unidades")
print(f"  - Crecimiento: {estadisticas[&#x27;producto_A'][&#x27;crecimiento_total']:+.0f} unidades")

print(f"\nProducto P2417 (Categoria B):")
print(f"  - Bodegas analizadas: {estadisticas[&#x27;producto_B'][&#x27;bodegas']}")
print(f"  - Demanda Feb 2025: {estadisticas[&#x27;producto_B'][&#x27;demanda_total_feb']:.0f} unidades")
print(f"  - Prediccion Mar 2025: {estadisticas[&#x27;producto_B'][&#x27;demanda_total_mar_pred']:.0f} unidades")
print(f"  - Crecimiento: {estadisticas[&#x27;producto_B'][&#x27;crecimiento_total']:+.0f} unidades")

print(f"\nTOTAL GENERAL:")
print(f"  - Demanda predicha (ambos productos): {estadisticas[&#x27;global'][&#x27;demanda_total_predicha']:.0f}
unidades")
print(f"  - Modelos entrenados: {estadisticas[&#x27;global'][&#x27;total_modelos']}")
print(f"  - Precision promedio (MAE): {(mae_promedio_A + mae_promedio_B)/2:.6f}")

print("\n" + "="*80)
print("Archivos generados:")
print("  - analisis_completo_producto_A.csv")
print("  - analisis_completo_producto_B.csv")
print("  - estadisticas_globales.json")
print("="*80)
```

3. Generador de Predicciones Reales

Script para cargar modelos y generar predicciones con datos nuevos.

`Archivo: predicciones_REALES.py`

```
# -*- coding: utf-8 -*-
"""
USO DE MODELOS CON DATOS REALES
================================
Este script muestra como cargar TUS datos reales y hacer predicciones
"""

import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# ============================================
# PASO 1: Cargar tus datos REALES
# ============================================

# Cargar el dataset original
url = "https://github.com/OscarT231/Proyecto-deep-/raw/refs/heads/main/Base_filtrada.xlsx"
df = pd.read_excel(url)

# Filtrar columnas
df.columns = df.columns.astype(str).str.strip()
columnas_deseadas = [
    "bodega", "producto", "calificacion_abc",
    "2024-09-01 00:00:00","2024-10-01 00:00:00","2024-11-01 00:00:00","2024-12-01 00:00:00",
    "2025-01-01 00:00:00","2025-02-01 00:00:00","2025-03-01 00:00:00","2025-04-01 00:00:00",
    "2025-05-01 00:00:00","2025-06-01 00:00:00","2025-07-01 00:00:00","2025-08-01 00:00:00"
]

df_sugerido = df[[col for col in columnas_deseadas if col in df.columns]].copy()
df_sugerido = df_sugerido[~df_sugerido["calificacion_abc"].isin(["O", "N"])].copy()

# Convertir a formato long
id_cols = ["bodega", "producto", "calificacion_abc"]
date_cols = [c for c in df_sugerido.columns if c not in id_cols]

df_long = df_sugerido.melt(id_vars=id_cols,
```

```
                            value_vars=date_cols,
                            var_name="fecha",
                            value_name="stock_solicitado")

df_long[&#x27;fecha'] = pd.to_datetime(df_long['fecha'])
df_long = df_long.sort_values(["bodega", "producto", "fecha"])

print("Datos cargados exitosamente!")
print(f"Total de registros: {len(df_long)}")

# ============================================
# PASO 2: Hacer prediccion REAL para una bodega
# ============================================

# Ejemplo: Producto P9933, Bodega BDG-19GNI
bodega_seleccionada = "BDG-19GNI"
producto_seleccionado = "P9933"

# Filtrar datos de esa bodega y producto
datos_bodega = df_long[
    (df_long[&#x27;bodega'] == bodega_seleccionada) &
    (df_long[&#x27;producto'] == producto_seleccionado)
].copy()

print(f"\n{&#x27;='*60}")
print(f"Prediccion REAL para: {producto_seleccionado} - {bodega_seleccionada}")
print(f"{&#x27;='*60}")

if len(datos_bodega) >= 6:
    # Obtener ultimos 6 meses REALES
    ultimos_6 = datos_bodega[&#x27;stock_solicitado'].tail(6).values

    print("\nDemanda historica REAL (ultimos 6 meses):")
    for i, val in enumerate(ultimos_6, 1):
        print(f"  Mes {i}: {val:.0f} unidades")

    # Normalizar
    scaler = MinMaxScaler()
    scaler.fit(ultimos_6.reshape(-1, 1))
    datos_norm = scaler.transform(ultimos_6.reshape(-1, 1))

    # Cargar modelo
    modelo = tf.keras.models.load_model(f&#x27;modelos_A/bodega_{bodega_seleccionada}/best_model.keras')

    # Predecir
    entrada = datos_norm.reshape(1, 6, 1)
    pred_norm = modelo.predict(entrada, verbose=0)
    pred_real = scaler.inverse_transform(pred_norm.reshape(-1, 1))[0][0]
```

```
    print(f"\n>>> PREDICCION REAL PROXIMO MES: {pred_real:.0f} unidades <<<")
    print(f"{&#x27;='*60}")
else:
    print("No hay suficientes datos historicos para esta bodega")

# ============================================
# PASO 3: Predicciones para TODAS las bodegas de un producto
# ============================================

print("\n" + "="*60)
print("PREDICCIONES PARA TODAS LAS BODEGAS - Producto P9933")
print("="*60)

# Filtrar solo producto P9933
df_P9933 = df_long[df_long[&#x27;producto'] == 'P9933']

resultados_reales = []

for bodega in df_P9933[&#x27;bodega'].unique():
    datos_bodega = df_P9933[df_P9933[&#x27;bodega'] == bodega]

    if len(datos_bodega) >= 6:
        ultimos_6 = datos_bodega[&#x27;stock_solicitado'].tail(6).values

        # Normalizar
        scaler = MinMaxScaler()
        scaler.fit(ultimos_6.reshape(-1, 1))
        datos_norm = scaler.transform(ultimos_6.reshape(-1, 1))

        # Cargar modelo
        try:
            modelo = tf.keras.models.load_model(f&#x27;modelos_A/bodega_{bodega}/best_model.keras')

            # Predecir
            entrada = datos_norm.reshape(1, 6, 1)
            pred_norm = modelo.predict(entrada, verbose=0)
            pred_real = scaler.inverse_transform(pred_norm.reshape(-1, 1))[0][0]

            resultados_reales.append({
                &#x27;bodega': bodega,
                &#x27;ult_mes_real': ultimos_6[-1],
                &#x27;prediccion': pred_real,
                &#x27;cambio': pred_real - ultimos_6[-1]
            })
        except:
            pass
```

```
# Mostrar resultados
df_resultados = pd.DataFrame(resultados_reales)
df_resultados = df_resultados.sort_values(&#x27;prediccion', ascending=False)

print("\nTop 10 bodegas con MAYOR demanda predicha:")
print(df_resultados.head(10).to_string(index=False))

print("\n" + "="*60)
print("ESTOS SON TUS DATOS REALES, NO UN DEMO")
print("="*60)
```

4. Ejemplo de Uso

Demo simplificada de cómo cargar un modelo y hacer una inferencia.

`Archivo: ejemplo_uso_modelo.py`

```
"""
EJEMPLO DE USO DE LOS MODELOS ENTRENADOS
========================================
Este script muestra cómo cargar y usar los modelos LSTM entrenados
para hacer predicciones de demanda de inventario.
"""

import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# ================================================
# EJEMPLO 1: PREDICCIÓN PARA PRODUCTO P9933 (A)
# ================================================

print("=" * 60)
print("EJEMPLO 1: Predicción para Producto P9933 - Bodega BDG-19GNI")
print("=" * 60)

# 1. Cargar el modelo entrenado
modelo_A = tf.keras.models.load_model(&#x27;modelos_A/bodega_BDG-19GNI/best_model.keras')
print("\n[OK] Modelo cargado exitosamente")
print(f"  Arquitectura: {modelo_A.summary()}\n")

# 2. Datos de ejemplo: últimos 6 meses de demanda real
# (En producción, estos vendrían de tu base de datos)
demanda_historica = np.array([150, 180, 200, 175, 190, 220])  # Ejemplo en unidades
print("Demanda histórica (últimos 6 meses):")
print(f"  {demanda_historica}")

# 3. Normalizar los datos (el modelo fue entrenado con datos normalizados)
scaler = MinMaxScaler()
# Necesitamos entrenar el scaler con datos históricos completos de esa bodega
# Por simplicidad, aquí uso min-max de estos 6 meses
scaler.fit(demanda_historica.reshape(-1, 1))
demanda_normalizada = scaler.transform(demanda_historica.reshape(-1, 1))
```

```
# 4. Preparar formato de entrada: (1, 6, 1)
#    1 muestra, 6 timesteps (meses), 1 feature (demanda)
entrada = demanda_normalizada.reshape(1, 6, 1)
print(f"\nDatos normalizados preparados con forma: {entrada.shape}")

# 5. Hacer la predicción
prediccion_norm = modelo_A.predict(entrada, verbose=0)
print(f"Predicción normalizada: {prediccion_norm[0][0]:.4f}")

# 6. Desnormalizar para obtener la demanda real predicha
prediccion_real = scaler.inverse_transform(prediccion_norm.reshape(-1, 1))[0][0]
print(f"\n>>> PREDICCION PROXIMO MES: {prediccion_real:.0f} unidades")

# ================================================
# EJEMPLO 2: PREDICCIÓN PARA PRODUCTO P2417 (B)
# ================================================

print("\n" + "=" * 60)
print("EJEMPLO 2: Predicción para Producto P2417 - Bodega BDG-19GNI")
print("=" * 60)

# 1. Cargar modelo del producto B
modelo_B = tf.keras.models.load_model(&#x27;modelos_B/bodega_BDG-19GNI/best_model.keras')
print("\n[OK] Modelo cargado exitosamente")

# 2. Datos históricos diferentes
demanda_historica_B = np.array([80, 95, 110, 88, 100, 115])
print("Demanda histórica (últimos 6 meses):")
print(f"  {demanda_historica_B}")

# 3. Normalizar
scaler_B = MinMaxScaler()
scaler_B.fit(demanda_historica_B.reshape(-1, 1))
demanda_normalizada_B = scaler_B.transform(demanda_historica_B.reshape(-1, 1))

# 4. Preparar entrada
entrada_B = demanda_normalizada_B.reshape(1, 6, 1)

# 5. Predecir
prediccion_norm_B = modelo_B.predict(entrada_B, verbose=0)

# 6. Desnormalizar
prediccion_real_B = scaler_B.inverse_transform(prediccion_norm_B.reshape(-1, 1))[0][0]
print(f"\n>>> PREDICCION PROXIMO MES: {prediccion_real_B:.0f} unidades")

# ================================================
# EJEMPLO 3: PREDICCIÓN PARA TODAS LAS BODEGAS
```

```
# ================================================

print("\n" + "=" * 60)
print("EJEMPLO 3: Predicciones para todas las bodegas del Producto A")
print("=" * 60)

import os

# Listar todas las bodegas disponibles
bodegas_A = os.listdir(&#x27;modelos_A')
print(f"\nBodegas disponibles: {len(bodegas_A)}")

# Hacer predicción para las primeras 5 bodegas (ejemplo)
resultados = []

for bodega in bodegas_A[:5]:
    modelo_path = f&#x27;modelos_A/{bodega}/best_model.keras'
    modelo = tf.keras.models.load_model(modelo_path)

    # Usar datos de ejemplo (en producción, usar datos reales por bodega)
    datos_ejemplo = np.array([100, 120, 110, 130, 125, 140])
    scaler_temp = MinMaxScaler()
    scaler_temp.fit(datos_ejemplo.reshape(-1, 1))
    datos_norm = scaler_temp.transform(datos_ejemplo.reshape(-1, 1))

    pred_norm = modelo.predict(datos_norm.reshape(1, 6, 1), verbose=0)
    pred_real = scaler_temp.inverse_transform(pred_norm.reshape(-1, 1))[0][0]

    resultados.append({
        &#x27;bodega': bodega.replace('bodega_', ''),
        &#x27;prediccion': pred_real
    })

# Mostrar resultados
df_resultados = pd.DataFrame(resultados)
print("\nPredicciones por bodega:")
print(df_resultados.to_string(index=False))

# ================================================
# MÉTRICAS DE RENDIMIENTO
# ================================================

print("\n" + "=" * 60)
print("MÉTRICAS DE RENDIMIENTO DE LOS MODELOS")
print("=" * 60)

# Leer métricas guardadas
metricas_A = pd.read_csv(&#x27;mejores_modelos_A.csv')
```

```
metricas_B = pd.read_csv(&#x27;mejores_modelos_B.csv')

print("\nTop 5 modelos con MENOR error (MAE) - Producto A:")
print(metricas_A.nsmallest(5, &#x27;mae')[['bodega', 'mae', 'loss']])

print("\nTop 5 modelos con MENOR error (MAE) - Producto B:")
print(metricas_B.nsmallest(5, &#x27;mae')[['bodega', 'mae', 'loss']])

print("\n" + "=" * 60)
print("[OK] Ejemplos completados")
print("=" * 60)
```

5. Predicción Simple

Versión minimalista para pruebas rápidas.

`Archivo: prediccion_simple_real.py`

```
# -*- coding: utf-8 -*-
import tensorflow as tf
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import warnings
warnings.filterwarnings(&#x27;ignore')

print("Cargando datos reales...")

# Cargar datos del Excel
url = "https://github.com/OscarT231/Proyecto-deep-/raw/refs/heads/main/Base_filtrada.xlsx"
df = pd.read_excel(url)

# Preparar datos
df.columns = df.columns.astype(str).str.strip()
columnas = [
    "bodega", "producto", "calificacion_abc",
    "2024-09-01 00:00:00","2024-10-01 00:00:00","2024-11-01 00:00:00","2024-12-01 00:00:00",
    "2025-01-01 00:00:00","2025-02-01 00:00:00","2025-03-01 00:00:00","2025-04-01 00:00:00",
    "2025-05-01 00:00:00","2025-06-01 00:00:00","2025-07-01 00:00:00","2025-08-01 00:00:00"
]

df = df[[col for col in columnas if col in df.columns]].copy()
df = df[~df["calificacion_abc"].isin(["O", "N"])].copy()

# Formato long
id_cols = ["bodega", "producto", "calificacion_abc"]
date_cols = [c for c in df.columns if c not in id_cols]
df_long = df.melt(id_vars=id_cols, value_vars=date_cols, var_name="fecha", value_name="stock")
df_long[&#x27;fecha'] = pd.to_datetime(df_long['fecha'])
df_long = df_long.sort_values(["bodega", "producto", "fecha"])

print("OK - Datos cargados")
print(f"Total registros: {len(df_long)}")

# Prediccion para bodega especifica
```

```
bodega = "BDG-19GNI"
producto = "P9933"

print("\n" + "="*60)
print(f"PREDICCION REAL: {producto} - {bodega}")
print("="*60)

datos = df_long[(df_long[&#x27;bodega'] == bodega) & (df_long['producto'] == producto)].copy()

if len(datos) >= 6:
    ultimos_6 = datos[&#x27;stock'].tail(6).values

    print("\nDemanda historica REAL:")
    for i, val in enumerate(ultimos_6, 1):
        print(f"  Mes {i}: {int(val)} unidades")

    # Normalizar y predecir
    scaler = MinMaxScaler()
    scaler.fit(ultimos_6.reshape(-1, 1))
    norm = scaler.transform(ultimos_6.reshape(-1, 1))

    modelo = tf.keras.models.load_model(f&#x27;modelos_A/bodega_{bodega}/best_model.keras')
    pred = modelo.predict(norm.reshape(1, 6, 1), verbose=0)
    pred_real = scaler.inverse_transform(pred.reshape(-1, 1))[0][0]

    print(f"\n>>> PREDICCION REAL: {int(pred_real)} unidades <<<")
    print(f"Cambio vs ultimo mes: {int(pred_real - ultimos_6[-1]):+d} unidades")

# Top bodegas
print("\n" + "="*60)
print("TOP 5 BODEGAS CON MAYOR DEMANDA PREDICHA - Producto P9933")
print("="*60)

df_P9933 = df_long[df_long[&#x27;producto'] == 'P9933']
resultados = []

for bod in df_P9933[&#x27;bodega'].unique()[:10]:
    d = df_P9933[df_P9933[&#x27;bodega'] == bod]
    if len(d) >= 6:
        try:
            u6 = d[&#x27;stock'].tail(6).values
            sc = MinMaxScaler()
            sc.fit(u6.reshape(-1, 1))
            n = sc.transform(u6.reshape(-1, 1))
            m = tf.keras.models.load_model(f&#x27;modelos_A/bodega_{bod}/best_model.keras')
            p = m.predict(n.reshape(1, 6, 1), verbose=0)
            pr = sc.inverse_transform(p.reshape(-1, 1))[0][0]
            resultados.append({&#x27;bodega': bod, 'actual': int(u6[-1]), 'prediccion': int(pr)})
```

```
        except:
            pass

df_res = pd.DataFrame(resultados).sort_values(&#x27;prediccion', ascending=False)
print("\n", df_res.head().to_string(index=False))

print("\n" + "="*60)
print("DATOS 100% REALES DE TU EXCEL")
print("="*60)
```

Documento generado automáticamente el 23/11/2025 06:59