```
!pip install numpy pandas matplotlib seaborn scikit-learn xgboost shap li
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (2.0.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.12/dist-packages (2.2.2)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: seaborn in /usr/local/lib/python3.12/dist-packages (0.13.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: xgboost in /usr/local/lib/python3.12/dist-packages (3.1.1)
Requirement already satisfied: shap in /usr/local/lib/python3.12/dist-packages (0.49.1)
Requirement already satisfied: lime in /usr/local/lib/python3.12/dist-packages (0.2.0.1)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-packages (2.19.0)
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (1.5.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (25.0)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.5)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.3)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.12/dist-packages (from xgboost) (2.27.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.12/dist-packages (from shap) (4.67.1)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.12/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.12/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.12/dist-packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.12/dist-packages (from shap) (4.15.0)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.12/dist-packages (from lime) (0.25.2)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.9.23)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.32.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.2.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.0.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.76.0)
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.10.0)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.15.1)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.5.3)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from astunparse>=1.6.0->tensorflo
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.17.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.12/dist-packages (from numba>=0.54->shap)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->t
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorf
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorf
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (3.5)
```

```
import os
import random
import numpy as np
import pandas as pd
import joblib
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
from xgboost import XGBRegressor

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```
from tensorflow.keras.callbacks import EarlyStopping

import shap
from lime.lime_tabular import LimeTabularExplainer
import re
# Reproducibility
SEED = 42
np.random.seed(SEED)
random.seed(SEED)
tf.random.set_seed(SEED)
os.makedirs("outputs", exist_ok=True)

#Synthetic Data Generation
def generate_synthetic_retail_data(n_days=1200, n_products=1, seed=SEED):
    np.random.seed(seed)
    rows = []
    base_date = pd.to_datetime("2020-01-01")
    for d in range(n_days):
        date = base_date + pd.Timedelta(days=d)
        seasonal = 1 + 0.2 * np.sin(2 * np.pi * d / 365)
        weather_event = 1 if np.random.rand() < 0.05 else 0
        for pid in range(n_products):
            price = float(np.round(np.random.uniform(10, 100), 2))
            discount = float(np.round(np.random.choice([0, 5, 10, 15, 20], p=[0.7,0.1,0.1,0.07,0.03]), 2))
            rival_price = float(np.round(price * np.random.uniform(0.8, 1.2), 2))
            marketing_spend = float(np.round(price * np.random.uniform(0, 0.2), 2))
            inventory_level = int(max(0, np.random.normal(200, 50)))
            elasticity = float(np.random.normal(-1.5, 0.5))
            noise = np.random.normal(0, 5)
            demand_forecast = np.exp(elasticity * np.log(max(price,1) / max(rival_price,1))) * seasonal * (1 + 0.1 * weather_ev
            demand_forecast = float(demand_forecast * (1 + np.random.normal(0, 0.1)))
            units_base = max(0, demand_forecast * 50)
            units_sold = int(max(0, units_base + marketing_spend * 0.5 + noise))
            units_sold = min(units_sold, inventory_level)
            revenue = float(price * units_sold)
            rows.append({
                "date": date,
                "product_id": pid,
                "price": price,
                "discount": discount,
                "rival_price": rival_price,
                "demand_forecast": demand_forecast,
                "inventory_level": inventory_level,
                "marketing_spend": marketing_spend,
                "elasticity": elasticity,
                "seasonality": seasonal,
                "weather_event": weather_event,
                "units_sold": units_sold,
                "revenue": revenue
            })
    return pd.DataFrame(rows)
# Preprocessing
def preprocess_tabular(df, target_col="revenue", drop_cols=None, scaler=None, scaler_type="minmax"):
    if drop_cols is None:
        drop_cols = ["date", "product_id", "units_sold"]
    df = df.copy()
    X = df.drop(columns=[c for c in drop_cols if c in df.columns] + [target_col], errors='ignore')
    y = df[target_col].values
    X = X.fillna(X.median())
    if scaler is None:
        scaler = MinMaxScaler() if scaler_type == "minmax" else StandardScaler()
        X_scaled = scaler.fit_transform(X)
    else:
        X_scaled = scaler.transform(X)
    return pd.DataFrame(X_scaled, columns=X.columns, index=X.index), y, scaler

def create_lstm_sequences(df, feature_cols, target_col="revenue", window=30):
    df_sorted = df.sort_values(["product_id", "date"]).reset_index(drop=True)
    X_list, y_list = [], []
    for pid, grp in df_sorted.groupby("product_id"):
        features = grp[feature_cols].values
        target = grp[target_col].values
        for i in range(window, len(grp)):
            X_list.append(features[i-window:i, :])
            y_list.append(target[i])
    return np.array(X_list), np.array(y_list)
# Models
```

```
def build_lstm_model(input_shape, units=64, dropout=0.2):
    model = Sequential([
        LSTM(units, return_sequences=True, input_shape=input_shape),
        Dropout(dropout),
        LSTM(units, return_sequences=False),
        Dropout(dropout),
        Dense(32, activation="relu"),
        Dense(1, activation="linear")
    ])
    model.compile(optimizer="adam", loss="mse", metrics=["mae"])
    return model

def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

def compute_metrics(y_true, y_pred):
    return {"MAE": float(mean_absolute_error(y_true, y_pred)),
            "RMSE": float(rmse(y_true, y_pred))}

def revenue_uplift_pct(rmse_base, rmse_model):
    if rmse_base == 0: return np.nan
    return (rmse_base - rmse_model) / rmse_base * 100.0
# Visualization Enhancements
def plot_training_history(history, save_path="outputs/lstm_training_curves.png"):
    plt.figure(figsize=(8,4))
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Val Loss')
    plt.plot(history.history['mae'], label='Train MAE')
    plt.plot(history.history['val_mae'], label='Val MAE')
    plt.title("LSTM Training Curves")
    plt.xlabel("Epochs")
    plt.ylabel("Value")
    plt.legend()
    plt.tight_layout()
    plt.savefig(save_path, dpi=150)
    plt.show()

def plot_feature_correlation(df):
    plt.figure(figsize=(8,6))
    corr = df.corr()
    sns.heatmap(corr, annot=False, cmap='coolwarm')
    plt.title("Feature Correlation Heatmap")
    plt.tight_layout()
    plt.savefig("outputs/correlation_heatmap.png", dpi=150)
    plt.show()

def plot_feature_importance_rf(rf_model, feature_names):
    importances = rf_model.feature_importances_
    idx = np.argsort(importances)[::-1]
    plt.figure(figsize=(8,5))
    plt.bar(range(len(importances)), importances[idx])
    plt.xticks(range(len(importances)), np.array(feature_names)[idx], rotation=45)
    plt.title("Random Forest Feature Importance")
    plt.tight_layout()
    plt.savefig("outputs/feature_importance_rf.png", dpi=150)
    plt.show()

def plot_model_comparison(results):
    metrics = ["MAE", "RMSE"]
    models = [m for m in results.keys() if isinstance(results[m], dict)]

    for metric in metrics:
        plt.figure(figsize=(7,5))
        vals = [results[m][metric] for m in models]
        plt.bar(models, vals, color=['skyblue','orange','green','purple'])
        plt.title(f"Model Comparison - {metric}")
        plt.ylabel(metric)
        plt.tight_layout()
        plt.savefig(f"outputs/model_comparison_{metric}.png", dpi=150)
        plt.show()
# Main Pipeline
def run_full_pipeline(n_days=1200, n_products=1, run_lstm=True):
    df = generate_synthetic_retail_data(n_days, n_products)
    split_idx = int(len(df)*0.8)
    df_train, df_test = df.iloc[:split_idx], df.iloc[split_idx:]
    plot_feature_correlation(df_train)
```

```
    X_train, y_train, scaler = preprocess_tabular(df_train)
    X_test, y_test, _ = preprocess_tabular(df_test, scaler=scaler)
    print("Train/Test:", X_train.shape, X_test.shape)

    # --- Models ---
    lr = LinearRegression().fit(X_train, y_train)
    rf = RandomForestRegressor(n_estimators=200, max_depth=12, random_state=SEED, n_jobs=-1).fit(X_train, y_train)
    xgb = XGBRegressor(n_estimators=250, learning_rate=0.1, max_depth=6, subsample=0.8, random_state=SEED).fit(X_train, y_trair

    preds = {"LR": lr.predict(X_test), "RF": rf.predict(X_test), "XGB": xgb.predict(X_test)}
    results = {k: compute_metrics(y_test, v) for k,v in preds.items()}
    results["RUP_RF"] = revenue_uplift_pct(results["LR"]["RMSE"], results["RF"]["RMSE"])
    results["RUP_XGB"] = revenue_uplift_pct(results["LR"]["RMSE"], results["XGB"]["RMSE"])

    plot_feature_importance_rf(rf, X_train.columns)

    # --- SHAP Explainability ---
    try:
        booster = xgb.get_booster()
        explainer = shap.TreeExplainer(booster)
        shap_values = explainer.shap_values(X_test)
        shap.summary_plot(shap_values, X_test, show=False)
        plt.tight_layout()
        plt.savefig("outputs/shap_summary_beeswarm.png", dpi=150)
        plt.close()

        shap.summary_plot(shap_values, X_test, plot_type="bar", show=False)
        plt.tight_layout()
        plt.savefig("outputs/shap_feature_importance_bar.png", dpi=150)
        plt.close()
        print("SHAP plots saved.")
    except Exception as e:
        print("SHAP skipped due to:", e)

    # --- LIME ---
    lime_exp = LimeTabularExplainer(np.array(X_train), feature_names=X_train.columns, mode='regression')
    exp = lime_exp.explain_instance(X_test.iloc[10].values, rf.predict, num_features=6)
    exp.save_to_file("outputs/lime_explanation_RF.html")
    print("✅ LIME explanation saved as outputs/lime_explanation_RF.html")

    # --- LSTM ---
    if run_lstm:
        feature_cols = ["price","discount","rival_price","demand_forecast","inventory_level","marketing_spend","elasticity","se
        mm = MinMaxScaler()
        df_scaled = df.copy()
        df_scaled[feature_cols] = mm.fit_transform(df[feature_cols])
        df_train_s, df_test_s = df_scaled.iloc[:split_idx], df_scaled.iloc[split_idx:]
        X_train_seq, y_train_seq = create_lstm_sequences(df_train_s, feature_cols)
        X_test_seq, y_test_seq = create_lstm_sequences(df_test_s, feature_cols)

        if len(X_train_seq) > 0:
            model = build_lstm_model((X_train_seq.shape[1], X_train_seq.shape[2]))
            es = EarlyStopping(monitor="val_loss", patience=10, restore_best_weights=True)
            history = model.fit(X_train_seq, y_train_seq, validation_split=0.1, epochs=50, batch_size=64, callbacks=[es], verbc
            plot_training_history(history)
            y_pred_lstm = model.predict(X_test_seq).flatten()
            results["LSTM"] = compute_metrics(y_test_seq, y_pred_lstm)
            print("LSTM training complete.")

    plot_model_comparison(results)
    return results
# Run
if __name__ == "__main__":
    results = run_full_pipeline(run_lstm=True)
    print("\nFinal Results:")
    for k, v in results.items():
        print(k, ":", v)
```