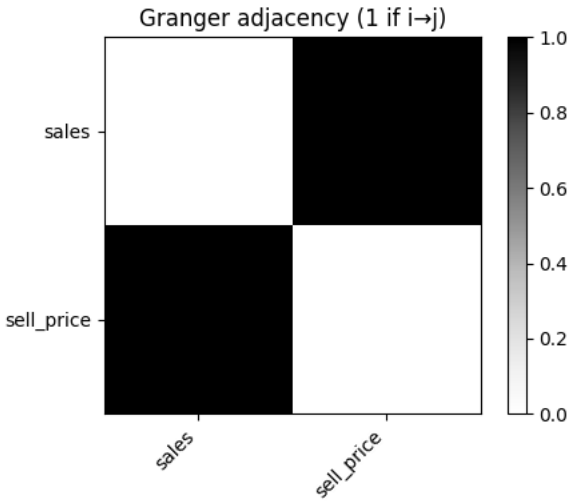
```
import os, warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from statsmodels.tsa.stattools import grangercausalitytests
from statsmodels.tsa.api import VAR

warnings.filterwarnings("ignore")
np.random.seed(42)

# ---------------- CONFIG ----------------
DATA_DIR   = "./"
SALES_EVAL = os.path.join(DATA_DIR, "sales_train_evaluation.csv")
CAL_PATH   = os.path.join(DATA_DIR, "calendar.csv")
PRICE_PATH = os.path.join(DATA_DIR, "sell_prices.csv")

STORE_FILTER = "CA_1"   # set None to auto-pick a store
TOPN_ITEMS   = 30
MAXLAG_GC    = 7
ALPHA_GC     = 0.05
VAR_LAG      = 2        # lag for VAR IRFs

# -------------- HELPERS -----------------
def melt_sales_wide_to_long(sales_wide: pd.DataFrame) -> pd.DataFrame:
    id_cols  = ["id","item_id","dept_id","cat_id","store_id","state_id"]
    day_cols = [c for c in sales_wide.columns if c.startswith("d_")]
    return sales_wide.melt(id_vars=id_cols, value_vars=day_cols,
                           var_name="d", value_name="sales")

def promo_from_price_drop(price, thr=-0.05):
    pct = price.pct_change()
    return (pct < thr).astype(int)

def drop_constant_cols(df):
    nun = df.nunique()
    keep = nun[nun > 1].index.tolist()
    return df[keep]

def compute_gc_matrix(X: np.ndarray, maxlag=5, alpha=0.05):
    T, d = X.shape
    adj = np.zeros((d, d))
    for i in range(d):
        for j in range(d):
            if i == j:
                continue
            arr = np.column_stack([X[:, j], X[:, i]])  # [target, cause]
            try:
                res  = grangercausalitytests(arr, maxlag=maxlag, verbose=False)
                pval = min(res[k][0]["ssr_ftest"][1] for k in range(1, maxlag+1))
                adj[i, j] = (pval < alpha)
            except Exception:
                pass
    return adj

# -------------- LOAD --------------------
sales_path = SALES_EVAL
sales_w = pd.read_csv(sales_path)
calendar = pd.read_csv(CAL_PATH, parse_dates=["date"])
prices = pd.read_csv(PRICE_PATH)

if STORE_FILTER:
    sales_w = sales_w[sales_w["store_id"] == STORE_FILTER]

# take top-N items by mean sales to keep small
day_cols = [c for c in sales_w.columns if c.startswith("d_")]
top_ids = (sales_w.assign(m=sales_w[day_cols].mean(axis=1))
           .sort_values("m", ascending=False).head(TOPN_ITEMS)["id"])
sales_w = sales_w[sales_w["id"].isin(top_ids)].reset_index(drop=True)

sales_l = melt_sales_wide_to_long(sales_w)
sales_l = sales_l.merge(calendar[["d","date","wm_yr_wk","event_name_1","event_name_2"]],
                        on="d", how="left")
sales_l = sales_l.merge(prices, on=["store_id","item_id","wm_yr_wk"], how="left")
sales_l = sales_l.sort_values(["item_id","store_id","date"]).reset_index(drop=True)
```

```
# build promo flag per (item,store)
sales_l["promo"] = (sales_l.groupby(["item_id","store_id"])["sell_price"]
                    .transform(promo_from_price_drop))

# pick the longest (item,store) series
key = sales_l.groupby(["item_id","store_id"])["date"].count().idxmax()
sub = sales_l[(sales_l["item_id"]==key[0]) & (sales_l["store_id"]==key[1])].copy()

# observed variables
df = sub[["date","sales","sell_price","promo"]].copy()
df = df.fillna(method="ffill").fillna(0.0)
Xdf = drop_constant_cols(df[["sales","sell_price","promo"]])
labels = list(Xdf.columns)
X = Xdf.values

print("[info] Series used:", key, "columns:", labels)

# --------- Granger causality (simple) ----------
adj = compute_gc_matrix(X, maxlag=MAXLAG_GC, alpha=ALPHA_GC)
print("[info] GC adjacency (1 if i→j). Order:", labels)
print(adj.astype(int))

plt.figure(figsize=(5,4))
plt.imshow(adj, cmap="Greys", interpolation="nearest")
plt.xticks(range(len(labels)), labels, rotation=45, ha="right")
plt.yticks(range(len(labels)), labels)
plt.title("Granger adjacency (1 if i→j)")
plt.colorbar()
plt.tight_layout()
plt.show()

# -------------- VAR (non-orthogonal IRFs) --------------
# No SVAR / Cholesky → avoids PD issues
model = VAR(X)
res = model.fit(VAR_LAG, trend="c")  # VAR with intercept
irf_analysis = res.irf(20)                    # impulse responses
# Use orth=False to avoid Cholesky factorization
irfs = irf_analysis.irfs                     # shape [h, d, d] # Corrected line

# Example: sales response to promo shock
if "sales" in labels and "promo" in labels:
    i_sales = labels.index("sales")
    j_promo = labels.index("promo")
    plt.figure(figsize=(6,3.6))
    plt.plot(irfs[:, i_sales, j_promo])
    plt.axhline(0, linewidth=1)
    plt.title("VAR IRF (non-orthogonal): Sales response to Promo shock")
    plt.xlabel("Horizon (days)"); plt.ylabel("Response")
    plt.tight_layout()
    plt.show()

print("[done] Simple GC + VAR pipeline finished.")
```

```
[info] Series used: ('FOODS_2_019', 'CA_1') columns: ['sales', 'sell_price']
[info] GC adjacency (1 if i→j). Order: ['sales', 'sell_price']
[[0 1]
 [1 0]]
```




```
[done] Simple GC + VAR pipeline finished.
```

```
# ---------------- NEW IMPORTS ----------------
from statsmodels.tsa.stattools import adfuller

# -------------- NEW HELPERS -----------------
def print_descriptive_stats(df):
    """Print basic statistics for the time series."""
    print("[info] Descriptive Statistics:")
    print(df.describe())

def check_stationarity(series, name):
    """Perform ADF test for stationarity."""
    result = adfuller(series.dropna())
    print(f"[info] ADF Test for {name}:")
    print(f"  ADF Statistic: {result[0]:.4f}")
    print(f"  p-value: {result[1]:.4f}")
    print(f"  Critical Values: {dict(result[4])}")
    is_stationary = result[1] < 0.05
    print(f"  Stationary: {is_stationary}\n")

def plot_rolling_mean(series, window=7, label=""):
    """Plot the rolling mean of a series."""
    rolling_mean = series.rolling(window=window).mean()
    plt.figure(figsize=(6,3.6))
    plt.plot(series, label=f"{label} (Original)")
    plt.plot(rolling_mean, label=f"{label} (Rolling Mean, {window} days)", color='red')
    plt.title(f"Rolling Mean of {label}")
    plt.xlabel("Time")
    plt.ylabel("Value")
    plt.legend()
    plt.tight_layout()
    plt.show()

# -------------- EXTENDED PIPELINE ----------------
# Descriptive Statistics
print_descriptive_stats(Xdf)

# Stationarity Checks
for col in Xdf.columns:
    check_stationarity(Xdf[col], col)

# Rolling Mean Plots
if "sales" in labels:
    plot_rolling_mean(Xdf["sales"], label="Sales")
if "sell_price" in labels:
    plot_rolling_mean(Xdf["sell_price"], label="Sell Price")

print("[done] Extended pipeline with stats, stationarity, and rolling mean plots finished.")
```

```
[info] Descriptive Statistics:
              sales   sell_price
count  1941.000000  1941.000000
mean     18.400824     3.228861
std      10.940235     0.159896
min       0.000000     2.980000
25%      12.000000     2.980000
50%      17.000000     3.280000
75%      25.000000     3.340000
max      64.000000     3.420000
[info] ADF Test for sales:
  ADF Statistic: -6.1408
  p-value: 0.0000
  Critical Values: {'1%': np.float64(-3.4337622297208146), '5%': np.float64(-2.863047304445204), '10%': np.float64(-2.567572430319552)
  Stationary: True

[info] ADF Test for sell_price:
  ADF Statistic: -1.3127
  p-value: 0.6234
  Critical Values: {'1%': np.float64(-3.4337252441664483), '5%': np.float64(-2.8630309758314314), '10%': np.float64(-2.56756373605059)
  Stationary: False
```