03_trends_seasonality_and_forecasting.ipynb

```python
#%%
import os
from IPython.display import display
import matplotlib.ticker as mticker
import matplotlib.dates as mdates
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np
from sklearn.linear_model import Ridge
from pathlib import Path
import pandas as pd
import matplotlib.pyplot as plt

#%%
repo_root = Path.cwd()
data_dir = (repo_root / "data") if (repo_root / "data").exists() else repo_root

paths = {
    'sales': data_dir / 'sales.csv',
    'stores': data_dir / 'stores.csv',
    'categories': data_dir / 'categories.csv',
    'dates': data_dir / 'dates.csv',
}

for k, p in paths.items():
    print(f"{k}: {p.exists()} -> {p.resolve()}")

def read_csv_smart(p):
    df = pd.read_csv(p, low_memory=False)
    df.columns = df.columns.str.lower().str.strip()
    date_cols = [c for c in df.columns if 'date' in c]
    for c in date_cols:
        df[c] = pd.to_datetime(df[c], errors='coerce')
    return df

sales_df      = read_csv_smart(paths['sales'])
stores_df     = read_csv_smart(paths['stores'])
categories_df = read_csv_smart(paths['categories'])
dates_df      = read_csv_smart(paths['dates'])

try:
    import holidays
    years = range(int(dates_df['date'].dt.year.min()), int(dates_df['date'].dt.year.max()) + 1)
    us_holidays = holidays.US(years=years)
    holiday_dates = pd.to_datetime(list(us_holidays.keys()))
    dates_df['is_holiday'] = dates_df['date'].isin(holiday_dates).astype(int)
    print("\nHoliday flag on dates_df (0=non-holiday, 1=holiday):")
    print(dates_df['is_holiday'].value_counts().sort_index())
except Exception as e:
    print("\n[warn] Could not add holiday flag (install `holidays`?):", e)
    dates_df['is_holiday'] = 0

try:
    from meteostat import Daily, Point
    import numpy as np

    start_dt = pd.to_datetime(dates_df['date'].min()).to_pydatetime()
    end_dt   = pd.to_datetime(dates_df['date'].max()).to_pydatetime()

    rep_point = None
    lat_cols = [c for c in stores_df.columns if c.lower() in ('lat','latitude')]
    lon_cols = [c for c in stores_df.columns if c.lower() in ('lon','lng','longitude')]

    if lat_cols and lon_cols:
        lat_med = pd.to_numeric(stores_df[lat_cols[0]], errors='coerce').median()
        lon_med = pd.to_numeric(stores_df[lon_cols[0]], errors='coerce').median()
        if pd.notna(lat_med) and pd.notna(lon_med):
            rep_point = Point(float(lat_med), float(lon_med))
            print(f"[weather] Using median store coords: lat={lat_med:.4f}, lon={lon_med:.4f}")

    if rep_point is None:
        rep_point = Point(39.8283, -98.5795)
        print("[weather] Using US centroid as representative location.")
        wx = Daily(rep_point, start_dt, end_dt)
        wx_full = wx.fetch()
        wx_df = (
            wx_full.reset_index()
                   .rename(columns={'time': 'date'})
                   .loc[:, ['date', 'tavg', 'tmin', 'tmax', 'prcp']]
        )
        wx_df['date'] = pd.to_datetime(wx_df['date'])

        if 'tavg' not in wx_df or wx_df['tavg'].isna().all():
            if {'tmin','tmax'}.issubset(wx_df.columns):
                wx_df['tavg'] = wx_df[['tmin','tmax']].mean(axis=1)
            else:
                doy = wx_df['date'].dt.dayofyear
                wx_df['tavg'] = (np.sin(2*np.pi*(doy/365.25)) * 10 + 13).round(1)
        if 'prcp' not in wx_df:
            wx_df['prcp'] = 0.0

        wx_df = wx_df[['date', 'tavg', 'prcp']]

        dates_df = dates_df.merge(wx_df, on='date', how='left')

        dates_df['tavg'] = dates_df['tavg'].ffill().bfill()
        dates_df['prcp'] = dates_df['prcp'].fillna(0.0)

        print("\nWeather columns added to dates_df (head):")
        print(dates_df[['date','tavg','prcp']].head())

    except Exception as e:
        print("[warn] Could not add weather features; proceeding without. Error:", e)
        try:
            import numpy as np
        except Exception:
            pass
        if 'tavg' not in dates_df.columns:
            if 'np' in globals():
                doy = dates_df['date'].dt.dayofyear
                dates_df['tavg'] = (np.sin(2*np.pi*(doy/365.25)) * 20 + 55).round(1)
            else:
                dates_df['tavg'] = pd.NA
        if 'prcp' not in dates_df.columns:
            dates_df['prcp'] = 0.0

    for name, df in [('sales', sales_df), ('stores', stores_df),
                     ('categories', categories_df), ('dates', dates_df)]:
        print(f"\n{name}: {df.shape}")
        display(df.head(3))
        display(df.dtypes.to_frame('dtype').T)

    #%%
    sales2_df = (sales_df
                 .merge(dates_df, on='date', how='left')
                 .merge(stores_df, on='store_id', how='left')
                 .merge(categories_df, on='category_id', how='left'))

    sales2_df['weekend'] = sales2_df['dow'].isin([5,6]).astype(int)

    print('shape:', sales2_df.shape)
    print(sales2_df.columns.tolist())

    print('\ndow counts:')
    print(sales2_df['dow'].value_counts().sort_index())

    print('\nweekend counts:')
    print(sales2_df['weekend'].value_counts().sort_index())

    daily_rev_df = (sales2_df.groupby('date')['sales_revenue']
                    .sum().reset_index())
    display(daily_rev_df.head())

    #%%
    sales = pd.read_csv("sales.csv", parse_dates=["date"])
    cats = pd.read_csv("categories.csv")
    df = sales.merge(cats, on="category_id")
    df["doy"] = df["date"].dt.dayofyear
    print("Category peak DOY:\n",
      df.groupby(["name","doy"])["sales_revenue"].mean()
        .reset_index()
        .loc[lambda x: x.groupby("name")["sales_revenue"].idxmax(), ["name","doy"]]
        .sort_values("name"))

#%%
daily_rev_df = sales2_df.groupby('date', as_index=False)['sales_revenue'].sum()

plt.figure(figsize=(12,4))
plt.plot(daily_rev_df['date'], daily_rev_df['sales_revenue'])
plt.title('daily revenue (all stores)')
plt.xlabel('date'); plt.ylabel('revenue')
plt.tight_layout()
plt.show()

dow_avg_df = (sales2_df.groupby('dow')['sales_revenue']
              .sum().div(sales2_df.groupby('dow')['sales_revenue'].count())
              .reset_index(name='avg_rev'))
plt.figure(figsize=(6,4))
plt.bar(dow_avg_df['dow'], dow_avg_df['avg_rev'])
plt.title('avg revenue by day-of-week (0=Mon)')
plt.xlabel('dow'); plt.ylabel('avg revenue')
plt.tight_layout()
plt.show()

wk_df = (sales2_df.groupby(['weekend','date'])['sales_revenue']
         .sum().reset_index())
wk_pvt_df = wk_df.pivot(index='date', columns='weekend', values='sales_revenue').rename(columns={0:'weekday',1:'weekend'})

plt.figure(figsize=(12,4))
plt.plot(wk_pvt_df.index, wk_pvt_df['weekday'], label='weekday')
plt.plot(wk_pvt_df.index, wk_pvt_df['weekend'], label='weekend', alpha=0.8)
plt.title('weekday vs weekend revenue')
plt.xlabel('date'); plt.ylabel('revenue'); plt.legend()
plt.tight_layout()
plt.show()

#%%
daily_rev_df = (sales2_df
                .assign(date=pd.to_datetime(sales2_df['date']))
                .groupby('date', as_index=False)['sales_revenue']
                .sum()
                .sort_values('date'))

daily_rev_df['rev_30dma'] = (daily_rev_df['sales_revenue']
                             .shift(1)                          # no peeking
                             .rolling(window=30, min_periods=7) # require some history
                             .mean())

plt.style.use("dark_background")
fig, ax = plt.subplots(figsize=(12, 4))

ax.plot(daily_rev_df['date'], daily_rev_df['sales_revenue'], alpha=0.8, label='daily')

ax.plot(daily_rev_df['date'], daily_rev_df['rev_30dma'], linewidth=2.2, label='30-day avg')

ax.set_title('daily revenue (all stores)')
ax.set_xlabel('date')
ax.set_ylabel('revenue')

ax.xaxis.set_minor_locator(mdates.MonthLocator(interval=3))
ax.xaxis.set_major_locator(mdates.YearLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

ax.yaxis.set_major_locator(mticker.MultipleLocator(1000))

ax.grid(True, which='major', linestyle='--', color='gray', alpha=0.4)
ax.grid(True, which='minor', linestyle=':', color='gray', alpha=0.2)

ax.legend()
plt.tight_layout()
plt.show()

#%%
wk_df = (
    sales2_df.groupby(['weekend', 'date'])['sales_revenue']
    .sum()
```

```
        .reset_index()
    )

    wk_pvt_df = (
        wk_df.pivot(index='date', columns='weekend', values='sales_revenue')
        .rename(columns={0: 'weekday', 1: 'weekend'})
    )

    plt.style.use("dark_background")
    fig, ax = plt.subplots(figsize=(12, 4))

    ax.plot(wk_pvt_df.index, wk_pvt_df['weekday'], label='weekday', linewidth=1.5)
    ax.plot(wk_pvt_df.index, wk_pvt_df['weekend'], label='weekend', linewidth=1.5, alpha=0.8)

    ax.set_title('weekday vs weekend revenue')
    ax.set_xlabel('date')
    ax.set_ylabel('revenue')

    ax.xaxis.set_minor_locator(mdates.MonthLocator(interval=3))

    ax.xaxis.set_major_locator(mdates.YearLocator())
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y'))

    ax.yaxis.set_major_locator(plt.MultipleLocator(1000))

    ax.grid(True, which='major', linestyle='--', color='gray', alpha=0.4)
    ax.grid(True, which='minor', linestyle=':', color='gray', alpha=0.2)

    ax.legend()
    plt.tight_layout()
    plt.show()

    #%%
    monthly_df = (sales2_df.groupby(['year','month'])['sales_revenue']
                  .sum().reset_index())
    monthly_df['date'] = pd.to_datetime(monthly_df['year'].astype(str) + '-' + monthly_df['month'].astype(str) + '-01')

    plt.figure(figsize=(12,4))
    plt.plot(monthly_df['date'], monthly_df['sales_revenue'])
    plt.title('monthly revenue (all stores)')
    plt.xlabel('month'); plt.ylabel('revenue')
    plt.tight_layout()
    plt.show()

    cat_df = (sales2_df.groupby(['category_id','year','month'])['sales_revenue']
              .sum().reset_index())
    cat_df['date'] = pd.to_datetime(cat_df['year'].astype(str) + '-' + cat_df['month'].astype(str) + '-01')
    cat_pvt_df = cat_df.pivot(index='date', columns='category_id', values='sales_revenue')

    cat_pvt_df.plot(figsize=(12,4))
    plt.title('monthly revenue by category')
    plt.xlabel('month'); plt.ylabel('revenue')
    plt.legend(title='category_id')
    plt.tight_layout()
    plt.show()

    #%%
    monthly_df = (
        sales2_df.groupby(['year','month'])['sales_revenue']
        .sum()
        .reset_index()
    )
    monthly_df['date'] = pd.to_datetime(
        monthly_df['year'].astype(str) + '-' + monthly_df['month'].astype(str) + '-01'
    )

    plt.style.use("dark_background")
    plt.figure(figsize=(12,4))
    plt.plot(monthly_df['date'], monthly_df['sales_revenue'], linewidth=1.8)
    plt.title('monthly revenue (all stores)')
    plt.xlabel('month')
    plt.ylabel('revenue')
    plt.tight_layout()
    plt.show()

    cat_df = (
        sales2_df.groupby(['category_id','year','month'])['sales_revenue']
    .sum()
    .reset_index()
)
cat_df['date'] = pd.to_datetime(
    cat_df['year'].astype(str) + '-' + cat_df['month'].astype(str) + '-01'
)

cat_pvt_df = cat_df.pivot(index='date', columns='category_id', values='sales_revenue')

category_names = {1: 'Beverages', 2: 'Bakery', 3: 'Meat', 4: 'Produce'}
cat_pvt_df = cat_pvt_df.rename(columns=category_names)

cat_pvt_df.plot(figsize=(12,4), linewidth=1.5)
plt.title('monthly revenue by category')
plt.xlabel('month')
plt.ylabel('revenue')
plt.legend(title='category')
plt.tight_layout()
plt.show()

#%%
store_m_df = (sales2_df.groupby(['store_id','year','month'])['sales_revenue']
              .sum().reset_index())
store_m_df['date'] = pd.to_datetime(store_m_df['year'].astype(str) + '-' + store_m_df['month'].astype(str) + '-01')
store_pvt_df = store_m_df.pivot(index='date', columns='store_id', values='sales_revenue')

store_pvt_df.plot(figsize=(12,4))
plt.title('monthly revenue by store')
plt.xlabel('month'); plt.ylabel('revenue')
plt.legend(title='store_id'); plt.tight_layout(); plt.show()

reg_m_df = (sales2_df.groupby(['region','year','month'])['sales_revenue']
            .sum().reset_index())
reg_m_df['date'] = pd.to_datetime(reg_m_df['year'].astype(str) + '-' + reg_m_df['month'].astype(str) + '-01')
reg_pvt_df = reg_m_df.pivot(index='date', columns='region', values='sales_revenue')

reg_pvt_df.plot(figsize=(12,4))
plt.title('monthly revenue by region')
plt.xlabel('month'); plt.ylabel('revenue')
plt.legend(title='region'); plt.tight_layout(); plt.show()

#%%
ts_df = (sales2_df.groupby('date', as_index=False)['sales_revenue']
         .sum().rename(columns={'sales_revenue':'y'}))

full_idx = pd.date_range(ts_df['date'].min(), ts_df['date'].max(), freq='D')
ts_df = (pd.DataFrame({'date': full_idx})
         .merge(ts_df, on='date', how='left')
         .fillna({'y':0}))

feat_cols = ['date','dow','woy','month','quarter','year','is_holiday','tavg','prcp']
ts_df = ts_df.merge(dates_df[feat_cols], on='date', how='left')
ts_df['weekend'] = ts_df['dow'].isin([5,6]).astype(int)

for lag in [1,7,14,28]:
    ts_df[f'lag_{lag}'] = ts_df['y'].shift(lag)

for win in [7, 14, 28]:
    ts_df[f'ma_{win}'] = ts_df['y'].shift(1).rolling(win).mean()

ts_df = ts_df.dropna().reset_index(drop=True)

print(ts_df.shape)
display(ts_df.head())

#%%
try:
    from sklearn.metrics import root_mean_squared_error as rmse_fn
except ImportError:
    from sklearn.metrics import mean_squared_error
    rmse_fn = lambda y_true, y_pred: mean_squared_error(y_true, y_pred, squared=False)

cut = int(len(ts_df)*0.8)
tr, te = ts_df.iloc[:cut], ts_df.iloc[cut:]

x_tr = tr.drop(columns=['date','y']); y_tr = tr['y']
x_te = te.drop(columns=['date','y']); y_te = te['y']
```

```
m = LinearRegression().fit(x_tr, y_tr)
y_hat = m.predict(x_te)

mae = mean_absolute_error(y_te, y_hat)
rmse = rmse_fn(y_te, y_hat)

print('mae:', round(mae,2))
print('rmse:', round(rmse,2))

#%%
def mape(y, yhat):
    y, yhat = np.asarray(y), np.asarray(yhat)
    return np.mean(np.abs((y - yhat) / np.clip(np.abs(y), 1e-8, None))) * 100

def smape(y, yhat):
    y, yhat = np.asarray(y), np.asarray(yhat)
    d = np.abs(y - yhat) / np.clip((np.abs(y) + np.abs(yhat)) / 2, 1e-8, None)
    return np.mean(d) * 100

def bt_lr(df, h=14, start_frac=0.6, feats=('lag_7','ma_7','ma_14','ma_28','weekend','is_holiday','tavg','prcp')):
    df = df.sort_values('date').reset_index(drop=True)
    n = len(df)
    start = int(n * start_frac)
    out = []
    last_fold = None

    for t in range(start, n - h):
        tr = df.iloc[:t].dropna(subset=list(feats))
        te = df.iloc[t:t+h]

        x_tr = tr[list(feats)].values
        y_tr = tr['y'].values
        x_te = te[list(feats)].values
        y_te = te['y'].values

        m = LinearRegression().fit(x_tr, y_tr)
        y_hat = m.predict(x_te)
        y_naive = te['lag_7'].values

        mae = mean_absolute_error(y_te, y_hat)
        rmse = np.sqrt(mean_squared_error(y_te, y_hat))

        mape_v = mape(y_te, y_hat)
        smape_v = smape(y_te, y_hat)

        mae_n = mean_absolute_error(y_te, y_naive)
        rmse_n = np.sqrt(mean_squared_error(y_te, y_naive))
        mape_n = mape(y_te, y_naive)
        smape_n = smape(y_te, y_naive)

        out.append({'t0': df.loc[t, 'date'], 'mae': mae, 'rmse': rmse,
                    'mape': mape_v, 'smape': smape_v,
                    'mae_naive': mae_n, 'rmse_naive': rmse_n,
                    'mape_naive': mape_n, 'smape_naive': smape_n})

        last_fold = (te['date'].values, y_te, y_hat, y_naive, mae, rmse)

    res = (
        np.mean([r['mae'] for r in out]),
        np.mean([r['rmse'] for r in out]),
        np.mean([r['mape'] for r in out]),
        np.mean([r['smape'] for r in out]),
        np.mean([r['mae_naive'] for r in out]),
        np.mean([r['rmse_naive'] for r in out]),
        np.mean([r['mape_naive'] for r in out]),
        np.mean([r['smape_naive'] for r in out]),
        len(out)
    )

    dts, y_true, y_hat, y_nv, mae_last, rmse_last = last_fold
    plt.figure(figsize=(10, 4))
    plt.plot(dts, y_true, label='Actual (y)', color='skyblue', linewidth=2)
    plt.plot(dts, y_hat, label='Linear Regression', color='orange', linewidth=2)
    plt.plot(dts, y_nv, label='Naive-7', color='limegreen', alpha=0.8)

    plt.fill_between(dts, y_hat - mae_last, y_hat + mae_last,
                     color='orange', alpha=0.2, label='±MAE')
    plt.fill_between(dts, y_hat - rmse_last, y_hat + rmse_last,
                     color='red', alpha=0.1, label='±RMSE')

    plt.legend()
    plt.title("Last Fold Forecast with ±MAE and ±RMSE Bands")
    plt.xlabel("Date"); plt.ylabel("Revenue")
    plt.tight_layout()
    plt.show()

    print('folds:', res[-1])
    print('lr  -> mae:', round(res[0],2), 'rmse:', round(res[1],2),
          'mape:', round(res[2],2), 'smape:', round(res[3],2))
    print('nv7 -> mae:', round(res[4],2), 'rmse:', round(res[5],2),
          'mape:', round(res[6],2), 'smape:', round(res[7],2))

bt_lr(ts_df, h=14, start_frac=0.6, feats=('lag_7','ma_7','ma_14','ma_28'))

#%%
def mape(y, yhat):
    y, yhat = np.asarray(y), np.asarray(yhat)
    return np.mean(np.abs((y - yhat) / np.clip(np.abs(y), 1e-8, None))) * 100

def smape(y, yhat):
    y, yhat = np.asarray(y), np.asarray(yhat)
    return np.mean(np.abs(y - yhat) / np.clip((np.abs(y) + np.abs(yhat)) / 2, 1e-8, None)) * 100

def walk_cv(df, h=14, start_frac=0.6, feats=('lag_7','ma_7','ma_14','ma_28')):
    df = df.sort_values('date').reset_index(drop=True)
    n = len(df)
    start = int(n * start_frac)
    folds = []
    for t in range(start, n - h):
        tr = df.iloc[:t].dropna(subset=list(feats))
        te = df.iloc[t:t+h]
        x_tr, y_tr = tr[list(feats)].values, tr['y'].values
        x_te, y_te = te[list(feats)].values, te['y'].values
        folds.append((x_tr, y_tr, x_te, y_te, te['lag_7'].values))
    return folds

def eval_model(model, folds):
    mets = []
    for x_tr, y_tr, x_te, y_te, naive7 in folds:
        m = model.fit(x_tr, y_tr)
        y_hat = m.predict(x_te)
        mets.append({
            'mae': mean_absolute_error(y_te, y_hat),
            'rmse': np.sqrt(mean_squared_error(y_te, y_hat)),
            'mape': mape(y_te, y_hat),
            'smape': smape(y_te, y_hat),
            'mae_nv7': mean_absolute_error(y_te, naive7),
            'rmse_nv7': np.sqrt(mean_squared_error(y_te, naive7)),
            'mape_nv7': mape(y_te, naive7),
            'smape_nv7': smape(y_te, naive7),
        })
    out = pd.DataFrame(mets).mean().to_dict()
    return {k: round(v, 2) for k, v in out.items()}

feats = ('lag_7','ma_7','ma_14','ma_28','weekend','is_holiday','tavg','prcp')
folds = walk_cv(ts_df, h=14, start_frac=0.6, feats=feats)

res = []
res.append({'model':'linreg', **eval_model(LinearRegression(), folds)})
res.append({'model':'ridge',  **eval_model(Ridge(alpha=1.0, random_state=42), folds)})
res.append({'model':'rf',     **eval_model(RandomForestRegressor(
    n_estimators=300, max_depth=8, random_state=42, n_jobs=-1), folds)})

res_df = pd.DataFrame(res).sort_values('rmse').reset_index(drop=True)
print(res_df)

#%%
feats = ['lag_7','ma_7','ma_14','ma_28','weekend','is_holiday','tavg','prcp']

hist_df = (
    ts_df.sort_values('date')
         .reset_index(drop=True)
         .dropna(subset=feats)
        .copy()
)
X = hist_df[feats].values
y = hist_df['y'].values
model = Ridge(alpha=1.0, random_state=42).fit(X, y)

h = 28
last_date = hist_df['date'].iloc[-1]
future_dates = pd.date_range(last_date + pd.Timedelta(days=1), periods=h, freq='D')

fut_df = pd.DataFrame({'date': future_dates})
fut_df['dow']     = fut_df['date'].dt.dayofweek
fut_df['woy']     = fut_df['date'].dt.isocalendar().week.astype(int)
fut_df['month']   = fut_df['date'].dt.month
fut_df['quarter'] = fut_df['date'].dt.quarter
fut_df['year']    = fut_df['date'].dt.year
fut_df['weekend'] = fut_df['dow'].isin([5, 6]).astype(int)

import holidays
years = range(int(hist_df['date'].dt.year.min()), int(future_dates[-1].year) + 1)
us_holidays = holidays.US(years=years)
holiday_dates = pd.to_datetime(list(us_holidays.keys()))
fut_df['is_holiday'] = fut_df['date'].isin(holiday_dates).astype(int)

hist_tavg = dates_df.set_index('date')['tavg'].asfreq('D').ffill() if 'tavg' in dates_df else pd.Series(dtype=float)
hist_prcp = dates_df.set_index('date')['prcp'].asfreq('D').fillna(0.0) if 'prcp' in dates_df else pd.Series(dtype=float)
tavg_seed = float(hist_tavg.tail(28).mean()) if not hist_tavg.empty else 55.0
prcp_seed = float(hist_prcp.tail(28).mean()) if not hist_prcp.empty else 0.0
fut_df['tavg'] = tavg_seed
fut_df['prcp'] = prcp_seed

hist_series = (
    hist_df[['date','y']]
      .set_index('date')['y']
      .asfreq('D')
      .ffill()
)

def roll_feat(s: pd.Series) -> pd.DataFrame:
    return pd.DataFrame({
        'lag_7': s.shift(7),
        'ma_7':  s.rolling(7).mean(),
        'ma_14': s.rolling(14).mean(),
        'ma_28': s.rolling(28).mean()
    })

work = pd.DataFrame({'y': hist_series.copy()})
preds = []
for d in future_dates:
    base = roll_feat(work['y']).iloc[-1][['lag_7','ma_7','ma_14','ma_28']].to_frame().T
    base = base.ffill(axis=1).bfill(axis=1)

    cal = fut_df.loc[fut_df['date'] == d, ['weekend','is_holiday','tavg','prcp']].reset_index(drop=True)

    r = pd.concat([base.reset_index(drop=True), cal], axis=1)[feats]

    yhat = model.predict(r.values).item()
    work.loc[d, 'y'] = yhat
    preds.append(yhat)

naive7 = hist_series.reindex(future_dates - pd.Timedelta(days=7)).values

out_df = pd.DataFrame({'date': future_dates, 'yhat_ridge': preds, 'naive7': naive7})

Path('reports').mkdir(exist_ok=True, parents=True)
out_df.to_csv('reports/forecast_28d.csv', index=False)
print('wrote reports/forecast_28d.csv')
display(out_df.head())

plt.figure(figsize=(10,4))
tail_hist = hist_df.tail(90)[['date','y']]
plt.plot(tail_hist['date'], tail_hist['y'], label='y')
plt.plot(out_df['date'], out_df['yhat_ridge'], label='forecast')
plt.plot(out_df['date'], out_df['naive7'], label='naive7', alpha=0.6)
plt.title('28d forecast (ridge)')
plt.legend(); plt.tight_layout(); plt.show()
```

```python
#%%
Path("reports").mkdir(parents=True, exist_ok=True)

last_dt = ts_df['date'].max()
h = len(out_df)

hist_tail = ts_df[ts_df['date'].between(last_dt - pd.Timedelta(days=90), last_dt)][['date','y']].copy()

plt.figure(figsize=(10,4))
plt.plot(hist_tail['date'], hist_tail['y'], label='y')
plt.plot(out_df['date'], out_df['yhat_ridge'], label='forecast')
plt.plot(out_df['date'], out_df['naive7'], label='naive7', alpha=0.6)
plt.title(f'{h}d forecast (ridge)')
plt.legend(); plt.tight_layout()
plt.savefig('reports/forecast_plot.png', dpi=150)
plt.close()

fc_sum = out_df['yhat_ridge'].sum()
nv_sum = out_df['naive7'].sum()
hist_28 = ts_df[ts_df['date'].between(last_dt - pd.Timedelta(days=27), last_dt)]['y'].sum()

same_start = (last_dt - pd.Timedelta(days=365 - (h - 1))).normalize()
same_end = same_start + pd.Timedelta(days=h - 1)
yoy_hist = ts_df[ts_df['date'].between(same_start, same_end)]['y'].sum()

summ = pd.DataFrame({
    'metric': ['forecast_sum', 'naive7_sum', 'last_28d_hist', 'same_window_prev_year'],
    'value': [fc_sum, nv_sum, hist_28, yoy_hist]
})
summ.to_csv('reports/forecast_summary.csv', index=False)

with open('reports/forecast_readme.md', 'w') as f:
    f.write(
        f"# forecast artifacts\n\n"
        f"- plot: `reports/forecast_plot.png`\n"
        f"- table: `reports/forecast_28d.csv`\n"
        f"- summary: `reports/forecast_summary.csv`\n\n"
        f"## quick nums\n"
        f"- forecast {h}d sum: {fc_sum:,.0f}\n"
        f"- naive7 {h}d sum: {nv_sum:,.0f}\n"
        f"- last 28d hist sum: {hist_28:,.0f}\n"
        f"- same window prev yr sum: {yoy_hist:,.0f}\n"
    )

print("wrote reports/forecast_plot.png, reports/forecast_summary.csv, reports/forecast_readme.md")

#%%
```