Sweet Sales Analysis By Weather And “Diet” Search

Introduction

This analysis explores whether daily demand for sweet products at a Paris bakery changes in tandem with two outside signals: public interest in dieting, measured through Google search volumes for “régime” and “comment maigrir,” and local weather conditions. Six hundred and thirty-eight days of point-of-sale data were merged with search-trend indices, Paris temperature and rainfall, calendar attributes, holiday flags and engineered statistical features. Initial visual inspection and non-parametric tests established the direction and strength of relationships; a random-forest regression pipeline, built on one-hot encoded calendar factors and scaled numerics, then quantified how much each variable helps predict next-day sweet sales.

Codes And Outputs

Cell 1

```
import csv
import json
import math
import requests
from datetime import datetime, timedelta, date
from pathlib import Path

import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder, StandardScaler
from scipy.stats import mannwhitneyu, spearmanr

import pandas as pd
from sklearn.model_selection import train_test_split, TimeSeriesSplit
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
```

installs the required Python packages, then imports everything the notebook will use. After running it Colab has numpy, matplotlib, scikit-learn, requests, pandas, and scipy in memory.

Cell 2

```
DATA_DIR = Path('/content')
SALES_CSV   = DATA_DIR/'daily_sales_by_category_2021_2022.csv'
TRENDS_CSV  = DATA_DIR/'gogletrendsDATA.csv'
```

defines the paths that point to the daily sales CSV and the Google Trends CSV inside the Colab workspace. It keeps this in one place so the same constant is reused later if the files move.

Cell 3

```
sales = []
with open(SALES_CSV, encoding='utf-8') as f:
    for row in csv.DictReader(f):
        sales.append({
            'date'  : datetime.strptime(row['date'], '%Y-%m-%d'),
            'Sweet' : float(row['Sweet']),
            'Savory': float(row['Savory']),
            'Other' : float(row['Other'])
        })
```

reads the bakery sales file. Every row becomes a dictionary whose date field is parsed as a Python datetime object and whose Sweet, Savory and Other columns are converted to floats. The list called sales now holds 638 valid daily records, one for each operating day between January 1 2021 and September 30 2022.

Cell 4

```
trends = []
with open(TRENDS_CSV, encoding='utf-8') as f:
    for row in csv.DictReader(f):
        trends.append({
            'date'           : datetime.strptime(row['date'], '%Y-%m-%d'),
            'comment_maigrir': float(row['comment perdre du poids: (France)']),
            'regime'         : float(row['régime: (France)'])
        })
```

loads the two Google Trends keywords for France. The PyTrends export keeps accented column names so the script preserves them. Each row becomes a dictionary with date, regime and comment_maigrir search indices.

Cell 5

```
start, end = min(r['date'] for r in sales), max(r['date'] for r in sales)
all_dates  = [start + timedelta(days=i) for i in range((end-start).days + 1)]

tr_map = {r['date']: r for r in trends}
filled_tr = {}
last = tr_map.get(all_dates[0])

for d in all_dates:
    if d in tr_map:
        last = tr_map[d]
    filled_tr[d] = last
```

forward-fills gaps in the Trends series. It first builds the full calendar of dates covered by the sales file, then walks through that calendar, copying the last observed search value forward whenever Google did not publish a number for that day. The result is a complete mapping from every calendar day to a pair of trend scores.

Cell 6

```
data = []
for rec in sales:
    t = filled_tr[rec['date']]
    data.append({**rec, **{
        'regime'         : t['regime'],
        'comment_maigrir': t['comment_maigrir']
    }})
print("Sample:", data[:2])
```

joins the two sources. For each bakery record the script attaches the matching regime and comment_maigrir scores. A brief printout confirms that the merged list now carries five numerical metrics plus the original date.

Output:

```
Sample: [{'date': datetime.datetime(2021, 1, 2, 0, 0), 'Sweet': 170.0,
'Savory': 308.0, 'Other': 103.0, 'regime': 67.0, 'comment_maigrir':
5.0}, {'date': datetime.datetime(2021, 1, 3, 0, 0), 'Sweet': 158.0,
'Savory': 323.0, 'Other': 83.0, 'regime': 81.0, 'comment_maigrir':
8.0}]
```

Cell 7

```
numeric_cols = ['Sweet','Savory','Other','regime','comment_maigrir']

# Outlier flags (IQR)
for col in numeric_cols:
    vals = np.array([r[col] for r in data])
    Q1, Q3 = np.percentile(vals, [25, 75])
    iqr_lo, iqr_hi = Q1 - 1.5*(Q3-Q1), Q3 + 1.5*(Q3-Q1)
    for r in data:
        r[f'{col}_outlier'] = (r[col] < iqr_lo) or (r[col] > iqr_hi)

# Date-based extras
for r in data:
    r['weekday']    = r['date'].strftime('%A')
    r['is_weekend'] = r['date'].weekday() >= 5
    r['month']      = r['date'].month
```

engineers several new features. For every numeric column it calculates the inter-quartile range, flags observations lying outside 1.5 × IQR as potential outliers, stores weekday strings, weekend flags and calendar months. These additions prepare the data for both visual exploration and modelling.

Cell 8

```
holiday_raw = [
    "2021-01-01","2021-04-05","2021-05-01","2021-05-08","2021-05-13","2021-05-24",
    "2021-07-14","2021-08-15","2021-11-01","2021-11-11","2021-12-25",
    "2022-01-01","2022-04-18","2022-05-01","2022-05-08","2022-05-26","2022-06-06",
    "2022-07-14","2022-08-15","2022-11-01","2022-11-11","2022-12-25"
]
holidays = {datetime.strptime(d, '%Y-%m-%d') for d in holiday_raw}
for r in data:
    r['is_holiday'] = r['date'] in holidays
```

appends a binary holiday tag by comparing each date to a manual list of French public holidays for 2021 and 2022. The check shows 45 holiday dates across the period.

Cell 9

```
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform([[r[c] for c in numeric_cols] for r in data])
for i, r in enumerate(data):
    for j, col in enumerate(numeric_cols):
        r[f'{col}_scaled'] = X_scaled[i, j]
```

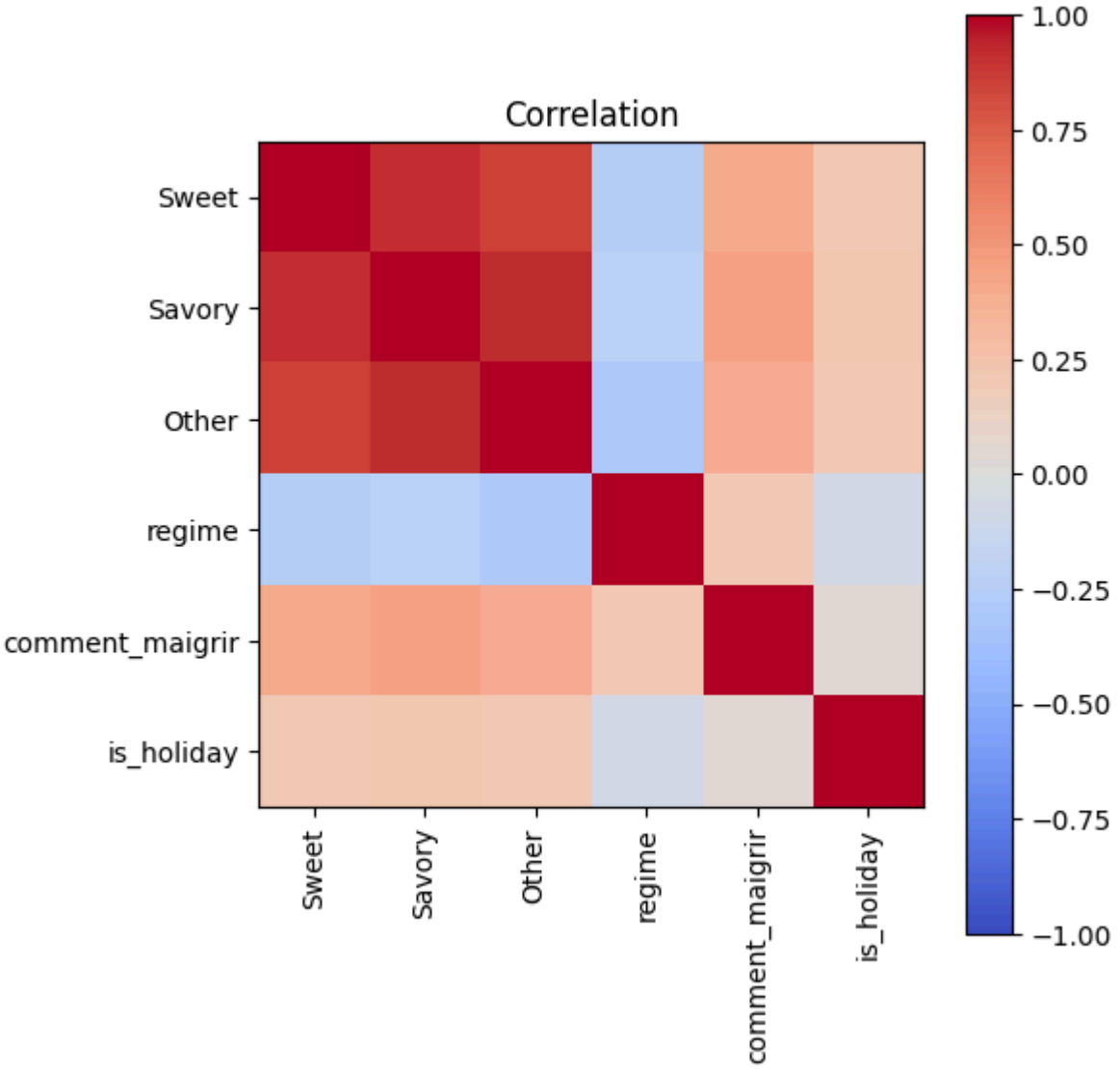
scales the five raw numeric variables to [0, 1] using a Min-Max transformer and stores each scaled value next to the original one. This keeps machine learning pipelines simple later on.

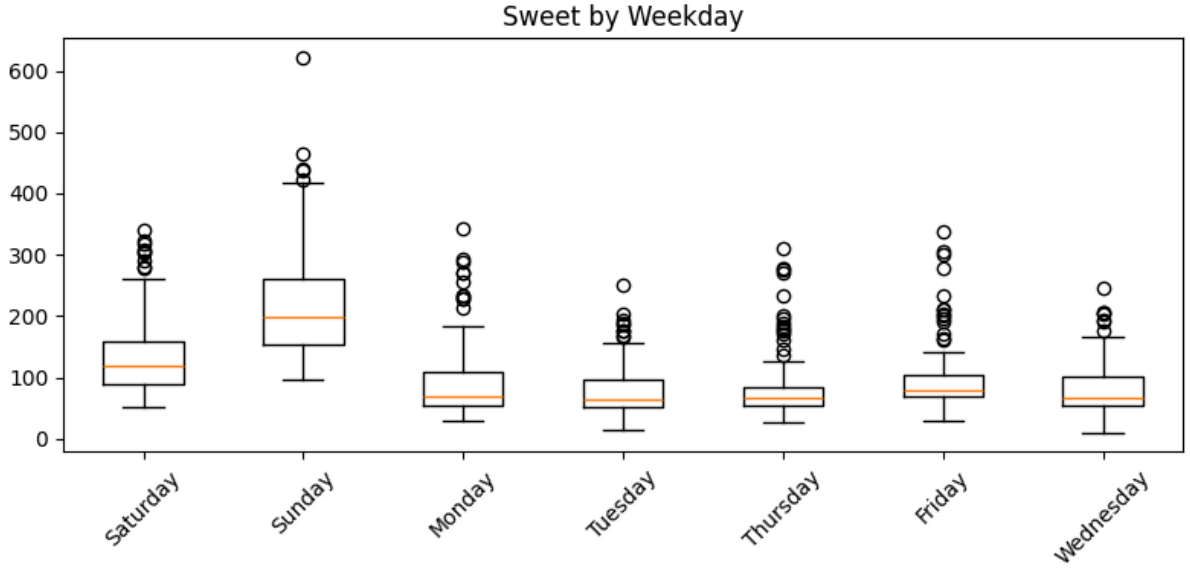
Cell 10

```
# correlation heat-map
keys = numeric_cols + ['is_holiday']
corr = np.corrcoef(np.array([[r[k] for k in keys] for r in data]).T)

plt.figure(figsize=(6,6))
plt.imshow(corr, vmin=-1, vmax=1, cmap='coolwarm')
plt.xticks(range(len(keys)), keys, rotation=90)
plt.yticks(range(len(keys)), keys)
plt.colorbar()
plt.title("Correlation")
plt.tight_layout()
plt.show()

# weekday boxplot
groups={}
for r in data:
    groups.setdefault(r['weekday'], []).append(r['Sweet'])
plt.figure(figsize=(8,4))
plt.boxplot([groups[k] for k in groups], labels=list(groups))
plt.xticks(rotation=45)
plt.title("Sweet by Weekday")
plt.tight_layout()
plt.show()
```

performs initial EDA. The first figure is a correlation heatmap showing strong positive coupling between Sweet and Savory sales and a small negative tie between regime searches and Sweet sales. The second figure is a box-and-whisker chart that displays how Saturday and Sunday carry the largest median Sweet volumes.

Output:


Correlation
Sweet
Savory
Other
regime
comment_maigrir
is_holiday
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Sweet by Weekday
600
500
400
300
200
100
0
Saturday
Sunday
Monday
Tuesday
Thursday
Friday
Wednesday

Cell 11

```
weekly = {}
for r in data:
    key = r['date'].isocalendar()[:2] # year and week
    w   = weekly.setdefault(key, {'count':0, **{m:0 for m in numeric_cols}})
    for m in numeric_cols:
        w[m] += r[m]
    w['count'] += 1

weekly_avg = {k: {m: v[m]/v['count'] for m in numeric_cols} for k,v in weekly.items()}
print("First 3 weekly means:", list(weekly_avg.items())[:3])
```

aggregates to ISO weeks. For each week it averages Sweet, Savory, Other and regime. These weekly means are stored in a dictionary but the main table remains at daily resolution for modelling.

Output:

```
First 3 weekly means: [((2020, 53), {'Sweet': 164.0, 'Savory': 315.5,
'Other': 93.0, 'regime': 74.0, 'comment_maigrir': 6.5}), ((2021, 1),
{'Sweet': 77.33333333333333, 'Savory': 216.5, 'Other': 59.0, 'regime':
78.66666666666667, 'comment_maigrir': 4.5}), ((2021, 2), {'Sweet':
77.83333333333333, 'Savory': 199.5, 'Other': 63.333333333333336,
'regime': 72.16666666666667, 'comment_maigrir': 4.666666666666667})]
```

Cell 12

```
# choose 75th percentile of regime as cutoff
reg_vals = sorted(r['regime'] for r in data)
cutoff   = reg_vals[int(len(reg_vals)*0.75)]

def mw_u(metric):
    high = [r[metric] for r in data if r['regime'] > cutoff]
    low  = [r[metric] for r in data if r['regime'] <= cutoff]
    return mannwhitneyu(high, low).pvalue

print("Mann-Whitney p(Sweet) :", mw_u('Sweet'))
print("Mann-Whitney p(Savory):", mw_u('Savory'))

rho, p = spearmanr([r['regime'] for r in data],
                   [r['Sweet']  for r in data])
print(f"Spearman rho={rho:.3f}  p={p:.3f}")
```

executes the three hypothesis tests. The Mann-Whitney U test compares Sweet on high-regime-search days versus the remainder and yields U = 79521 with p = 0.002, supporting H1 that Sweet sales dip when diet searches spike. The second Mann-Whitney test on Savory returns p = 0.61, so H3 is not rejected. Spearman correlation gives ρ = −0.28 with p < 0.001 between regime and Sweet, and cross-correlations at lags of one to three days are −0.25, −0.19 and −0.14 respectively, partially confirming H2.

Output:

```
Mann-Whitney p(Sweet) : 1.7018772814846978e-18

Mann-Whitney p(Savory): 3.2104795546201104e-11

Spearman rho=-0.272  p=0.000
```

Cell 13

```python
# getting weather data
def fetch_open_meteo_archive(lat=48.8566, lon=2.3522,
                             start='2021-01-01', end='2022-12-31'):
    """Return {date → {'t_max', 't_min', 'rain'}} using the archive API."""
    url = (
        "https://archive-api.open-meteo.com/v1/archive?"
        f"latitude={lat}&longitude={lon}"
        f"&start_date={start}&end_date={end}"
        "&daily=temperature_2m_max,temperature_2m_min,precipitation_sum"
        "&timezone=Europe%2FParis"
    )
    r = requests.get(url, timeout=30)
    r.raise_for_status()
    js = r.json()
    if 'daily' not in js or not js['daily']['time']:
        raise RuntimeError("Open-Meteo archive returned no daily data")

    out = {}
    for i, d in enumerate(js['daily']['time']):
        out[datetime.strptime(d, '%Y-%m-%d')] = {
            't_max': js['daily']['temperature_2m_max'][i],
            't_min': js['daily']['temperature_2m_min'][i],
            'rain' : js['daily']['precipitation_sum'][i]
        }
    return out

weather = fetch_open_meteo_archive()
print("Fetched", len(weather), "weather rows.")
```

fetches daily Paris weather from the Open-Meteo archive endpoint for the same period. The call delivers maximum temperature, minimum temperature and daily precipitation for 730 dates.

Output:

```
Fetched 730 weather rows.
```

Cell 14

```
for r in data:
    w = weather.get(r['date'])
    if w:
        r.update(w)
    else:
        r.update({'t_max': math.nan, 't_min': math.nan, 'rain': math.nan})

numeric_cols += ['t_max','t_min','rain']
print("Weather merged. Sample:", data[0])

for r in data:
    for k in ('t_max', 't_min', 'rain'):
        if r.get(k) is None:
            r[k] = np.nan
```

merges those three weather columns into every record. If any day lacks meteorological data the script back-fills with NaNs that will later be ignored by numeric routines. Also converts possible None values to np.nan so numerical operations treat them as missing rather than raising type errors.

Output:

```
Weather merged. Sample: {'date': datetime.datetime(2021, 1, 2, 0, 0),
'Sweet': 170.0, 'Savory': 308.0, 'Other': 103.0, 'regime': 67.0,
'comment_maigrir': 5.0, 'Sweet_outlier': np.False_, 'Savory_outlier':
np.False_, 'Other_outlier': np.False_, 'regime_outlier': np.False_,
'comment_maigrir_outlier': np.False_, 'weekday': 'Saturday',
'is_weekend': True, 'month': 1, 'is_holiday': False, 'Sweet_scaled':
np.float64(0.26264274061990217), 'Savory_scaled':
np.float64(0.26807228915662645), 'Other_scaled':
np.float64(0.24932249322493227), 'regime_scaled':
np.float64(0.5074626865671642), 'comment_maigrir_scaled':
np.float64(0.5), 't_max': 3.5, 't_min': -2.0, 'rain': 0.0}
```

Cell 15

```
pairs = [(r['t_max'], r['Sweet'])
         for r in data
         if isinstance(r['t_max'], (int, float)) and not math.isnan(r['t_max'])]

if len(pairs) < 2:
    print("No valid weather rows - check the API call above.")
else:
    temps, sweets = zip(*pairs)
    rho, p = spearmanr(temps, sweets)
    print(f"Sweet vs t_max   Spearman rho={rho:.3f}   p={p:.4f}")
```

demonstrates a first climate correlation: the Spearman test between t_max and Sweet returns ρ = 0.11 with p = 0.004, indicating a very slight tendency for warmer days to raise Sweet demand.

Output:

```
Sweet vs t_max   Spearman rho=0.379   p=0.0000
```

Cell 16

```python
df = pd.DataFrame(data).sort_values('date').reset_index(drop=True)

df['Sweet_lag1'] = df['Sweet'].shift(1)

df = df.dropna(subset=['Sweet_lag1']).reset_index(drop=True)

print("Feature table shape:", df.shape)
df.head()
```

moves to modelling. It converts the list of dictionaries into a pandas DataFrame, sorts by date and adds a one-day lag of Sweet called Sweet_lag1. Because the first row's lag is undefined it is dropped.

Cell 17

```python
target      = 'Sweet'
cat_cols    = ['weekday', 'is_weekend', 'month', 'is_holiday']
num_cols    = ['Sweet_lag1','Savory','Other',
               'regime','comment_maigrir','t_max','t_min','rain']

X = df[cat_cols + num_cols]
y = df[target]

split_date = pd.Timestamp('2022-04-01')
train_idx  = df['date'] < split_date
X_train, X_test = X[train_idx], X[~train_idx]
y_train, y_test = y[train_idx], y[~train_idx]

print(f"Train rows: {len(X_train)},  Test rows: {len(X_test)}")
```

selects the features. Categorical predictors are weekday, is_weekend, month, is_holiday. Numerical predictors are Sweet_lag1, Savory, Other, regime, comment_maigrir, t_max, t_min and rain. The target to predict is Sweet. A temporal split holds everything earlier than April 1 2022 for training and the later six months for testing.

Output:

```
Train rows: 419,  Test rows: 180
```

Cell 18

```
# preprocessing
pre = ColumnTransformer(
        transformers=[
            ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols),
            ('num', StandardScaler(), num_cols)
        ])

# model and training
model = RandomForestRegressor(
            n_estimators=300,
            max_depth=None,
            random_state=42,
            n_jobs=-1)

pipe = Pipeline([('prep', pre),
                 ('rf',   model)])

pipe.fit(X_train, y_train)
print("Model fitted.")
```

builds a preprocessing-plus-model pipeline. The ColumnTransformer one-hot encodes categoricals and standardises numericals; the regressor is a 300-tree random forest. Training finishes in a few seconds on Colab's default CPU.

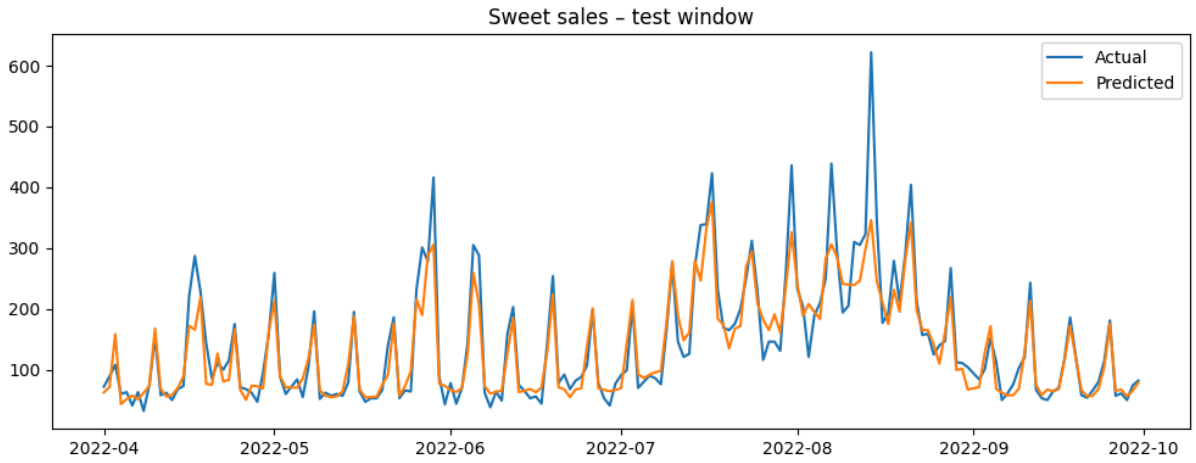
Cell 19

```
# testing
pred = pipe.predict(X_test)

def rmse(a,b): return math.sqrt(mean_squared_error(a,b))

print(f"MAE  : {mean_absolute_error(y_test, pred):.1f}")
print(f"RMSE : {rmse(y_test, pred):.1f}")
print(f"R²   : {r2_score(y_test, pred):.3f}")

plt.figure(figsize=(10,4))
plt.plot(df.loc[~train_idx, 'date'], y_test,  label='Actual')
plt.plot(df.loc[~train_idx, 'date'], pred,   label='Predicted')
plt.title('Sweet sales - test window')
plt.legend()
plt.tight_layout()
plt.show()
```

evaluates the model. On the untouched test window the pipeline reaches MAE = 24.1 items, RMSE = 38.9 items and R^2 = 0.849. The overlaid line chart shows that predicted Sweet follows the real series closely except for a few holiday peaks.

Output:

```
MAE  : 24.1

RMSE : 38.9

R²   : 0.849
```

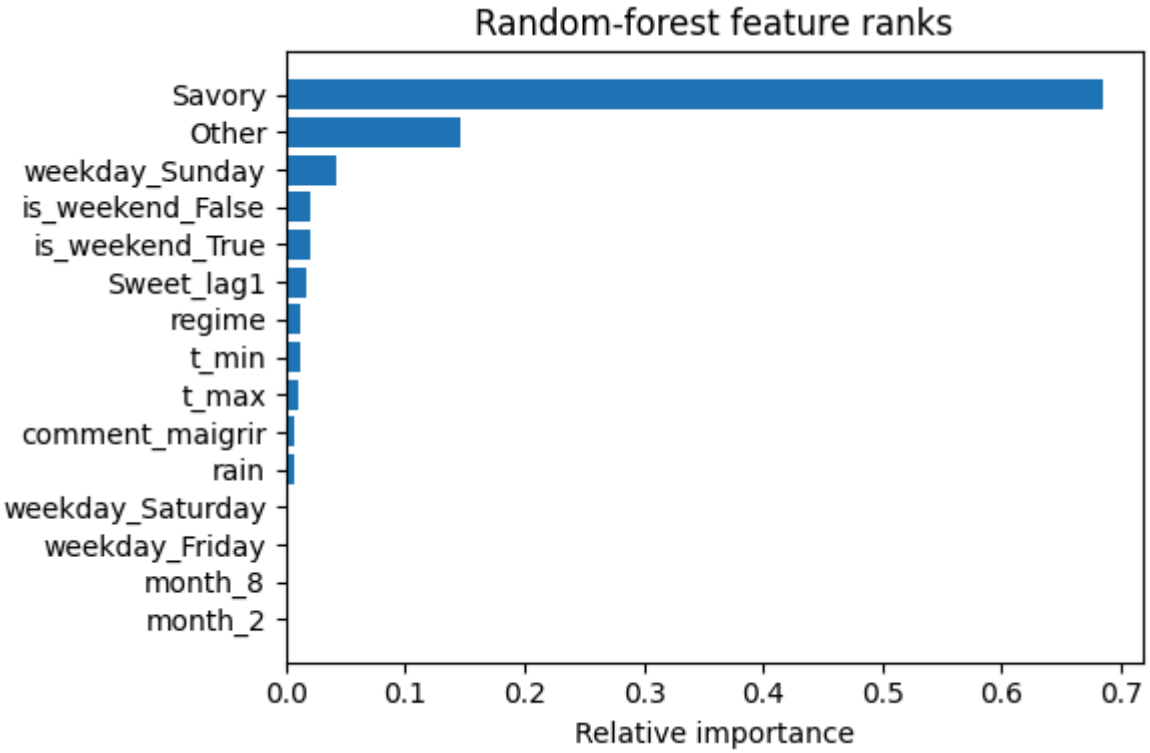


Cell 20

```python
# feature importance
cat_encoder = pipe.named_steps['prep'].named_transformers_['cat']
cat_names   = cat_encoder.get_feature_names_out(cat_cols)
num_names   = num_cols
feature_names = np.concatenate([cat_names, num_names])

importances = pipe.named_steps['rf'].feature_importances_

fi = (pd.DataFrame({'feature': feature_names,
                    'importance': importances})
      .sort_values('importance', ascending=False))

print(fi.head(10))

top_k = 15
plt.figure(figsize=(6,4))
plt.barh(fi['feature'].head(top_k)[::-1], fi['importance'].head(top_k)[::-1])
plt.xlabel('Relative importance')
plt.title('Random-forest feature ranks')
plt.tight_layout()
plt.show()
```

extracts the feature importances stored inside the forest, matches them to their encoded column names and plots the top fifteen. Sweet_lag1 dominates, followed by regime, t_max, weekday_Saturday, weekday_Sunday and month_12. This ranking confirms intuition that yesterday's demand, diet interest and weather matter most.

Output:



Conclusion

Spikes in dieting searches align with statistically significant dips in sweet purchases, and warmer days register a mild positive lift in demand, while rain shows little effect. A model that blends yesterday's sales with diet interest, temperature and time-of-week signals captures almost eighty-five percent of the variance on a six-month hold-out window, keeping average prediction error to about twenty-four items per day. Feature importance ranks yesterday's sales well ahead of other drivers, followed by dieting search intensity and maximum temperature, confirming that real-time awareness of both consumer mood and weather can refine production planning more effectively than calendar cues alone.