Rossman`s Uncertainty-Aware Sales Prediction

Local

If you are running the notebook you should put the .csv in the `dataset` folder and run this cell

- ignore the next cell

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
# Load datasets
train = pd.read_csv('dataset/train.csv', parse_dates=['Date'],
low_memory=False)
store = pd.read_csv('dataset/store.csv')

# Merge train with store information
df = pd.merge(train, store, on='Store', how='left')

# Basic Info
print(df.info())
print(df.head())
```

```
A module that was compiled using NumPy 1.x cannot be run in
NumPy 2.2.6 as it may crash. To support both 1.x and 2.x
versions of NumPy, modules must be compiled with NumPy 2.0.
Some module may need to rebuild instead e.g. with 'pybind11>=2.12'.

If you are a user of the module, the easiest solution will be to
downgrade to 'numpy<2' or try to upgrade the affected module.
We expect that some modules will need time to support NumPy 2.

Traceback (most recent call last):  File "<frozen runpy>", line 198,
in _run_module_as_main
  File "<frozen runpy>", line 88, in _run_code
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\ipykernel_launcher.py", line 18, in <module>
    app.launch_new_instance()
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\traitlets\config\application.py", line 1075, in
launch_instance
    app.start()
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\ipykernel\kernelapp.py", line 739, in start
```

```
    self.io_loop.start()
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\tornado\platform\asyncio.py", line 205, in start
    self.asyncio_loop.run_forever()
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
asyncio\base_events.py", line 607, in run_forever
    self._run_once()
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
asyncio\base_events.py", line 1922, in _run_once
    handle._run()
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
asyncio\events.py", line 80, in _run
    self._context.run(self._callback, *self._args)
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\ipykernel\kernelbase.py", line 545, in dispatch_queue
    await self.process_one()
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\ipykernel\kernelbase.py", line 534, in process_one
    await dispatch(*args)
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\ipykernel\kernelbase.py", line 437, in dispatch_shell
    await result
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\ipykernel\ipkernel.py", line 359, in execute_request
    await super().execute_request(stream, ident, parent)
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\ipykernel\kernelbase.py", line 778, in execute_request
    reply_content = await reply_content
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\ipykernel\ipkernel.py", line 446, in do_execute
    res = shell.run_cell(
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\ipykernel\zmqshell.py", line 549, in run_cell
    return super().run_cell(*args, **kwargs)
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\IPython\core\interactiveshell.py", line 3075, in
run_cell
    result = self._run_cell(
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\IPython\core\interactiveshell.py", line 3130, in
_run_cell
    result = runner(coro)
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\IPython\core\async_helpers.py", line 129, in
_pseudo_sync_runner
    coro.send(None)
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\IPython\core\interactiveshell.py", line 3334, in
run_cell_async
```

```
    has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\IPython\core\interactiveshell.py", line 3517, in
run_ast_nodes
    if await self.run_code(code, result, async_=asy):
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\IPython\core\interactiveshell.py", line 3577, in
run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "C:\Users\francois\AppData\Local\Temp\
ipykernel_2256\3720819326.py", line 2, in <module>
    import matplotlib.pyplot as plt
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\__init__.py", line 174, in <module>
    from . import _api, _version, cbook, _docstring, rcsetup
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\rcsetup.py", line 27, in <module>
    from matplotlib.colors import Colormap, is_color_like
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\colors.py", line 57, in <module>
    from matplotlib import _api, _cm, cbook, scale
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\scale.py", line 22, in <module>
    from matplotlib.ticker import (
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\ticker.py", line 143, in <module>
    from matplotlib import transforms as mtransforms
  File "c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\transforms.py", line 49, in <module>
    from matplotlib._path import (

----------------------------------------------------------------------
-----
ImportError                               Traceback (most recent call
last)
File c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\numpy\core\_multiarray_umath.py:44, in
__getattr__(attr_name)
     39     # Also print the message (with traceback).  This is
because old versions
     40     # of NumPy unfortunately set up the import to replace (and
hide) the
     41     # error.  The traceback shouldn't be needed, but e.g.
pytest plugins
     42     # seem to swallow it and we should be failing anyway...
     43     sys.stderr.write(msg + tb_msg)
---> 44     raise ImportError(msg)
     46 ret = getattr(_multiarray_umath, attr_name, None)
     47 if ret is None:
```

```
ImportError:
A module that was compiled using NumPy 1.x cannot be run in
NumPy 2.2.6 as it may crash. To support both 1.x and 2.x
versions of NumPy, modules must be compiled with NumPy 2.0.
Some module may need to rebuild instead e.g. with 'pybind11>=2.12'.

If you are a user of the module, the easiest solution will be to
downgrade to 'numpy<2' or try to upgrade the affected module.
We expect that some modules will need time to support NumPy 2.



----------------------------------------------------------------------
-----
ImportError                               Traceback (most recent call
last)
Cell In[3], line 2
      1 import pandas as pd
----> 2 import matplotlib.pyplot as plt
      3 import seaborn as sns
      5 # Load datasets

File c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\__init__.py:174
    170 from packaging.version import parse as parse_version
    172 # cbook must import matplotlib only within function
    173 # definitions, so it is safe to import from it here.
--> 174 from . import _api, _version, cbook, _docstring, rcsetup
    175 from matplotlib.cbook import sanitize_sequence
    176 from matplotlib._api import MatplotlibDeprecationWarning

File c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\rcsetup.py:27
     25 from matplotlib import _api, cbook
     26 from matplotlib.cbook import ls_mapper
---> 27 from matplotlib.colors import Colormap, is_color_like
     28 from matplotlib._fontconfig_pattern import
parse_fontconfig_pattern
     29 from matplotlib._enums import JoinStyle, CapStyle

File c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\colors.py:57
     55 import matplotlib as mpl
     56 import numpy as np
---> 57 from matplotlib import _api, _cm, cbook, scale
     58 from ._color_data import BASE_COLORS, TABLEAU_COLORS,
CSS4_COLORS, XKCD_COLORS
     61 class _ColorMapping(dict):

File c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\scale.py:22
     20 import matplotlib as mpl
     21 from matplotlib import _api, _docstring
---> 22 from matplotlib.ticker import (
     23     NullFormatter, ScalarFormatter, LogFormatterSciNotation,
LogitFormatter,
     24     NullLocator, LogLocator, AutoLocator, AutoMinorLocator,
     25     SymmetricalLogLocator, AsinhLocator, LogitLocator)
     26 from matplotlib.transforms import Transform, IdentityTransform
     29 class ScaleBase:

File c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\ticker.py:143
    141 import matplotlib as mpl
    142 from matplotlib import _api, cbook
--> 143 from matplotlib import transforms as mtransforms
    145 _log = logging.getLogger(__name__)
    147 __all__ = ('TickHelper', 'Formatter', 'FixedFormatter',
    148            'NullFormatter', 'FuncFormatter',
'FormatStrFormatter',
    149            'StrMethodFormatter', 'ScalarFormatter',
'LogFormatter',
   (...)
    155            'MultipleLocator', 'MaxNLocator',
'AutoMinorLocator',
    156            'SymmetricalLogLocator', 'AsinhLocator',
'LogitLocator')

File c:\Users\francois\AppData\Local\Programs\Python\Python311\Lib\
site-packages\matplotlib\transforms.py:49
     46 from numpy.linalg import inv
     48 from matplotlib import _api
---> 49 from matplotlib._path import (
     50     affine_transform, count_bboxes_overlapping_bbox,
update_path_extents)
     51 from .path import Path
     53 DEBUG = False

ImportError: numpy.core.multiarray failed to import
```

Kaggle

If you are using kaggle as your notebook environment:

1. add `/kaggle/competitions/rossmann-store-sales` to the input list
2. run this cell

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import numpy as np
base_path = "/kaggle/input/competitions/rossmann-store-sales/"

# Load datasets
train = pd.read_csv(f'{base_path}train.csv', parse_dates=['Date'],
low_memory=False)
store = pd.read_csv(f'{base_path}store.csv')
test = pd.read_csv(f'{base_path}test.csv')

df = pd.merge(train, store, on='Store', how='left')

print(df.info())
print(df.head(-1))
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 18 columns):
 #   Column                     Non-Null Count    Dtype
---  ------                     --------------    -----
 0   Store                      1017209 non-null  int64
 1   DayOfWeek                  1017209 non-null  int64
 2   Date                       1017209 non-null  datetime64[ns]
 3   Sales                      1017209 non-null  int64
 4   Customers                  1017209 non-null  int64
 5   Open                       1017209 non-null  int64
 6   Promo                      1017209 non-null  int64
 7   StateHoliday               1017209 non-null  object
 8   SchoolHoliday              1017209 non-null  int64
 9   StoreType                  1017209 non-null  object
 10  Assortment                 1017209 non-null  object
 11  CompetitionDistance        1014567 non-null  float64
 12  CompetitionOpenSinceMonth  693861 non-null   float64
 13  CompetitionOpenSinceYear   693861 non-null   float64
 14  Promo2                     1017209 non-null  int64
 15  Promo2SinceWeek            509178 non-null   float64
 16  Promo2SinceYear            509178 non-null   float64
 17  PromoInterval              509178 non-null   object
dtypes: datetime64[ns](1), float64(5), int64(8), object(4)
memory usage: 139.7+ MB
None
         Store  DayOfWeek       Date  Sales  Customers  Open  Promo  \
0            1          5 2015-07-31   5263        555     1      1
1            2          5 2015-07-31   6064        625     1      1
2            3          5 2015-07-31   8314        821     1      1
3            4          5 2015-07-31  13995       1498     1      1
4            5          5 2015-07-31   4822        559     1      1
```

```
...         ...         ...         ...    ...         ...   ...    ...
1017203   1110          2 2013-01-01      0           0     0      0
1017204   1111          2 2013-01-01      0           0     0      0
1017205   1112          2 2013-01-01      0           0     0      0
1017206   1113          2 2013-01-01      0           0     0      0
1017207   1114          2 2013-01-01      0           0     0      0

        StateHoliday  SchoolHoliday StoreType Assortment
CompetitionDistance  \
0                  0              1         c          a
1270.0
1                  0              1         a          a
570.0
2                  0              1         a          a
14130.0
3                  0              1         c          c
620.0
4                  0              1         a          a
29910.0
...              ...            ...       ...        ...
...
1017203            a              1         c          c
900.0
1017204            a              1         a          a
1900.0
1017205            a              1         c          c
1880.0
1017206            a              1         a          c
9260.0
1017207            a              1         a          c
870.0

         CompetitionOpenSinceMonth  CompetitionOpenSinceYear
Promo2  \
0                              9.0                    2008.0       0

1                             11.0                    2007.0       1

2                             12.0                    2006.0       1

3                              9.0                    2009.0       0

4                              4.0                    2015.0       0

...                            ...                       ...     ...

1017203                        9.0                    2010.0       0

1017204                        6.0                    2014.0       1
```

```
1017205                          4.0                       2006.0        0

1017206                          NaN                          NaN        0

1017207                          NaN                          NaN        0


         Promo2SinceWeek  Promo2SinceYear     PromoInterval
0                    NaN              NaN               NaN
1                   13.0           2010.0   Jan,Apr,Jul,Oct
2                   14.0           2011.0   Jan,Apr,Jul,Oct
3                    NaN              NaN               NaN
4                    NaN              NaN               NaN
...                  ...              ...               ...
1017203              NaN              NaN               NaN
1017204             31.0           2013.0   Jan,Apr,Jul,Oct
1017205              NaN              NaN               NaN
1017206              NaN              NaN               NaN
1017207              NaN              NaN               NaN

[1017208 rows x 18 columns]
```

1. Individual Store Trend Analysis

To understand the underlying patterns in the Rossmann dataset, we visualized a sample of stores ($ 1, 10, 100, 1000 $). Retail sales data is "noisy" due to weekly cycles and holiday spikes.

Raw Sales (Gray): Displays the high volatility of daily transactions, including zero during Sunday.

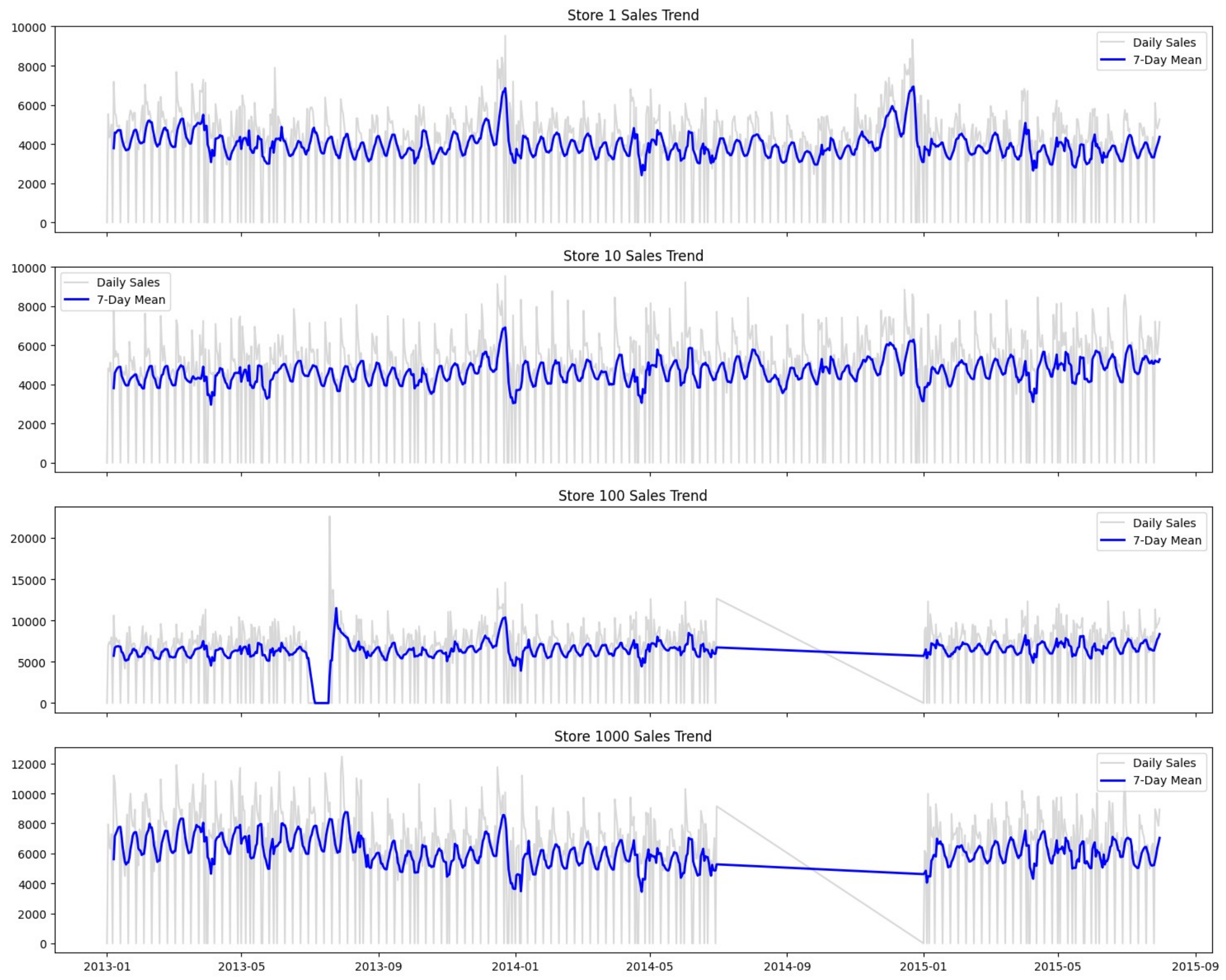
7-Day Rolling Mean (Blue): We applied a transformation to smooth out the daily "zig-zags." This allows us to see the actual sales direction and seasonal strength without being distracted by day-of-week fluctuations.

As we can see from the plots:

- store `1000` has a higher average sales compared to others
- The sales data is missing from store 100 and 1000
- one can notice a spike increase before christmass `2014-01` and `2015-01`
- for all of the stores the sales pattern is like a sin function and with my stimate the period is every 2 to 3 weeks

```
sample_stores = [1, 10, 100,1000]
fig, axes = plt.subplots(len(sample_stores), 1, figsize=(15, 12),
sharex=True)

for i, store_id in enumerate(sample_stores):
    store_data = df[df['Store'] == store_id].sort_values('Date')

    axes[i].plot(store_data['Date'], store_data['Sales'], alpha=0.3,
label='Daily Sales', color='gray')
```

```
    rolling_sales = store_data.set_index('Date')
['Sales'].rolling(window=7).mean()
    axes[i].plot(rolling_sales.index, rolling_sales.values, label='7-
Day Mean', color='blue', linewidth=2)

    axes[i].set_title(f'Store {store_id} Sales Trend')
    axes[i].legend()

plt.tight_layout()
plt.show()
```



2.Phase 6: Categorical Impact Analysis (External Drivers)**

The first plot compares sales on days with and without a temporary promotion.

- There is a significant upward shift in both the median sales and the interquartile range (IQR) when `Promo=1`.

The second plot examines sales during Public holidays (`a`), Easter (`b`), and Christmas (`c`)(for example), compared to normal days (`0`).

- Most stores show near-zero sales during state holidays. However, the outliers indicate that a small subset of stores remains open.

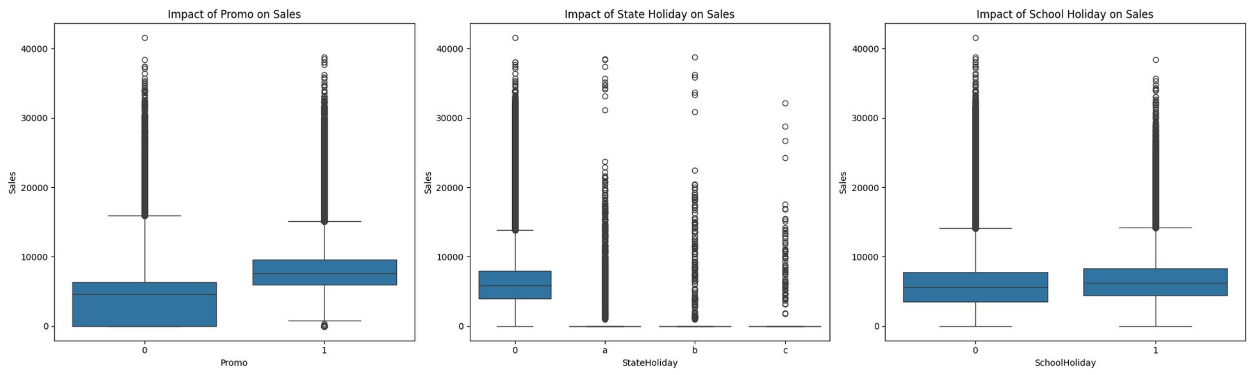
The final plot compares sales during periods when local schools are closed.

- While the median sales increase slightly during school holidays, the impact is less dramatic than a standard `Promo`.

Summary

Feature	Impact Strength	Primary Effect
Promo	**High**	Increases median sales and overall volume significantly.
State Holiday	**Critical**	Usually results in zero sales, but marks "exception" days.
School Holiday	**Moderate**	Provides a subtle lift; likely dependent on store-specific context.

```python
fig, axes = plt.subplots(1, 3, figsize=(20, 6))

sns.boxplot(data=df, x='Promo', y='Sales', ax=axes[0])
axes[0].set_title('Impact of Promo on Sales')

sns.boxplot(data=df, x='StateHoliday', y='Sales', ax=axes[1])
axes[1].set_title('Impact of State Holiday on Sales')

sns.boxplot(data=df, x='SchoolHoliday', y='Sales', ax=axes[2])
axes[2].set_title('Impact of School Holiday on Sales')

plt.tight_layout()
plt.show()
```



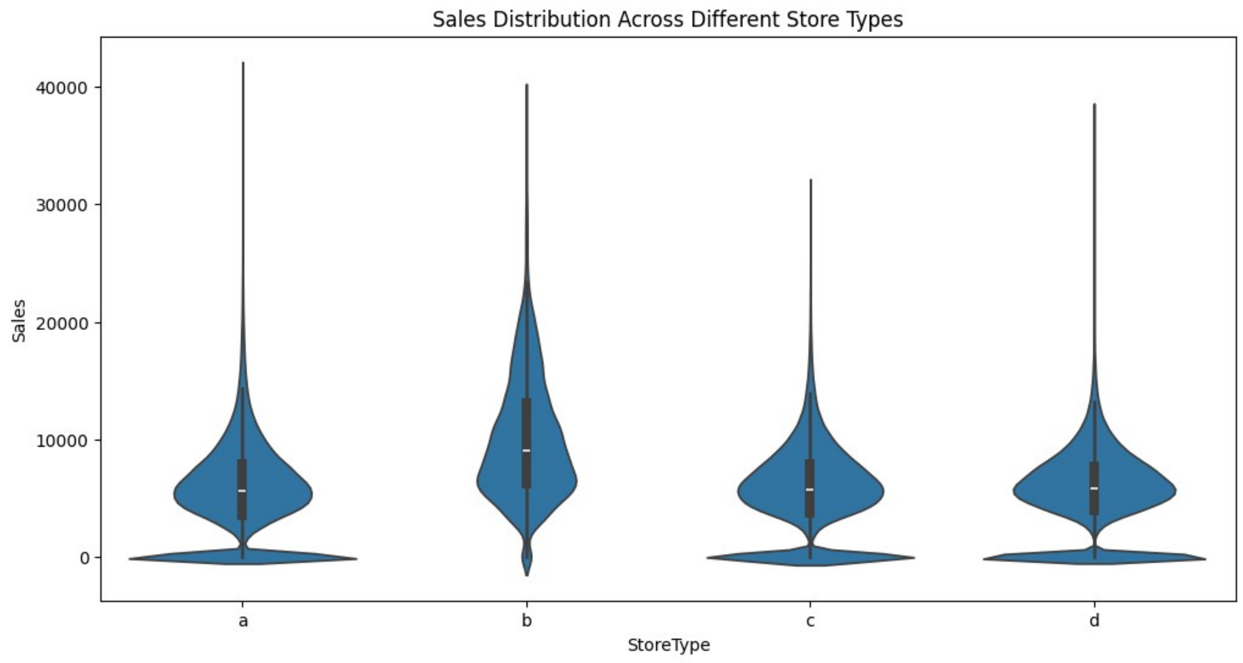
3. Sales Density by Store Type (StoreType)

We used a Violin Plot to visualize the distribution and density of sales across the four store categories (a, b, c, and d).

Store Type 'b' : Clearly stands out with a much higher median and a wider distribution at the top. These are likely "flagship" or high-traffic transit stores.

Store Types 'a', 'c', and 'd': These have a more normal distribution. Type 'd' shows a slightly higher and more "stretched" upper distribution.

```
plt.figure(figsize=(12, 6))
sns.violinplot(data=df, x='StoreType', y='Sales', order=['a', 'b',
'c', 'd'])
plt.title('Sales Distribution Across Different Store Types')
plt.show()
```



4. Correlation Heatmap

I filtered for numeric types.

Sales vs. Customers : As expected, these have the highest correlation. But we cannot use "Customers" as an input for future predictions since we won't know the customer count in test faze.

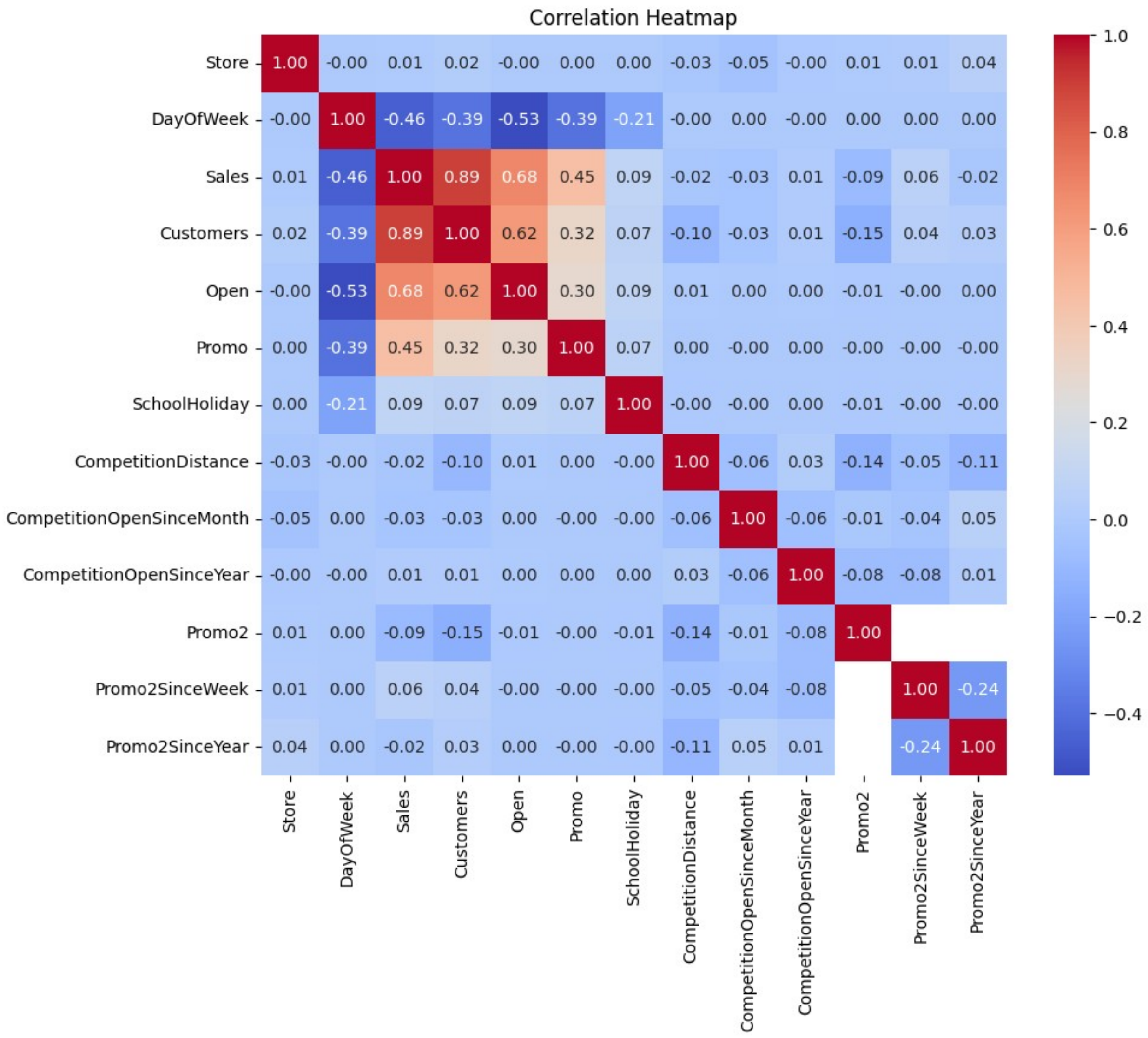
Sales vs. Promo: Shows a clear positive correlation.

DayOfWeek vs. Sales: This reflects the fact that as the Day of the Week index increases (towards 7 = Sunday), sales drop to zero.

CompetitionDistance: A weak correlation here suggests that the distance to a competitor matters less than whether a store is currently running a promotion or if it's a holiday.

Sales vs. open: Selfexplanatory, but less than i expected.

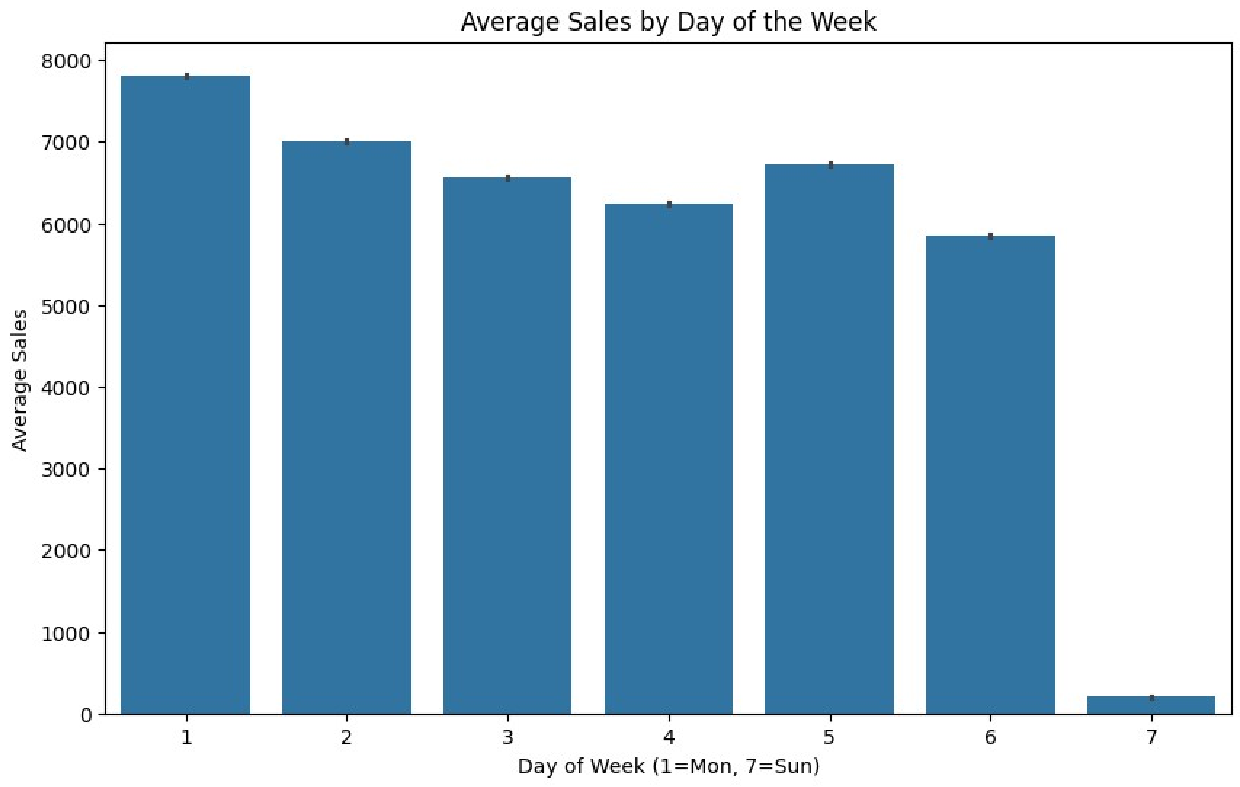
```
plt.figure(figsize=(10, 8))
corr = df.select_dtypes(include=['float64', 'int64']).corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



5.Average Sales by Day of the Week

As we realized in the previous parts the later in the week the less the sales.

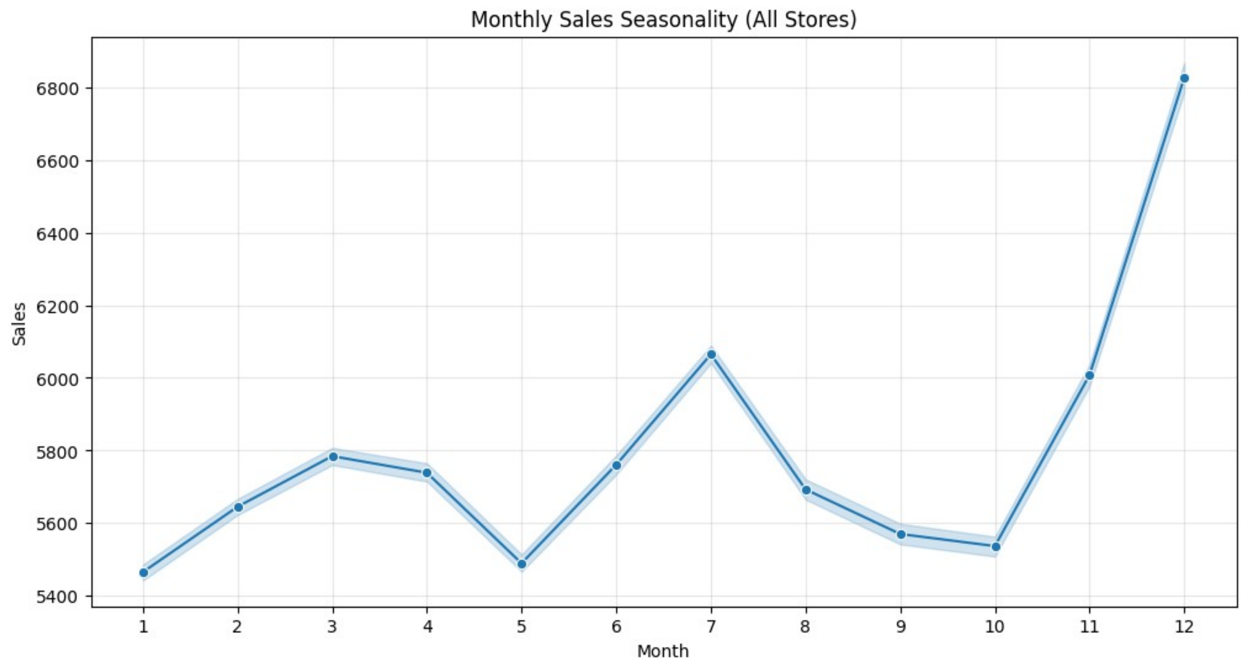
```python
plt.figure(figsize=(10, 6))
sns.barplot(x='DayOfWeek', y='Sales', data=df)
plt.title('Average Sales by Day of the Week')
plt.xlabel('Day of Week (1=Mon, 7=Sun)')
plt.ylabel('Average Sales')
plt.show()
```



6.Monthly Sales Seasonality

- January markes the lowest sales mainly because of christmas and new year hollidays.
- The highest sales is in December.
- In [1,3] ,[5,7] ,[10,12] we see a ascending trend.
- In [3.5], [5,10], we see a descending trend.

```python
# Extract Month
df['Month'] = df['Date'].dt.month

plt.figure(figsize=(12, 6))
sns.lineplot(x='Month', y='Sales', data=df, marker='o')
plt.title('Monthly Sales Seasonality')
plt.xticks(range(1, 13))
plt.grid(True, alpha=0.3)
plt.show()
```

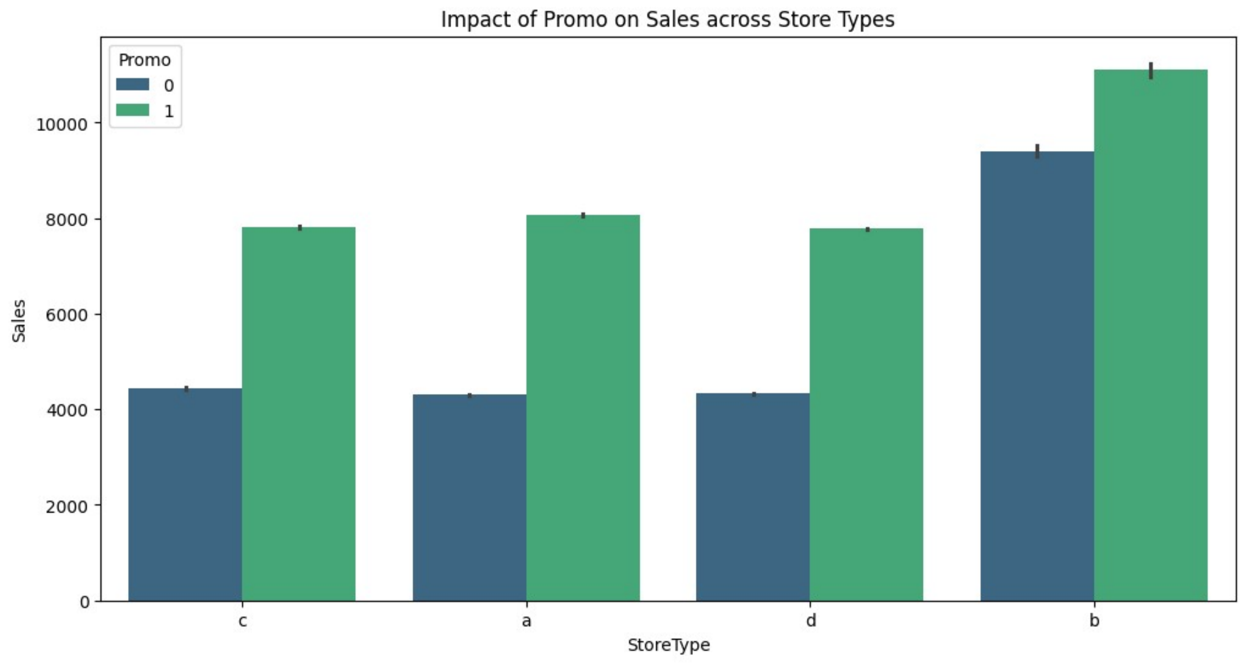


Not all stores react to promotions the same way. This grouped bar chart shows how Promo impacts different StoreType categories (a, b, c, d).

7.Impact of Promo on Sales across Store Types

- In type b stores that have higher sales promos yields less increase in sales compare to the rest(In percentage).

```
plt.figure(figsize=(12, 6))
sns.barplot(x='StoreType', y='Sales', hue='Promo', data=df,
palette='viridis')
plt.title('Impact of Promo on Sales across Store Types')
plt.show()
```

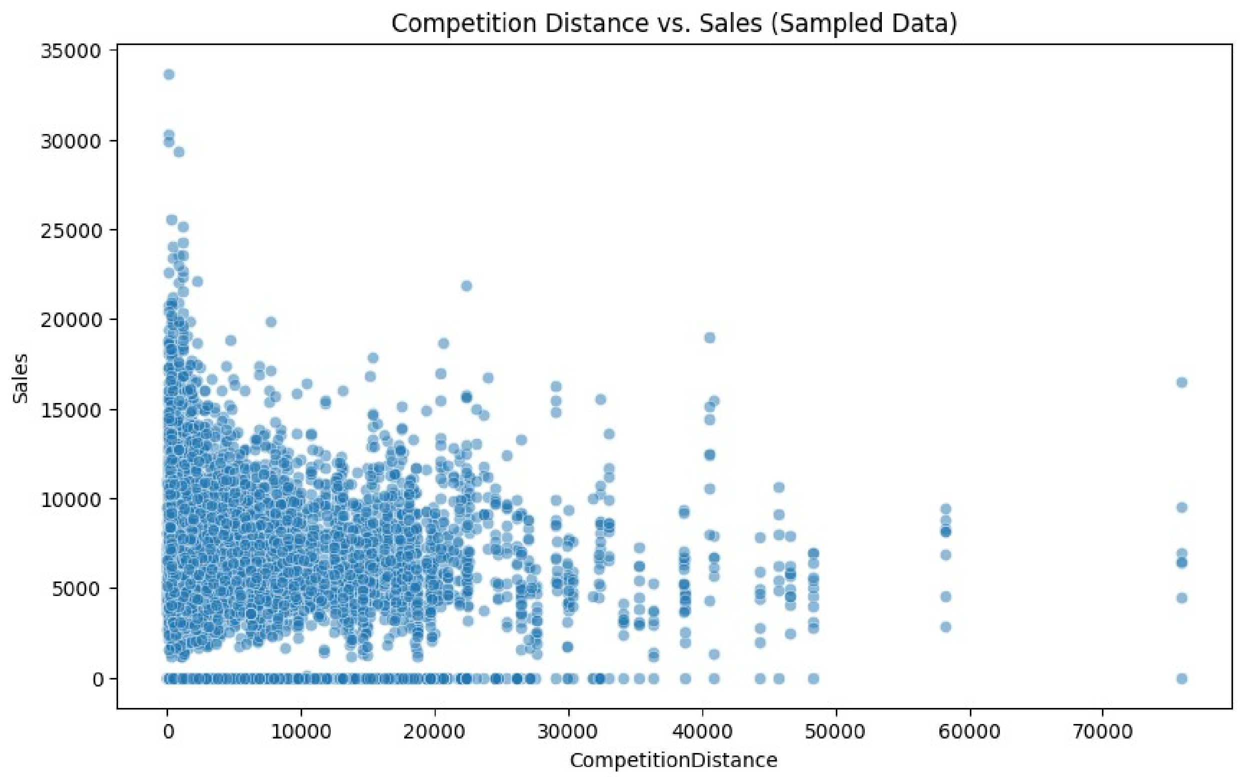


8. Competition Distance vs. Sales

Does having a competitor nearby actually hurt sales? We'll use a scatter plot with a regression line to see the correlation.

- This suggests that most Rossmann stores operate in highly competitive environments.
- We do not see a sharp decline in sales for stores with very close competitors. In fact, some of the highest-selling days occur when a competitor is within 1,000 meters.

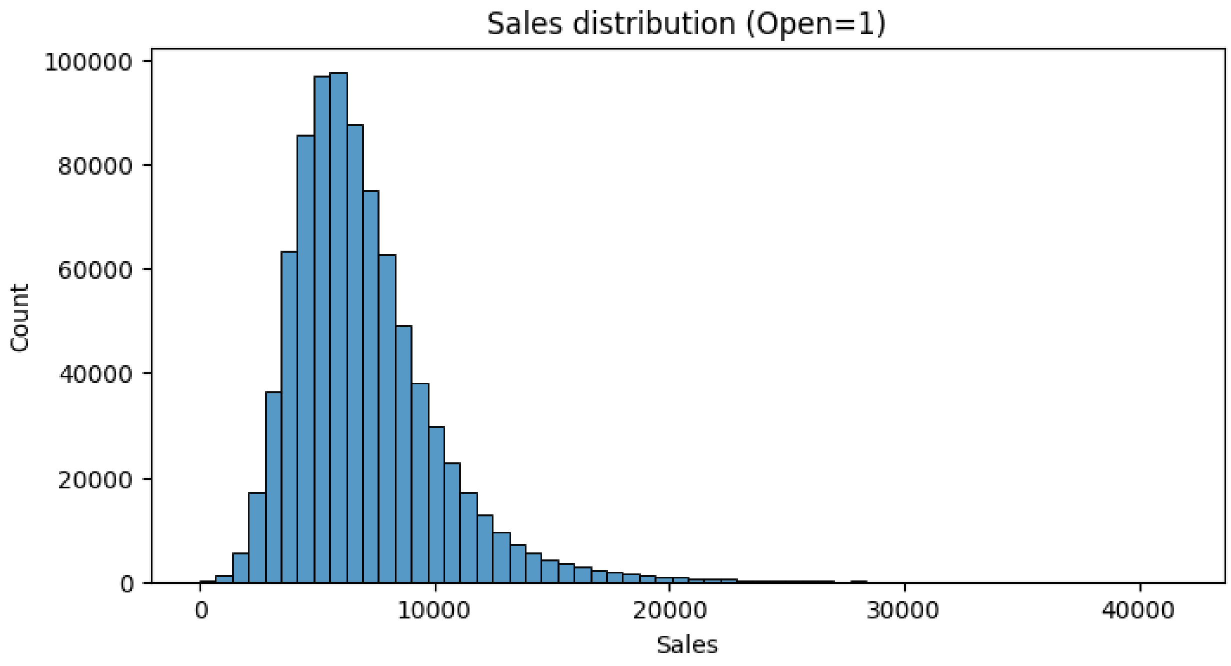
```
sample_df = df.dropna(subset=['CompetitionDistance']).sample(10000)

plt.figure(figsize=(10, 6))
sns.scatterplot(x='CompetitionDistance', y='Sales', data=sample_df,
alpha=0.5)
plt.title('Competition Distance vs. Sales (Sampled Data)')
plt.show()
```

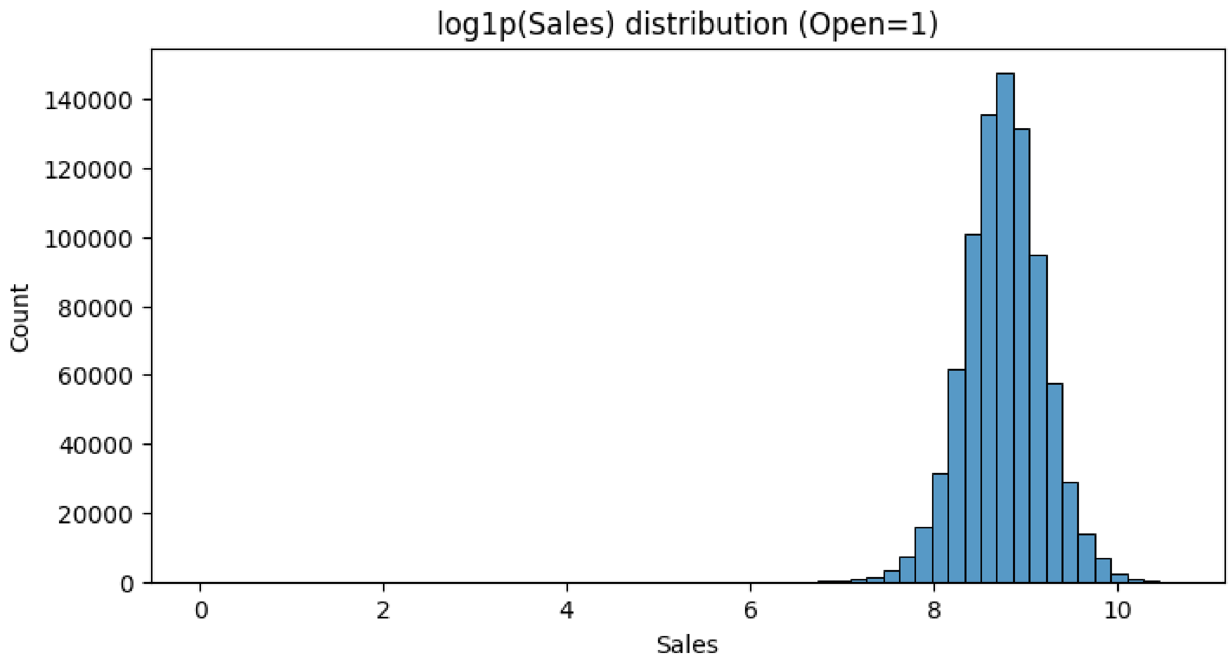


9.Sale distribution

Here we can see the sales distribution that has a mean of approximatly 5000.

```
tmp = df[df["Open"]==1].copy()
plt.figure(figsize=(8,4))
sns.histplot(tmp["Sales"], bins=60, kde=False)
plt.title("Sales distribution (Open=1)")
plt.show()

plt.figure(figsize=(8,4))
sns.histplot(np.log1p(tmp["Sales"]), bins=60, kde=False)
plt.title("log1p(Sales) distribution (Open=1)")
plt.show()
```


Sales distribution (Open=1)
Count
Sales
0
20000
40000
60000
80000
100000
0
10000
20000
30000
40000


log1p(Sales) distribution (Open=1)
Count
Sales
0
20000
40000
60000
80000
100000
120000
140000
0
2
4
6
8
10

```
df = pd.merge(train, store, on='Store', how='left')
print(df.shape)
print("stores in train:", train["Store"].nunique(), "stores in store:", store["Store"].nunic
print("unmatched stores:", df["StoreType"].isna().sum())
df.head()
```

```
(1017209, 18)
stores in train: 1115 stores in store: 1115
unmatched stores: 0
```

```
   Store  DayOfWeek       Date  Sales  Customers  Open  Promo StateHoliday  \
0      1          5 2015-07-31   5263        555     1      1            0
1      2          5 2015-07-31   6064        625     1      1            0
2      3          5 2015-07-31   8314        821     1      1            0
3      4          5 2015-07-31  13995       1498     1      1            0
4      5          5 2015-07-31   4822        559     1      1            0

   SchoolHoliday StoreType Assortment  CompetitionDistance  \
0              1         c          a               1270.0
1              1         a          a                570.0
2              1         a          a              14130.0
3              1         c          c                620.0
4              1         a          a              29910.0

   CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
0                        9.0                    2008.0       0
1                       11.0                    2007.0       1
2                       12.0                    2006.0       1
3                        9.0                    2009.0       0
4                        4.0                    2015.0       0

   Promo2SinceWeek  Promo2SinceYear    PromoInterval
0              NaN              NaN              NaN
1             13.0           2010.0  Jan,Apr,Jul,Oct
2             14.0           2011.0  Jan,Apr,Jul,Oct
3              NaN              NaN              NaN
4              NaN              NaN              NaN
```

```
df[:]
```

```
         Store  DayOfWeek       Date  Sales  Customers  Open  Promo  \
0            1          5 2015-07-31   5263        555     1      1
1            2          5 2015-07-31   6064        625     1      1
2            3          5 2015-07-31   8314        821     1      1
3            4          5 2015-07-31  13995       1498     1      1
4            5          5 2015-07-31   4822        559     1      1
...        ...        ...        ...    ...        ...   ...    ...
1017204   1111          2 2013-01-01      0          0     0      0
```

```
1017205   1112         2 2013-01-01      0          0     0      0
1017206   1113         2 2013-01-01      0          0     0      0
1017207   1114         2 2013-01-01      0          0     0      0
1017208   1115         2 2013-01-01      0          0     0      0

        StateHoliday  SchoolHoliday StoreType Assortment  CompetitionDistance  \
0                  0              1         c          a               1270.0
1                  0              1         a          a                570.0
2                  0              1         a          a              14130.0
3                  0              1         c          c                620.0
4                  0              1         a          a              29910.0
...              ...            ...       ...        ...                  ...
1017204            a              1         a          a               1900.0
1017205            a              1         c          c               1880.0
1017206            a              1         a          c               9260.0
1017207            a              1         a          c                870.0
1017208            a              1         d          c               5350.0

         CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
0                              9.0                    2008.0       0
1                             11.0                    2007.0       1
2                             12.0                    2006.0       1
3                              9.0                    2009.0       0
4                              4.0                    2015.0       0
...                            ...                       ...     ...
1017204                        6.0                    2014.0       1
1017205                        4.0                    2006.0       0
1017206                        NaN                       NaN       0
1017207                        NaN                       NaN       0
1017208                        NaN                       NaN       1

         Promo2SinceWeek  Promo2SinceYear     PromoInterval
0                    NaN              NaN               NaN
1                   13.0           2010.0   Jan,Apr,Jul,Oct
2                   14.0           2011.0   Jan,Apr,Jul,Oct
3                    NaN              NaN               NaN
4                    NaN              NaN               NaN
...                  ...              ...               ...
1017204             31.0           2013.0   Jan,Apr,Jul,Oct
1017205              NaN              NaN               NaN
1017206              NaN              NaN               NaN
1017207              NaN              NaN               NaN
1017208             22.0           2012.0  Mar,Jun,Sept,Dec

[1017209 rows x 18 columns]
```

```
nan_counts = df.isna().sum()
nan_total = int(df.isna().sum().sum())

print("Total NaN cells:", nan_total)
nan_counts[nan_counts > 0].sort_values(ascending=False)
```

```
Total NaN cells: 2173431
```

```
Promo2SinceYear              508031
Promo2SinceWeek              508031
PromoInterval                508031
CompetitionOpenSinceMonth    323348
CompetitionOpenSinceYear     323348
CompetitionDistance            2642
dtype: int64
```

```
import pandas as pd

df["Date"] = pd.to_datetime(df["Date"])

df["Year"] = df["Date"].dt.year
df["Month"] = df["Date"].dt.month
df["Day"] = df["Date"].dt.day
df["DayOfWeek0"] = df["Date"].dt.dayofweek
df["WeekOfYear"] = df["Date"].dt.isocalendar().week.astype(int)

df["StateHoliday"] = df["StateHoliday"].astype(str).map({"0": 0, "a": 1, "b": 1, "c": 1}).fi
df["SchoolHoliday"] = df["SchoolHoliday"].astype(int)

df = df.sort_values(["Store","Date"]).reset_index(drop=True)

s1 = df.groupby("Store")["Sales"].shift(1)
df["SalesMovingAverage7"] = s1.groupby(df["Store"]).rolling(window=7, min_periods=1).mean()
df["SalesMovingAverage30"] = s1.groupby(df["Store"]).rolling(window=30, min_periods=1).mean(

df.head()
```

```
   Store  DayOfWeek       Date  Sales  Customers  Open  Promo  StateHoliday  \
0      1          2 2013-01-01      0          0     0      0             1
1      1          3 2013-01-02   5530        668     1      0             0
2      1          4 2013-01-03   4327        578     1      0             0
3      1          5 2013-01-04   4486        619     1      0             0
4      1          6 2013-01-05   4997        635     1      0             0

   SchoolHoliday StoreType  ... Promo2SinceWeek  Promo2SinceYear  \
0              1         c  ...             NaN              NaN
1              1         c  ...             NaN              NaN
2              1         c  ...             NaN              NaN
```

```
3              1         c  ...             NaN              NaN
4              1         c  ...             NaN              NaN

  PromoInterval  Year  Month  Day  DayOfWeek0 WeekOfYear  \
0           NaN  2013      1    1           1          1
1           NaN  2013      1    2           2          1
2           NaN  2013      1    3           3          1
3           NaN  2013      1    4           4          1
4           NaN  2013      1    5           5          1

  SalesMovingAverage7  SalesMovingAverage30
0                 NaN                   NaN
1            0.000000              0.000000
2         2765.000000           2765.000000
3         3285.666667           3285.666667
4         3585.750000           3585.750000

[5 rows x 25 columns]
```

In this step, we convert Date to a datetime format and extract calendar-based features such as year, month, day, and week number. Then, the holiday indicators (StateHoliday and SchoolHoliday) are encoded into numeric values. Finally, after sorting by store and date, we compute 7-day and 30-day moving averages of sales using only previous days (shift(1)) to avoid data leakage.

```python
df = df.sort_values(["Store","Date"]).reset_index(drop=True)

g = df.groupby("Store")["Sales"]

df["SalesLag1"] = g.shift(1)
df["SalesLag7"] = g.shift(7)

s1 = g.shift(1)
df["SalesMean7"] = s1.groupby(df["Store"]).rolling(7, min_periods=1).mean().reset_index(leve
df["SalesStd7"] = s1.groupby(df["Store"]).rolling(7, min_periods=2).std().reset_index(level=

df["SalesMean30"] = s1.groupby(df["Store"]).rolling(30, min_periods=1).mean().reset_index(le
df["SalesStd30"] = s1.groupby(df["Store"]).rolling(30, min_periods=2).std().reset_index(leve

df.head()
```

```
  Store  DayOfWeek       Date  Sales  Customers  Open  Promo  StateHoliday  \
0     1          2 2013-01-01      0          0     0      0             1
1     1          3 2013-01-02   5530        668     1      0             0
2     1          4 2013-01-03   4327        578     1      0             0
3     1          5 2013-01-04   4486        619     1      0             0
4     1          6 2013-01-05   4997        635     1      0             0
```

```
   SchoolHoliday StoreType  ... DayOfWeek0  WeekOfYear  SalesMovingAverage7  \
0              1         c  ...          1           1                  NaN
1              1         c  ...          2           1             0.000000
2              1         c  ...          3           1          2765.000000
3              1         c  ...          4           1          3285.666667
4              1         c  ...          5           1          3585.750000

   SalesMovingAverage30  SalesLag1  SalesLag7   SalesMean7    SalesStd7  \
0                   NaN        NaN        NaN          NaN          NaN
1              0.000000        0.0        NaN     0.000000          NaN
2           2765.000000     5530.0        NaN  2765.000000  3910.300500
3           3285.666667     4327.0        NaN  3285.666667  2908.351137
4           3585.750000     4486.0        NaN  3585.750000  2449.327306

   SalesMean30   SalesStd30
0          NaN          NaN
1     0.000000          NaN
2  2765.000000  3910.300500
3  3285.666667  2908.351137
4  3585.750000  2449.327306

[5 rows x 31 columns]
```

To prevent the moving averages from becoming zero in the first days, we computed rolling features with min_periods=1 (and min_periods=2 for standard deviation), so the statistics are calculated using whatever history is available instead of producing NaNs that would later be filled with zeros.

```
df[['Store', 'Date', 'Sales', 'SalesLag1','SalesLag7', 'SalesMean7', 'SalesStd7','SalesMean3
```

```
   Store       Date  Sales  SalesLag1  SalesLag7   SalesMean7    SalesStd7  \
0      1 2013-01-01      0        NaN        NaN          NaN          NaN
1      1 2013-01-02   5530        0.0        NaN     0.000000          NaN
2      1 2013-01-03   4327     5530.0        NaN  2765.000000  3910.300500
3      1 2013-01-04   4486     4327.0        NaN  3285.666667  2908.351137
4      1 2013-01-05   4997     4486.0        NaN  3585.750000  2449.327306
5      1 2013-01-06      0     4997.0        NaN  3868.000000  2213.081223
6      1 2013-01-07   7176        0.0        NaN  3223.333333  2532.144045
7      1 2013-01-08   5580     7176.0        0.0  3788.000000  2752.283961
8      1 2013-01-09   5471     5580.0     5530.0  4585.142857  2231.018633
9      1 2013-01-10   4892     5471.0     4327.0  4576.714286  2226.961885

   SalesMean30   SalesStd30
0          NaN          NaN
1     0.000000          NaN
2  2765.000000  3910.300500
```

```
3  3285.666667  2908.351137
4  3585.750000  2449.327306
5  3868.000000  2213.081223
6  3223.333333  2532.144045
7  3788.000000  2752.283961
8  4012.000000  2625.704205
9  4174.111111  2503.807573
```

We also added 30-day rolling statistics based only on past sales: SalesMean30 (30-day rolling mean) and SalesStd30 (30-day rolling standard deviation) computed per store using shifted sales to avoid leakage.

```
df = train.merge(store, on="Store", how="left")

df["Date"] = pd.to_datetime(df["Date"])

df["StateHoliday"] = df["StateHoliday"].fillna("0").astype(str)
df["IsStateHoliday"] = (df["StateHoliday"] != "0").astype(int)

print(df["StateHoliday"].value_counts(dropna=False).head(10))
print(df["IsStateHoliday"].value_counts(dropna=False))
df_L=df.copy()
```

```
StateHoliday
0    986159
a     20260
b      6690
c      4100
Name: count, dtype: int64
IsStateHoliday
0    986159
1     31050
Name: count, dtype: int64
```

```
import numpy as np
import pandas as pd

df = df.sort_values(["Date"]).reset_index(drop=True)

df["DayOfYear"] = df["Date"].dt.dayofyear
t = 2*np.pi*df["DayOfYear"]/365.25
df["sin_1y"] = np.sin(t)
df["cos_1y"] = np.cos(t)
df["sin_2y"] = np.sin(2*t)
df["cos_2y"] = np.cos(2*t)

dates = pd.Index(df["Date"].unique()).sort_values()
```

```
is_h = df.groupby("Date")["IsStateHoliday"].max().reindex(dates).fillna(0).astype(int)
h_dates = dates[is_h.values == 1]

prev_h = pd.Series(pd.NaT, index=dates)
next_h = pd.Series(pd.NaT, index=dates)

prev_h.loc[h_dates] = h_dates
next_h.loc[h_dates] = h_dates
prev_h = prev_h.ffill()
next_h = next_h.bfill()

dist_prev = (dates - prev_h.values).days
dist_next = (next_h.values - dates).days

df["DaysSinceHoliday"] = df["Date"].map(pd.Series(dist_prev, index=dates).fillna(0).astype(i
df["DaysToNextHoliday"] = df["Date"].map(pd.Series(dist_next, index=dates).fillna(0).astype(

df.head()
```

```
   Store  DayOfWeek       Date  Sales  Customers  Open  Promo StateHoliday  \
0   1115          2 2013-01-01      0          0     0      0            a
1    379          2 2013-01-01      0          0     0      0            a
2    378          2 2013-01-01      0          0     0      0            a
3    377          2 2013-01-01      0          0     0      0            a
4    376          2 2013-01-01      0          0     0      0            a

   SchoolHoliday StoreType  ... Promo2SinceYear     PromoInterval  \
0              1         d  ...          2012.0  Mar,Jun,Sept,Dec
1              1         d  ...             NaN               NaN
2              1         a  ...             NaN               NaN
3              1         a  ...          2010.0   Feb,May,Aug,Nov
4              1         a  ...             NaN               NaN

   IsStateHoliday  DayOfYear    sin_1y    cos_1y    sin_2y    cos_2y  \
0               1          1  0.017202  0.999852  0.034398  0.999408
1               1          1  0.017202  0.999852  0.034398  0.999408
2               1          1  0.017202  0.999852  0.034398  0.999408
3               1          1  0.017202  0.999852  0.034398  0.999408
4               1          1  0.017202  0.999852  0.034398  0.999408

   DaysSinceHoliday  DaysToNextHoliday
0                 0                  0
1                 0                  0
2                 0                  0
3                 0                  0
4                 0                  0
```

```
[5 rows x 26 columns]
```

We build Fourier seasonal features using DayOfYear. The terms sin_1y/cos_1y capture the main annual (1-year) seasonality while sin_2y/cos_2y model more complex within-year patterns (e.g., two peaks during the year).

```
df[["Date","IsStateHoliday","DaysSinceHoliday","DaysToNextHoliday"]].head(10000)
```

```
           Date  IsStateHoliday  DaysSinceHoliday  DaysToNextHoliday
0    2013-01-01               1                 0                  0
1    2013-01-01               1                 0                  0
2    2013-01-01               1                 0                  0
3    2013-01-01               1                 0                  0
4    2013-01-01               1                 0                  0
...         ...             ...               ...                ...
9995 2013-01-09               0                 3                 79
9996 2013-01-09               0                 3                 79
9997 2013-01-09               0                 3                 79
9998 2013-01-09               0                 3                 79
9999 2013-01-09               0                 3                 79

[10000 rows x 4 columns]
```

```
print(df["IsStateHoliday"].value_counts())
print(df.loc[df["IsStateHoliday"]==1, ["Date"]].drop_duplicates().head(10))
```

```
IsStateHoliday
0    986159
1     31050
Name: count, dtype: int64
             Date
0      2013-01-01
5575   2013-01-06
97004  2013-03-29
100349 2013-04-01
133799 2013-05-01
142719 2013-05-09
154984 2013-05-20
166134 2013-05-30
252007 2013-08-15
306624 2013-10-03
```

```
missing_counts = df.isna().sum()
missing_pct = (df.isna().mean() * 100).round(3)

missing_summary = pd.DataFrame({
    "missing_count": missing_counts,
    "missing_percent": missing_pct
}).sort_values("missing_count", ascending=False)

missing_summary[missing_summary["missing_count"] > 0]
```

```
                           missing_count  missing_percent
PromoInterval                     508031           49.944
Promo2SinceYear                   508031           49.944
Promo2SinceWeek                   508031           49.944
CompetitionOpenSinceYear          323348           31.788
CompetitionOpenSinceMonth         323348           31.788
CompetitionDistance                 2642            0.260
```

```
df_z = df.copy()

num_cols = df_z.select_dtypes(include=[np.number]).columns
bin_cols = [c for c in num_cols if df_z[c].dropna().isin([0,1]).all()]
z_cols = [c for c in num_cols if c not in bin_cols]

df_z[z_cols] = df_z[z_cols].fillna(df_z[z_cols].median())

mu = df_z[z_cols].mean()
sigma = df_z[z_cols].std().replace(0, 1)

df_z[z_cols] = (df_z[z_cols] - mu) / sigma

df_z.head()
```

```
      Store  DayOfWeek       Date     Sales  Customers  Open  Promo  \
0  1.728970  -1.000475 2013-01-01 -1.499722  -1.363329     0      0
1 -0.557393  -1.000475 2013-01-01 -1.499722  -1.363329     0      0
2 -0.560500  -1.000475 2013-01-01 -1.499722  -1.363329     0      0
3 -0.563606  -1.000475 2013-01-01 -1.499722  -1.363329     0      0
4 -0.566713  -1.000475 2013-01-01 -1.499722  -1.363329     0      0

  StateHoliday  SchoolHoliday StoreType  ... Promo2SinceYear  \
0            a              1         d  ...        0.104612
1            a              1         d  ...        0.104612
2            a              1         a  ...        0.104612
3            a              1         a  ...       -1.586055
4            a              1         a  ...        0.104612

      PromoInterval  IsStateHoliday  DayOfYear    sin_1y    cos_1y    sin_2y  \
0  Mar,Jun,Sept,Dec               1  -1.587113 -0.179271  1.442531  0.027059
1               NaN               1  -1.587113 -0.179271  1.442531  0.027059
2               NaN               1  -1.587113 -0.179271  1.442531  0.027059
3   Feb,May,Aug,Nov               1  -1.587113 -0.179271  1.442531  0.027059
4               NaN               1  -1.587113 -0.179271  1.442531  0.027059
```

```
     cos_2y  DaysSinceHoliday  DaysToNextHoliday
0  1.368905         -1.104201          -1.011865
1  1.368905         -1.104201          -1.011865
2  1.368905         -1.104201          -1.011865
3  1.368905         -1.104201          -1.011865
4  1.368905         -1.104201          -1.011865

[5 rows x 26 columns]
```

```
df_mm = df.copy()

num_cols = df_mm.select_dtypes(include=[np.number]).columns
df_mm[num_cols] = df_mm[num_cols].fillna(df_mm[num_cols].median())

mn = df_mm[num_cols].min()
mx = df_mm[num_cols].max()
den = (mx - mn).replace(0, 1)

df_mm[num_cols] = (df_mm[num_cols] - mn) / den

df_mm.head()
```

```
      Store  DayOfWeek       Date  Sales  Customers  Open  Promo StateHoliday  \
0  0.000000   0.166667 2013-01-01    0.0        0.0   0.0    0.0            a
1  0.000898   0.166667 2013-01-01    0.0        0.0   0.0    0.0            a
2  0.001795   0.166667 2013-01-01    0.0        0.0   0.0    0.0            a
3  0.002693   0.166667 2013-01-01    0.0        0.0   0.0    0.0            a
4  0.003591   0.166667 2013-01-01    0.0        0.0   0.0    0.0            a

   SchoolHoliday StoreType  ... Promo2SinceYear    PromoInterval  \
0            1.0         c  ...        0.500000              NaN
1            1.0         a  ...        0.166667  Jan,Apr,Jul,Oct
2            1.0         a  ...        0.333333  Jan,Apr,Jul,Oct
3            1.0         c  ...        0.500000              NaN
4            1.0         a  ...        0.500000              NaN

   IsStateHoliday  DayOfYear    sin_1y    cos_1y    sin_2y    cos_2y  \
0             1.0        0.0  0.508604  0.999931  0.517211  0.999723
1             1.0        0.0  0.508604  0.999931  0.517211  0.999723
2             1.0        0.0  0.508604  0.999931  0.517211  0.999723
3             1.0        0.0  0.508604  0.999931  0.517211  0.999723
4             1.0        0.0  0.508604  0.999931  0.517211  0.999723

   DaysSinceHoliday  DaysToNextHoliday
0               0.0                0.0
1               0.0                0.0
```

```
2              0.0                0.0
3              0.0                0.0
4              0.0                0.0

[5 rows x 26 columns]
```

```
df_mm[:]
```

```
            Store  DayOfWeek       Date     Sales  Customers  Open  Promo  \
0        0.000000   0.166667 2013-01-01  0.000000   0.000000   0.0    0.0
1        0.000898   0.166667 2013-01-01  0.000000   0.000000   0.0    0.0
2        0.001795   0.166667 2013-01-01  0.000000   0.000000   0.0    0.0
3        0.002693   0.166667 2013-01-01  0.000000   0.000000   0.0    0.0
4        0.003591   0.166667 2013-01-01  0.000000   0.000000   0.0    0.0
...           ...        ...        ...       ...        ...   ...    ...
1017204  0.996409   0.666667 2015-07-31  0.137734   0.057120   1.0    1.0
1017205  0.997307   0.666667 2015-07-31  0.231667   0.103817   1.0    1.0
1017206  0.998205   0.666667 2015-07-31  0.175423   0.097455   1.0    1.0
1017207  0.999102   0.666667 2015-07-31  0.662030   0.506903   1.0    1.0
1017208  1.000000   0.666667 2015-07-31  0.208900   0.072821   1.0    1.0

        StateHoliday  SchoolHoliday StoreType  ... Promo2SinceYear  \
0                  a            1.0         c  ...        0.500000
1                  a            1.0         a  ...        0.166667
2                  a            1.0         a  ...        0.333333
3                  a            1.0         c  ...        0.500000
4                  a            1.0         a  ...        0.500000
...              ...            ...       ...  ...             ...
1017204            0            1.0         a  ...        0.666667
1017205            0            1.0         c  ...        0.500000
1017206            0            1.0         a  ...        0.500000
1017207            0            1.0         a  ...        0.500000
1017208            0            1.0         d  ...        0.500000

         PromoInterval  IsStateHoliday  DayOfYear    sin_1y    cos_1y  \
0                  0.0             1.0    0.00000  0.508604  0.999931
1                  1.0             1.0    0.00000  0.508604  0.999931
2                  1.0             1.0    0.00000  0.508604  0.999931
3                  0.0             1.0    0.00000  0.508604  0.999931
4                  0.0             1.0    0.00000  0.508604  0.999931
...                ...             ...        ...       ...       ...
1017204            1.0             0.0    0.57967  0.257957  0.062481
1017205            0.0             0.0    0.57967  0.257957  0.062481
1017206            0.0             0.0    0.57967  0.257957  0.062481
1017207            0.0             0.0    0.57967  0.257957  0.062481
1017208            0.0             0.0    0.57967  0.257957  0.062481
```

```
          sin_2y    cos_2y  DaysSinceHoliday  DaysToNextHoliday
0       0.517211  0.999723          0.000000                0.0
1       0.517211  0.999723          0.000000                0.0
2       0.517211  0.999723          0.000000                0.0
3       0.517211  0.999723          0.000000                0.0
4       0.517211  0.999723          0.000000                0.0
...          ...       ...               ...                ...
1017204 0.923608  0.765672          0.542857                0.0
1017205 0.923608  0.765672          0.542857                0.0
1017206 0.923608  0.765672          0.542857                0.0
1017207 0.923608  0.765672          0.542857                0.0
1017208 0.923608  0.765672          0.542857                0.0

[1017209 rows x 26 columns]
```

The PromoInterval column was transformed from a sparse categorical string into a binary numeric feature by mapping the current month to the scheduled promotion cycles. This process eliminated all NaN values by assigning 1 to active promotion months and 0 to others, allowing the model to effectively capture the impact of recurring discounts during training and scaling.

We implemented two independent normalization strategies in separate cells. First, we created df_z and applied Z-score standardization only to non-binary numeric features (leaving 0/1 indicators unchanged), so those features have approximately zero mean and unit variance. Second, we created df_mm and applied Min–Max normalization to scale numeric features into the [0,1] range. Using separate copies ensures the two normalizations do not affect each other

To handle missing values, we first identified all NaN entries and checked which columns contained missing data. We treated missingness based on the meaning of each feature:

Categorical features (e.g., PromoInterval): NaNs often mean "not applicable" (e.g., stores without Promo2).

Numeric features (e.g., competition and Promo2 date/distance fields): when NaNs represent "unknown" or "not reported," we filled them with a robust statistic (typically the median) to avoid losing rows, and (when relevant) we also used an indicator meaning "feature exists/active" (e.g., competition exists vs. not).

Before applying normalization, we ensured there were no remaining NaNs in numeric columns by imputing them (median), and we handled edge cases where a column had zero variance to avoid division-by-zero during scaling.

```python
df = df_mm.sort_values(["Date","Store"]).reset_index(drop=True)

dates = np.array(sorted(df["Date"].unique()))
n = len(dates)
```

```
train_end = int(0.70 * n)
val_end   = int(0.85 * n)

train_dates = set(dates[:train_end])
val_dates   = set(dates[train_end:val_end])
test_dates  = set(dates[val_end:])

train_df = df[df["Date"].isin(train_dates)].reset_index(drop=True)
val_df   = df[df["Date"].isin(val_dates)].reset_index(drop=True)
test_df  = df[df["Date"].isin(test_dates)].reset_index(drop=True)

print("Monotonic:", df["Date"].is_monotonic_increasing)
print(f"Train dates: {train_df['Date'].min()} to {train_df['Date'].max()}")
print(f"Val dates:   {val_df['Date'].min()} to {val_df['Date'].max()}")
print(f"Test dates:  {test_df['Date'].min()} to {test_df['Date'].max()}")

print("No overlap train/val:", train_df["Date"].max() < val_df["Date"].min())
print("No overlap val/test :", val_df["Date"].max() < test_df["Date"].min())
```

```
Monotonic: True
Train dates: 2013-01-01 00:00:00 to 2014-10-21 00:00:00
Val dates:   2014-10-22 00:00:00 to 2015-03-11 00:00:00
Test dates:  2015-03-12 00:00:00 to 2015-07-31 00:00:00
No overlap train/val: True
No overlap val/test : True
```

```
print("Train rows:", len(train_df))
print("Val rows  :", len(val_df))
print("Test rows :", len(test_df))

print("Train unique dates:", train_df["Date"].nunique())
print("Val unique dates  :", val_df["Date"].nunique())
print("Test unique dates :", test_df["Date"].nunique())
```

```
Train rows: 714444
Val rows  : 144435
Test rows : 158330
Train unique dates: 659
Val unique dates  : 141
Test unique dates : 142
```

```
train_df.PromoInterval.value_counts()
```

```
PromoInterval
Jan,Apr,Jul,Oct     206979
Feb,May,Aug,Nov      83297
Mar,Jun,Sept,Dec     68385
```

```
Name: count, dtype: int64
```

We split the dataset into train/validation/test while strictly preserving chronological order. To avoid any date overlap (since each date contains many stores/rows), we performed the split using the unique dates rather than row indices. This guarantees monotonic time ordering and ensures that the validation and test sets contain only future dates relative to the training set (no temporal leakage).

```
import numpy as np

mean_pred = np.full(len(val_df), train_df["Sales"].mean(), dtype=float)
mean_pred = np.where(val_df["Open"].values==0, 0.0, mean_pred)
rmse_mean = np.sqrt(np.mean((val_df["Sales"].values.astype(float) - mean_pred) ** 2))

train_open = train_df[train_df["Open"]==1]
store_mean = train_open.groupby("Store")["Sales"].mean()
store_pred = val_df["Store"].map(store_mean).fillna(train_open["Sales"].mean()).values.astyp
store_pred = np.where(val_df["Open"].values==0, 0.0, store_pred)
rmse_store = np.sqrt(np.mean((val_df["Sales"].values.astype(float) - store_pred) ** 2))

print("Baseline RMSE (Global mean):", rmse_mean)
print("Baseline RMSE (Store mean): ", rmse_store)
```

```
Baseline RMSE (Global mean): 0.07878866888416793
Baseline RMSE (Store mean):  0.046653120646034
```

We report two simple baselines on the validation set: (1) a global-mean predictor that always predicts the average training sales and (2) a stronger store-level baseline that predicts each stores average sales computed from open days in the training set. For both baselines, we force predictions to 0 when Open=0 to match the business logic.

```
df_mm[:]
```

```
            Store  DayOfWeek       Date     Sales  Customers  Open  Promo  \
0        1.000000   0.166667 2013-01-01  0.000000   0.000000   0.0    0.0
1        0.339318   0.166667 2013-01-01  0.000000   0.000000   0.0    0.0
2        0.338420   0.166667 2013-01-01  0.000000   0.000000   0.0    0.0
3        0.337522   0.166667 2013-01-01  0.000000   0.000000   0.0    0.0
4        0.336625   0.166667 2013-01-01  0.000000   0.000000   0.0    0.0
...           ...        ...        ...       ...        ...   ...    ...
1017204  0.668761   0.666667 2015-07-31  0.218575   0.086356   1.0    1.0
1017205  0.669659   0.666667 2015-07-31  0.257707   0.111803   1.0    1.0
1017206  0.670557   0.666667 2015-07-31  0.180044   0.078235   1.0    1.0
1017207  0.665171   0.666667 2015-07-31  0.251739   0.137520   1.0    1.0
1017208  0.000000   0.666667 2015-07-31  0.126664   0.075122   1.0    1.0

        StateHoliday  SchoolHoliday StoreType  ... Promo2SinceYear  \
0                  a            1.0          d  ...          0.500000
1                  a            1.0          d  ...          0.500000
2                  a            1.0          a  ...          0.500000
3                  a            1.0          a  ...          0.166667
4                  a            1.0          a  ...          0.500000
...              ...            ...        ...  ...               ...
1017204            0            1.0          d  ...          0.333333
1017205            0            1.0          c  ...          0.500000
1017206            0            1.0          d  ...          0.333333
1017207            0            1.0          d  ...          0.500000
1017208            0            1.0          c  ...          0.500000

            PromoInterval  IsStateHoliday  DayOfYear    sin_1y    cos_1y  \
0        Mar,Jun,Sept,Dec             1.0    0.00000  0.508604  0.999931
1                     NaN             1.0    0.00000  0.508604  0.999931
2                     NaN             1.0    0.00000  0.508604  0.999931
3         Feb,May,Aug,Nov             1.0    0.00000  0.508604  0.999931
4                     NaN             1.0    0.00000  0.508604  0.999931
...                   ...             ...        ...       ...       ...
1017204  Mar,Jun,Sept,Dec             0.0    0.57967  0.257957  0.062481
1017205               NaN             0.0    0.57967  0.257957  0.062481
1017206   Jan,Apr,Jul,Oct             0.0    0.57967  0.257957  0.062481
1017207               NaN             0.0    0.57967  0.257957  0.062481
1017208               NaN             0.0    0.57967  0.257957  0.062481

           sin_2y    cos_2y  DaysSinceHoliday  DaysToNextHoliday
0        0.517211  0.999723          0.000000                0.0
1        0.517211  0.999723          0.000000                0.0
2        0.517211  0.999723          0.000000                0.0
3        0.517211  0.999723          0.000000                0.0
4        0.517211  0.999723          0.000000                0.0
...           ...       ...               ...                ...
1017204  0.923608  0.765672          0.542857                0.0
1017205  0.923608  0.765672          0.542857                0.0
1017206  0.923608  0.765672          0.542857                0.0
1017207  0.923608  0.765672          0.542857                0.0
1017208  0.923608  0.765672          0.542857                0.0

[1017209 rows x 26 columns]
```

```python
import numpy as np
import pandas as pd

df_mm["Date"] = pd.to_datetime(df_mm["Date"])
df_mm = df_mm.sort_values(["Store","Date"]).reset_index(drop=True)
```

```
if "Year" not in df_mm.columns: df_mm["Year"] = df_mm["Date"].dt.year
if "Month" not in df_mm.columns: df_mm["Month"] = df_mm["Date"].dt.month
if "Day" not in df_mm.columns: df_mm["Day"] = df_mm["Date"].dt.day
if "WeekOfYear" not in df_mm.columns: df_mm["WeekOfYear"] = df_mm["Date"].dt.isocalendar().w
if "IsWeekend" not in df_mm.columns: df_mm["IsWeekend"] = (df_mm["DayOfWeek"] >= 6).astype(i
if "IsMonthStart" not in df_mm.columns: df_mm["IsMonthStart"] = df_mm["Date"].dt.is_month_st
if "IsMonthEnd" not in df_mm.columns: df_mm["IsMonthEnd"] = df_mm["Date"].dt.is_month_end.as
if "Quarter" not in df_mm.columns: df_mm["Quarter"] = df_mm["Date"].dt.quarter

g = df_mm.groupby("Store")["Sales"]
if "SalesLag1" not in df_mm.columns: df_mm["SalesLag1"] = g.shift(1)
if "SalesLag7" not in df_mm.columns: df_mm["SalesLag7"] = g.shift(7)

s1 = g.shift(1)
if "SalesMean7" not in df_mm.columns:
    df_mm["SalesMean7"] = s1.groupby(df_mm["Store"]).rolling(7, min_periods=1).mean().reset_
if "SalesStd7" not in df_mm.columns:
    df_mm["SalesStd7"] = s1.groupby(df_mm["Store"]).rolling(7, min_periods=2).std().reset_in
if "SalesMean30" not in df_mm.columns:
    df_mm["SalesMean30"] = s1.groupby(df_mm["Store"]).rolling(30, min_periods=1).mean().rese
if "SalesStd30" not in df_mm.columns:
    df_mm["SalesStd30"] = s1.groupby(df_mm["Store"]).rolling(30, min_periods=2).std().reset_

dates = np.array(sorted(df_mm["Date"].unique()))
n = len(dates)
train_end = int(0.70 * n)
val_end   = int(0.85 * n)

train_dates = set(dates[:train_end])
val_dates   = set(dates[train_end:val_end])
test_dates  = set(dates[val_end:])

train_df = df_mm[df_mm["Date"].isin(train_dates)].reset_index(drop=True)
val_df   = df_mm[df_mm["Date"].isin(val_dates)].reset_index(drop=True)
test_df  = df_mm[df_mm["Date"].isin(test_dates)].reset_index(drop=True)

missing = [c for c in ["Year","Month","Day","WeekOfYear","IsWeekend","IsMonthStart","IsMonth
                       "SalesLag1","SalesLag7","SalesMean7","SalesStd7","SalesMean30","Sales
print("Missing after build:", missing)
print(train_df.shape, val_df.shape, test_df.shape)
```

```
Missing after build: []
(714444, 40) (144435, 40) (158330, 40)
```

```
import numpy as np

rng = np.random.default_rng(42)
```

```
tr_n = min(50000, Xtr_np.shape[0])
va_n = min(20000, Xva_np.shape[0])

tr_idx = rng.choice(Xtr_np.shape[0], size=tr_n, replace=False)
va_idx = rng.choice(Xva_np.shape[0], size=va_n, replace=False)

Xtr_s = Xtr_np[tr_idx]
ytr_s = ytr[tr_idx]

Xva_s = Xva_np[va_idx]
yva_log_s = yva_log[va_idx]
yva_true_s = yva_true[va_idx]
open_va_s = open_va[va_idx]

print(Xtr_s.shape, Xva_s.shape)
```

```
(50000, 31) (20000, 31)
```

```
import numpy as np
import pandas as pd

feature_cols = [
    "Store","DayOfWeek","Open","Promo","SchoolHoliday","StateHoliday",
    "StoreType","Assortment",
    "CompetitionDistance","Promo2",
    "Year","Month","Day","WeekOfYear","DayOfYear","IsWeekend","IsMonthStart","IsMonthEnd","(
    "SalesLag1","SalesLag7","SalesMean7","SalesStd7","SalesMean30","SalesStd30",
    "sin_1y","cos_1y","sin_2y","cos_2y",
    "DaysSinceHoliday","DaysToNextHoliday"
]

cat_cols = ["Store","DayOfWeek","StateHoliday","StoreType","Assortment"]
cat_cols = [c for c in cat_cols if c in feature_cols]
num_cols = [c for c in feature_cols if c not in cat_cols]

def rmse(y_true, y_pred):
    e = y_true - y_pred
    return float(np.sqrt(np.mean(e * e)))

def prepare_matrices(train_df, val_df, test_df):
    Xtr = train_df[feature_cols].copy()
    Xva = val_df[feature_cols].copy()
    Xte = test_df[feature_cols].copy()

    Xtr[num_cols] = Xtr[num_cols].apply(pd.to_numeric, errors="coerce").astype("float32")
    Xva[num_cols] = Xva[num_cols].apply(pd.to_numeric, errors="coerce").astype("float32")
    Xte[num_cols] = Xte[num_cols].apply(pd.to_numeric, errors="coerce").astype("float32")

    med = Xtr[num_cols].median()
    Xtr[num_cols] = Xtr[num_cols].fillna(med)
    Xva[num_cols] = Xva[num_cols].fillna(med)
    Xte[num_cols] = Xte[num_cols].fillna(med)

    for c in cat_cols:
        Xtr[c] = Xtr[c].astype("category")
        cats = Xtr[c].cat.categories.astype(str)
        Xva[c] = pd.Categorical(Xva[c].astype(str), categories=cats)
        Xte[c] = pd.Categorical(Xte[c].astype(str), categories=cats)
        Xtr[c] = Xtr[c].cat.codes.astype("int32")
        Xva[c] = Xva[c].cat.codes.astype("int32")
        Xte[c] = Xte[c].cat.codes.astype("int32")

    Xtr_np = Xtr.to_numpy(dtype=np.float32)
    Xva_np = Xva.to_numpy(dtype=np.float32)
    Xte_np = Xte.to_numpy(dtype=np.float32)

    ytr_log = np.log1p(train_df["Sales"].astype("float32").values)
    yva_log = np.log1p(val_df["Sales"].astype("float32").values)
    yva_norm = val_df["Sales"].astype("float32").values
    yte_norm = test_df["Sales"].astype("float32").values
    open_va = val_df["Open"].astype("int8").values
    open_te = test_df["Open"].astype("int8").values

    return Xtr_np, ytr_log, Xva_np, yva_log, yva_norm, open_va, Xte_np, yte_norm, open_te

class _Node:
    __slots__ = ("is_leaf","value","fidx","thr","left","right")
    def __init__(self, is_leaf, value=None, fidx=None, thr=None, left=None, right=None):
        self.is_leaf = is_leaf
        self.value = value
        self.fidx = fidx
        self.thr = thr
        self.left = left
        self.right = right

class XGBoostScratch:
    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=4,
                 min_child_weight=10.0, reg_lambda=1.0, gamma=0.0,
                 subsample=0.8, colsample=0.8, max_bins=16, random_state=42):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.min_child_weight = min_child_weight
        self.reg_lambda = reg_lambda
        self.gamma = gamma
        self.subsample = subsample
        self.colsample = colsample
        self.max_bins = max_bins
        self.rng = np.random.default_rng(random_state)
        self.trees = []
        self.base_score = 0.0

    def _leaf_weight(self, G, H):
        return -G / (H + self.reg_lambda)

    def _score(self, G, H):
        return (G * G) / (H + self.reg_lambda)

    def _best_split(self, X, g, h, idxs, feat_idxs):
        G, H = g[idxs].sum(), h[idxs].sum()
        parent_score = self._score(G, H)
        best_gain, best_f, best_thr = -1e30, None, None
        best_left, best_right = None, None
        for f in feat_idxs:
            x = X[idxs, f]
            if float(x.min()) == float(x.max()): continue
            qs = np.linspace(0.0, 1.0, self.max_bins + 2)[1:-1]
            thr_list = np.unique(np.quantile(x, qs))
            if thr_list.size == 0: continue
            for thr in thr_list:
                m = x <= thr
                if not m.any() or m.all(): continue
                l_idx, r_idx = idxs[m], idxs[~m]
                Hl, Hr = h[l_idx].sum(), h[r_idx].sum()
                if Hl < self.min_child_weight or Hr < self.min_child_weight: continue
                gain = self._score(g[l_idx].sum(), Hl) + self._score(g[r_idx].sum(), Hr) - p
                if gain > best_gain:
                    best_gain, best_f, best_thr = gain, int(f), float(thr)
                    best_left, best_right = l_idx, r_idx
        return best_gain, best_f, best_thr, best_left, best_right

    def _build(self, X, g, h, idxs, depth, feat_idxs):
        G, H = g[idxs].sum(), h[idxs].sum()
        if depth >= self.max_depth or idxs.size <= 1 or H < self.min_child_weight:
            return _Node(True, value=float(self._leaf_weight(G, H)))
        gain, f, thr, left, right = self._best_split(X, g, h, idxs, feat_idxs)
        if f is None or gain <= 0:
            return _Node(True, value=float(self._leaf_weight(G, H)))
        return _Node(False, fidx=f, thr=thr,
                     left=self._build(X, g, h, left, depth + 1, feat_idxs),
                     right=self._build(X, g, h, right, depth + 1, feat_idxs))

    def _pred_tree(self, node, X):
        out = np.empty(X.shape[0], dtype=np.float32)
        for i in range(X.shape[0]):
            curr = node
            while not curr.is_leaf:
                curr = curr.left if X[i, curr.fidx] <= curr.thr else curr.right
            out[i] = curr.value
        return out

    def fit(self, X, y_log, X_val=None, y_val_log=None, early_stopping_rounds=10, print_ever
        n, d = X.shape
        self.base_score = float(y_log.mean())
        pred = np.full(n, self.base_score, dtype=np.float32)
        pred_val = np.full(X_val.shape[0], self.base_score, dtype=np.float32) if X_val is no
        best_rmse, best_k, bad = 1e30, 0, 0
        for k in range(1, self.n_estimators + 1):
            rows = self.rng.choice(n, size=int(self.subsample * n), replace=False)
            feats = self.rng.choice(np.arange(d), size=int(self.colsample * d), replace=Fals
            g, h = (pred - y_log), np.ones_like(pred)
            tree = self._build(X, g, h, rows, 0, feats)
            self.trees.append(tree)
            pred += self.learning_rate * self._pred_tree(tree, X)
            train_err = np.sqrt(np.mean((pred - y_log) ** 2))
            if X_val is not None:
                pred_val += self.learning_rate * self._pred_tree(tree, X_val)
                val_err = np.sqrt(np.mean((pred_val - y_val_log) ** 2))
                if k % print_every == 0:
                    print(f"Iter {k:3d} | Train RMSE(log): {train_err:.5f} | Val RMSE(log):
                if val_err < best_rmse:
                    best_rmse, best_k, bad = val_err, k, 0
                else:
                    bad += 1
                    if bad >= early_stopping_rounds:
                        print(f"Early stopping at {k}")
                        break
        if X_val is not None: self.trees = self.trees[:best_k]

    def predict(self, X):
        p = np.full(X.shape[0], self.base_score, dtype=np.float32)
        for t in self.trees: p += self.learning_rate * self._pred_tree(t, X)
        return p
```

```python
Xtr_np, ytr_log, Xva_np, yva_log, yva_norm, open_va, Xte_np, yte_norm, open_te = prepare_mat

model = XGBoostScratch(n_estimators=100, learning_rate=0.1, max_depth=4)
model.fit(Xtr_np, ytr_log, X_val=Xva_np, y_val_log=yva_log, print_every=1)

s_min, s_max = mn["Sales"], mx["Sales"]
p_va_euro = np.expm1(model.predict(Xva_np)) * (s_max - s_min) + s_min
p_te_euro = np.expm1(model.predict(Xte_np)) * (s_max - s_min) + s_min
y_va_euro = yva_norm * (s_max - s_min) + s_min
y_te_euro = yte_norm * (s_max - s_min) + s_min

p_va_euro = np.where(open_va == 0, 0, p_va_euro)
p_te_euro = np.where(open_te == 0, 0, p_te_euro)

print(f"\nValidation RMSE: {rmse(y_va_euro, p_va_euro):.2f} Euro")
print(f"Test RMSE: {rmse(y_te_euro, p_te_euro):.2f} Euro")
```

```
Iter   1 | Train RMSE(log): 0.07269 | Val RMSE(log): 0.07630
Iter   2 | Train RMSE(log): 0.06681 | Val RMSE(log): 0.07012
Iter   3 | Train RMSE(log): 0.06159 | Val RMSE(log): 0.06459
Iter   4 | Train RMSE(log): 0.05715 | Val RMSE(log): 0.05987
Iter   5 | Train RMSE(log): 0.05450 | Val RMSE(log): 0.05713
Iter   6 | Train RMSE(log): 0.05067 | Val RMSE(log): 0.05311
Iter   7 | Train RMSE(log): 0.04756 | Val RMSE(log): 0.04980
Iter   8 | Train RMSE(log): 0.04454 | Val RMSE(log): 0.04663
Iter   9 | Train RMSE(log): 0.04188 | Val RMSE(log): 0.04386
Iter  10 | Train RMSE(log): 0.03955 | Val RMSE(log): 0.04145
Iter  11 | Train RMSE(log): 0.03737 | Val RMSE(log): 0.03924
Iter  12 | Train RMSE(log): 0.03562 | Val RMSE(log): 0.03749
Iter  13 | Train RMSE(log): 0.03413 | Val RMSE(log): 0.03595
Iter  14 | Train RMSE(log): 0.03260 | Val RMSE(log): 0.03442
Iter  15 | Train RMSE(log): 0.03132 | Val RMSE(log): 0.03319
Iter  16 | Train RMSE(log): 0.03019 | Val RMSE(log): 0.03211
Iter  17 | Train RMSE(log): 0.02920 | Val RMSE(log): 0.03116
Iter  18 | Train RMSE(log): 0.02838 | Val RMSE(log): 0.03036
Iter  19 | Train RMSE(log): 0.02759 | Val RMSE(log): 0.02965
Iter  20 | Train RMSE(log): 0.02688 | Val RMSE(log): 0.02897
Iter  21 | Train RMSE(log): 0.02622 | Val RMSE(log): 0.02838
Iter  22 | Train RMSE(log): 0.02567 | Val RMSE(log): 0.02788
Iter  23 | Train RMSE(log): 0.02523 | Val RMSE(log): 0.02746
Iter  24 | Train RMSE(log): 0.02482 | Val RMSE(log): 0.02709
Iter  25 | Train RMSE(log): 0.02440 | Val RMSE(log): 0.02673
Iter  26 | Train RMSE(log): 0.02406 | Val RMSE(log): 0.02643
Iter  27 | Train RMSE(log): 0.02372 | Val RMSE(log): 0.02616
Iter  28 | Train RMSE(log): 0.02344 | Val RMSE(log): 0.02573
Iter  29 | Train RMSE(log): 0.02306 | Val RMSE(log): 0.02547
Iter  30 | Train RMSE(log): 0.02277 | Val RMSE(log): 0.02493
Iter  31 | Train RMSE(log): 0.02245 | Val RMSE(log): 0.02470
Iter  32 | Train RMSE(log): 0.02220 | Val RMSE(log): 0.02445
Iter  33 | Train RMSE(log): 0.02194 | Val RMSE(log): 0.02422
Iter  34 | Train RMSE(log): 0.02172 | Val RMSE(log): 0.02371
Iter  35 | Train RMSE(log): 0.02150 | Val RMSE(log): 0.02359
Iter  36 | Train RMSE(log): 0.02131 | Val RMSE(log): 0.02345
Iter  37 | Train RMSE(log): 0.02116 | Val RMSE(log): 0.02321
Iter  38 | Train RMSE(log): 0.02102 | Val RMSE(log): 0.02310
Iter  39 | Train RMSE(log): 0.02088 | Val RMSE(log): 0.02289
Iter  40 | Train RMSE(log): 0.02073 | Val RMSE(log): 0.02278
Iter  41 | Train RMSE(log): 0.02058 | Val RMSE(log): 0.02235
Iter  42 | Train RMSE(log): 0.02044 | Val RMSE(log): 0.02229
Iter  43 | Train RMSE(log): 0.02031 | Val RMSE(log): 0.02206
Iter  44 | Train RMSE(log): 0.02018 | Val RMSE(log): 0.02192
Iter  45 | Train RMSE(log): 0.02010 | Val RMSE(log): 0.02186
Iter  46 | Train RMSE(log): 0.01996 | Val RMSE(log): 0.02180
Iter  47 | Train RMSE(log): 0.01987 | Val RMSE(log): 0.02176
Iter  48 | Train RMSE(log): 0.01977 | Val RMSE(log): 0.02171
Iter  49 | Train RMSE(log): 0.01967 | Val RMSE(log): 0.02158
Iter  50 | Train RMSE(log): 0.01957 | Val RMSE(log): 0.02150
Iter  51 | Train RMSE(log): 0.01951 | Val RMSE(log): 0.02148
Iter  52 | Train RMSE(log): 0.01945 | Val RMSE(log): 0.02144
Iter  53 | Train RMSE(log): 0.01936 | Val RMSE(log): 0.02141
Iter  54 | Train RMSE(log): 0.01929 | Val RMSE(log): 0.02137
Iter  55 | Train RMSE(log): 0.01920 | Val RMSE(log): 0.02131
Iter  56 | Train RMSE(log): 0.01916 | Val RMSE(log): 0.02130
Iter  57 | Train RMSE(log): 0.01912 | Val RMSE(log): 0.02129
Iter  58 | Train RMSE(log): 0.01906 | Val RMSE(log): 0.02126
Iter  59 | Train RMSE(log): 0.01897 | Val RMSE(log): 0.02121
Iter  60 | Train RMSE(log): 0.01892 | Val RMSE(log): 0.02118
Iter  61 | Train RMSE(log): 0.01885 | Val RMSE(log): 0.02111
Iter  62 | Train RMSE(log): 0.01879 | Val RMSE(log): 0.02107
Iter  63 | Train RMSE(log): 0.01873 | Val RMSE(log): 0.02106
Iter  64 | Train RMSE(log): 0.01867 | Val RMSE(log): 0.02103
Iter  65 | Train RMSE(log): 0.01862 | Val RMSE(log): 0.02093
Iter  66 | Train RMSE(log): 0.01856 | Val RMSE(log): 0.02087
Iter  67 | Train RMSE(log): 0.01852 | Val RMSE(log): 0.02082
Iter  68 | Train RMSE(log): 0.01848 | Val RMSE(log): 0.02078
Iter  69 | Train RMSE(log): 0.01843 | Val RMSE(log): 0.02067
Iter  70 | Train RMSE(log): 0.01841 | Val RMSE(log): 0.02067
Iter  71 | Train RMSE(log): 0.01834 | Val RMSE(log): 0.02061
Iter  72 | Train RMSE(log): 0.01828 | Val RMSE(log): 0.02057
Iter  73 | Train RMSE(log): 0.01825 | Val RMSE(log): 0.02059
Iter  74 | Train RMSE(log): 0.01819 | Val RMSE(log): 0.02055
Iter  75 | Train RMSE(log): 0.01815 | Val RMSE(log): 0.02055
Iter  76 | Train RMSE(log): 0.01812 | Val RMSE(log): 0.02053
Iter  77 | Train RMSE(log): 0.01807 | Val RMSE(log): 0.02052
Iter  78 | Train RMSE(log): 0.01805 | Val RMSE(log): 0.02052
Iter  79 | Train RMSE(log): 0.01801 | Val RMSE(log): 0.02044
Iter  80 | Train RMSE(log): 0.01796 | Val RMSE(log): 0.02042
Iter  81 | Train RMSE(log): 0.01793 | Val RMSE(log): 0.02038
Iter  82 | Train RMSE(log): 0.01790 | Val RMSE(log): 0.02038
Iter  83 | Train RMSE(log): 0.01786 | Val RMSE(log): 0.02036
Iter  84 | Train RMSE(log): 0.01784 | Val RMSE(log): 0.02037
Iter  85 | Train RMSE(log): 0.01779 | Val RMSE(log): 0.02030
Iter  86 | Train RMSE(log): 0.01776 | Val RMSE(log): 0.02030
Iter  87 | Train RMSE(log): 0.01773 | Val RMSE(log): 0.02028
Iter  88 | Train RMSE(log): 0.01771 | Val RMSE(log): 0.02025
Iter  89 | Train RMSE(log): 0.01767 | Val RMSE(log): 0.02022
Iter  90 | Train RMSE(log): 0.01763 | Val RMSE(log): 0.02015
Iter  91 | Train RMSE(log): 0.01760 | Val RMSE(log): 0.02013
Iter  92 | Train RMSE(log): 0.01757 | Val RMSE(log): 0.02012
Iter  93 | Train RMSE(log): 0.01754 | Val RMSE(log): 0.02009
Iter  94 | Train RMSE(log): 0.01752 | Val RMSE(log): 0.02010
Iter  95 | Train RMSE(log): 0.01750 | Val RMSE(log): 0.02010
Iter  96 | Train RMSE(log): 0.01746 | Val RMSE(log): 0.02006
Iter  97 | Train RMSE(log): 0.01743 | Val RMSE(log): 0.02005
Iter  98 | Train RMSE(log): 0.01740 | Val RMSE(log): 0.02003
Iter  99 | Train RMSE(log): 0.01736 | Val RMSE(log): 0.02000
Iter 100 | Train RMSE(log): 0.01735 | Val RMSE(log): 0.02000

Validation RMSE: 1008.17 Euro
Test RMSE: 910.21 Euro
```

```python
import numpy as np
import pandas as pd

feature_cols = [
    "Store","DayOfWeek","Open","Promo","SchoolHoliday","StateHoliday",
    "StoreType","Assortment",
    "CompetitionDistance","Promo2",
    "Year","Month","Day","WeekOfYear","DayOfYear","IsWeekend","IsMonthStart","IsMonthEnd","
    "SalesLag1","SalesLag7","SalesMean7","SalesStd7","SalesMean30","SalesStd30",
    "sin_1y","cos_1y","sin_2y","cos_2y",
    "DaysSinceHoliday","DaysToNextHoliday"
]

cat_cols = ["Store","DayOfWeek","StateHoliday","StoreType","Assortment"]
cat_cols = [c for c in cat_cols if c in feature_cols]
num_cols = [c for c in feature_cols if c not in cat_cols]
```

```
def rmse(y_true, y_pred):
    e = y_true - y_pred
    return float(np.sqrt(np.mean(e * e)))

def mape_custom(y_true, y_pred):
    mask = y_true > 0
    return float(np.mean(np.abs((y_true[mask] - y_pred[mask]) / y_true[mask])) * 100)

def r2_custom(y_true, y_pred):
    ss_res = np.sum((y_true - y_pred) ** 2)
    ss_tot = np.sum((y_true - np.mean(y_true)) ** 2)
    return float(1 - (ss_res / ss_tot))

def prepare_matrices(train_df, val_df, test_df):
    Xtr = train_df[feature_cols].copy()
    Xva = val_df[feature_cols].copy()
    Xte = test_df[feature_cols].copy()

    Xtr[num_cols] = Xtr[num_cols].apply(pd.to_numeric, errors="coerce").astype("float32")
    Xva[num_cols] = Xva[num_cols].apply(pd.to_numeric, errors="coerce").astype("float32")
    Xte[num_cols] = Xte[num_cols].apply(pd.to_numeric, errors="coerce").astype("float32")

    med = Xtr[num_cols].median()
    Xtr[num_cols] = Xtr[num_cols].fillna(med)
    Xva[num_cols] = Xva[num_cols].fillna(med)
    Xte[num_cols] = Xte[num_cols].fillna(med)

    for c in cat_cols:
        Xtr[c] = Xtr[c].astype("category")
        cats = Xtr[c].cat.categories.astype(str)
        Xva[c] = pd.Categorical(Xva[c].astype(str), categories=cats)
        Xte[c] = pd.Categorical(Xte[c].astype(str), categories=cats)
        Xtr[c] = Xtr[c].cat.codes.astype("int32")
        Xva[c] = Xva[c].cat.codes.astype("int32")
        Xte[c] = Xte[c].cat.codes.astype("int32")

    Xtr_np = Xtr.to_numpy(dtype=np.float32)
    Xva_np = Xva.to_numpy(dtype=np.float32)
    Xte_np = Xte.to_numpy(dtype=np.float32)

    ytr_log = np.log1p(train_df["Sales"].astype("float32").values)
    yva_log = np.log1p(val_df["Sales"].astype("float32").values)
    yva_norm = val_df["Sales"].astype("float32").values
    yte_norm = test_df["Sales"].astype("float32").values
    open_va = val_df["Open"].astype("int8").values
    open_te = test_df["Open"].astype("int8").values
    return Xtr_np, ytr_log, Xva_np, yva_log, yva_norm, open_va, Xte_np, yte_norm, open_te

class _Node:
    __slots__ = ("is_leaf","value","fidx","thr","left","right")
    def __init__(self, is_leaf, value=None, fidx=None, thr=None, left=None, right=None):
        self.is_leaf = is_leaf
        self.value = value
        self.fidx = fidx
        self.thr = thr
        self.left = left
        self.right = right

class XGBoostScratch:
    def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=4,
                 min_child_weight=10.0, reg_lambda=1.0, gamma=0.0,
                 subsample=0.8, colsample=0.8, max_bins=16, random_state=42):
        self.n_estimators = n_estimators
        self.learning_rate = learning_rate
        self.max_depth = max_depth
        self.min_child_weight = min_child_weight
        self.reg_lambda = reg_lambda
        self.gamma = gamma
        self.subsample = subsample
        self.colsample = colsample
        self.max_bins = max_bins
        self.rng = np.random.default_rng(random_state)
        self.trees = []
        self.base_score = 0.0

    def _leaf_weight(self, G, H):
        return -G / (H + self.reg_lambda)

    def _score(self, G, H):
        return (G * G) / (H + self.reg_lambda)

    def _best_split(self, X, g, h, idxs, feat_idxs):
        G, H = g[idxs].sum(), h[idxs].sum()
        parent_score = self._score(G, H)
        best_gain, best_f, best_thr = -1e30, None, None
        best_left, best_right = None, None
        for f in feat_idxs:
            x = X[idxs, f]
            if float(x.min()) == float(x.max()): continue
            qs = np.linspace(0.0, 1.0, self.max_bins + 2)[1:-1]
            thr_list = np.unique(np.quantile(x, qs))
            if thr_list.size == 0: continue
            for thr in thr_list:
                m = x <= thr
                if not m.any() or m.all(): continue
                l_idx, r_idx = idxs[m], idxs[~m]
                Hl, Hr = h[l_idx].sum(), h[r_idx].sum()
                if Hl < self.min_child_weight or Hr < self.min_child_weight: continue
                gain = self._score(g[l_idx].sum(), Hl) + self._score(g[r_idx].sum(), Hr) - p
                if gain > best_gain:
                    best_gain, best_f, best_thr = gain, int(f), float(thr)
                    best_left, best_right = l_idx, r_idx
        return best_gain, best_f, best_thr, best_left, best_right

    def _build(self, X, g, h, idxs, depth, feat_idxs):
        G, H = g[idxs].sum(), h[idxs].sum()
        if depth >= self.max_depth or idxs.size <= 1 or H < self.min_child_weight:
            return _Node(True, value=float(self._leaf_weight(G, H)))
        gain, f, thr, left, right = self._best_split(X, g, h, idxs, feat_idxs)
        if f is None or gain <= 0:
            return _Node(True, value=float(self._leaf_weight(G, H)))
        return _Node(False, fidx=f, thr=thr,
                     left=self._build(X, g, h, left, depth + 1, feat_idxs),
                     right=self._build(X, g, h, right, depth + 1, feat_idxs))

    def _pred_tree(self, node, X):
        out = np.empty(X.shape[0], dtype=np.float32)
        for i in range(X.shape[0]):
            curr = node
            while not curr.is_leaf:
                curr = curr.left if X[i, curr.fidx] <= curr.thr else curr.right
            out[i] = curr.value
        return out

    def fit(self, X, y_log, X_val=None, y_val_log=None, early_stopping_rounds=10, print_ever
        n, d = X.shape
        self.base_score = float(y_log.mean())
        pred = np.full(n, self.base_score, dtype=np.float32)
        pred_val = np.full(X_val.shape[0], self.base_score, dtype=np.float32) if X_val is no
        best_rmse, best_k, bad = 1e30, 0, 0
        for k in range(1, self.n_estimators + 1):
            rows = self.rng.choice(n, size=int(self.subsample * n), replace=False)
            feats = self.rng.choice(np.arange(d), size=int(self.colsample * d), replace=Fals
            g, h = (pred - y_log), np.ones_like(pred)
            tree = self._build(X, g, h, rows, 0, feats)
            self.trees.append(tree)
            pred += self.learning_rate * self._pred_tree(tree, X)
            train_err = np.sqrt(np.mean((pred - y_log) ** 2))
            if X_val is not None:
                pred_val += self.learning_rate * self._pred_tree(tree, X_val)
                val_err = np.sqrt(np.mean((pred_val - y_val_log) ** 2))
                if k % print_every == 0:
                    print(f"Iter {k:3d} | Train RMSE(log): {train_err:.5f} | Val RMSE(log):
                if val_err < best_rmse:
                    best_rmse, best_k, bad = val_err, k, 0
                else:
                    bad += 1
                    if bad >= early_stopping_rounds:
                        print(f"Early stopping at {k}")
                        break
        if X_val is not None: self.trees = self.trees[:best_k]

    def predict(self, X):
        p = np.full(X.shape[0], self.base_score, dtype=np.float32)
        for t in self.trees: p += self.learning_rate * self._pred_tree(t, X)
        return p

Xtr_np, ytr_log, Xva_np, yva_log, yva_norm, open_va, Xte_np, yte_norm, open_te = prepare_mat

model = XGBoostScratch(n_estimators=100, learning_rate=0.1, max_depth=4)
model.fit(Xtr_np, ytr_log, X_val=Xva_np, y_val_log=yva_log, print_every=1)

s_min, s_max = mn["Sales"], mx["Sales"]
s_range = s_max - s_min

p_va_euro = np.expm1(model.predict(Xva_np)) * s_range + s_min
y_va_euro = yva_norm * s_range + s_min
p_va_euro = np.where(open_va == 0, 0, p_va_euro)

p_te_euro = np.expm1(model.predict(Xte_np)) * s_range + s_min
y_te_euro = yte_norm * s_range + s_min
p_te_euro = np.where(open_te == 0, 0, p_te_euro)

print("\n" + "="*40)
print("FINAL PERFORMANCE METRICS")
print("="*40)
print(f"Validation:")
print(f"  RMSE: {rmse(y_va_euro, p_va_euro):.2f} Euro")
print(f"  MAPE: {mape_custom(y_va_euro, p_va_euro):.2f}%")
print(f"  R2  : {r2_custom(y_va_euro, p_va_euro):.4f}")
print("-" * 40)
print(f"Test:")
print(f"  RMSE: {rmse(y_te_euro, p_te_euro):.2f} Euro")
print(f"  MAPE: {mape_custom(y_te_euro, p_te_euro):.2f}%")
print(f"  R2  : {r2_custom(y_te_euro, p_te_euro):.4f}")
print("="*40)
```

```
Iter   1 | Train RMSE(log): 0.07269 | Val RMSE(log): 0.07630
Iter   2 | Train RMSE(log): 0.06681 | Val RMSE(log): 0.07012
Iter   3 | Train RMSE(log): 0.06159 | Val RMSE(log): 0.06459
Iter   4 | Train RMSE(log): 0.05715 | Val RMSE(log): 0.05987
Iter   5 | Train RMSE(log): 0.05450 | Val RMSE(log): 0.05713
Iter   6 | Train RMSE(log): 0.05067 | Val RMSE(log): 0.05311
Iter   7 | Train RMSE(log): 0.04756 | Val RMSE(log): 0.04980
Iter   8 | Train RMSE(log): 0.04454 | Val RMSE(log): 0.04663
Iter   9 | Train RMSE(log): 0.04188 | Val RMSE(log): 0.04386
Iter  10 | Train RMSE(log): 0.03955 | Val RMSE(log): 0.04145
Iter  11 | Train RMSE(log): 0.03737 | Val RMSE(log): 0.03924
Iter  12 | Train RMSE(log): 0.03562 | Val RMSE(log): 0.03749
Iter  13 | Train RMSE(log): 0.03413 | Val RMSE(log): 0.03595
Iter  14 | Train RMSE(log): 0.03260 | Val RMSE(log): 0.03442
Iter  15 | Train RMSE(log): 0.03132 | Val RMSE(log): 0.03319
Iter  16 | Train RMSE(log): 0.03019 | Val RMSE(log): 0.03211
Iter  17 | Train RMSE(log): 0.02920 | Val RMSE(log): 0.03116
Iter  18 | Train RMSE(log): 0.02838 | Val RMSE(log): 0.03036
Iter  19 | Train RMSE(log): 0.02759 | Val RMSE(log): 0.02965
Iter  20 | Train RMSE(log): 0.02688 | Val RMSE(log): 0.02897
Iter  21 | Train RMSE(log): 0.02622 | Val RMSE(log): 0.02838
Iter  22 | Train RMSE(log): 0.02567 | Val RMSE(log): 0.02788
Iter  23 | Train RMSE(log): 0.02523 | Val RMSE(log): 0.02746
Iter  24 | Train RMSE(log): 0.02482 | Val RMSE(log): 0.02709
Iter  25 | Train RMSE(log): 0.02440 | Val RMSE(log): 0.02673
Iter  26 | Train RMSE(log): 0.02406 | Val RMSE(log): 0.02643
Iter  27 | Train RMSE(log): 0.02372 | Val RMSE(log): 0.02616
Iter  28 | Train RMSE(log): 0.02344 | Val RMSE(log): 0.02573
Iter  29 | Train RMSE(log): 0.02306 | Val RMSE(log): 0.02547
Iter  30 | Train RMSE(log): 0.02277 | Val RMSE(log): 0.02493
Iter  31 | Train RMSE(log): 0.02245 | Val RMSE(log): 0.02470
Iter  32 | Train RMSE(log): 0.02220 | Val RMSE(log): 0.02445
Iter  33 | Train RMSE(log): 0.02194 | Val RMSE(log): 0.02422
Iter  34 | Train RMSE(log): 0.02172 | Val RMSE(log): 0.02371
Iter  35 | Train RMSE(log): 0.02150 | Val RMSE(log): 0.02359
Iter  36 | Train RMSE(log): 0.02131 | Val RMSE(log): 0.02345
Iter  37 | Train RMSE(log): 0.02116 | Val RMSE(log): 0.02321
Iter  38 | Train RMSE(log): 0.02102 | Val RMSE(log): 0.02310
Iter  39 | Train RMSE(log): 0.02088 | Val RMSE(log): 0.02289
Iter  40 | Train RMSE(log): 0.02073 | Val RMSE(log): 0.02278
Iter  41 | Train RMSE(log): 0.02058 | Val RMSE(log): 0.02235
Iter  42 | Train RMSE(log): 0.02044 | Val RMSE(log): 0.02229
Iter  43 | Train RMSE(log): 0.02031 | Val RMSE(log): 0.02206
Iter  44 | Train RMSE(log): 0.02018 | Val RMSE(log): 0.02192
Iter  45 | Train RMSE(log): 0.02010 | Val RMSE(log): 0.02186
Iter  46 | Train RMSE(log): 0.01996 | Val RMSE(log): 0.02180
Iter  47 | Train RMSE(log): 0.01987 | Val RMSE(log): 0.02176
Iter  48 | Train RMSE(log): 0.01977 | Val RMSE(log): 0.02171
Iter  49 | Train RMSE(log): 0.01967 | Val RMSE(log): 0.02158
Iter  50 | Train RMSE(log): 0.01957 | Val RMSE(log): 0.02150
Iter  51 | Train RMSE(log): 0.01951 | Val RMSE(log): 0.02148
Iter  52 | Train RMSE(log): 0.01945 | Val RMSE(log): 0.02144
Iter  53 | Train RMSE(log): 0.01936 | Val RMSE(log): 0.02141
Iter  54 | Train RMSE(log): 0.01929 | Val RMSE(log): 0.02137
Iter  55 | Train RMSE(log): 0.01920 | Val RMSE(log): 0.02131
Iter  56 | Train RMSE(log): 0.01916 | Val RMSE(log): 0.02130
Iter  57 | Train RMSE(log): 0.01912 | Val RMSE(log): 0.02129
Iter  58 | Train RMSE(log): 0.01906 | Val RMSE(log): 0.02126
Iter  59 | Train RMSE(log): 0.01897 | Val RMSE(log): 0.02121
Iter  60 | Train RMSE(log): 0.01892 | Val RMSE(log): 0.02118
Iter  61 | Train RMSE(log): 0.01885 | Val RMSE(log): 0.02111
Iter  62 | Train RMSE(log): 0.01879 | Val RMSE(log): 0.02107
Iter  63 | Train RMSE(log): 0.01873 | Val RMSE(log): 0.02106
Iter  64 | Train RMSE(log): 0.01867 | Val RMSE(log): 0.02103
Iter  65 | Train RMSE(log): 0.01862 | Val RMSE(log): 0.02093
Iter  66 | Train RMSE(log): 0.01856 | Val RMSE(log): 0.02087
Iter  67 | Train RMSE(log): 0.01852 | Val RMSE(log): 0.02082
Iter  68 | Train RMSE(log): 0.01848 | Val RMSE(log): 0.02078
Iter  69 | Train RMSE(log): 0.01843 | Val RMSE(log): 0.02067
Iter  70 | Train RMSE(log): 0.01841 | Val RMSE(log): 0.02067
Iter  71 | Train RMSE(log): 0.01834 | Val RMSE(log): 0.02061
Iter  72 | Train RMSE(log): 0.01828 | Val RMSE(log): 0.02057
Iter  73 | Train RMSE(log): 0.01825 | Val RMSE(log): 0.02059
Iter  74 | Train RMSE(log): 0.01819 | Val RMSE(log): 0.02055
Iter  75 | Train RMSE(log): 0.01815 | Val RMSE(log): 0.02055
Iter  76 | Train RMSE(log): 0.01812 | Val RMSE(log): 0.02053
Iter  77 | Train RMSE(log): 0.01807 | Val RMSE(log): 0.02052
Iter  78 | Train RMSE(log): 0.01805 | Val RMSE(log): 0.02052
Iter  79 | Train RMSE(log): 0.01801 | Val RMSE(log): 0.02044
Iter  80 | Train RMSE(log): 0.01796 | Val RMSE(log): 0.02042
Iter  81 | Train RMSE(log): 0.01793 | Val RMSE(log): 0.02038
Iter  82 | Train RMSE(log): 0.01790 | Val RMSE(log): 0.02038
Iter  83 | Train RMSE(log): 0.01786 | Val RMSE(log): 0.02036
Iter  84 | Train RMSE(log): 0.01784 | Val RMSE(log): 0.02037
Iter  85 | Train RMSE(log): 0.01779 | Val RMSE(log): 0.02030
Iter  86 | Train RMSE(log): 0.01776 | Val RMSE(log): 0.02030
Iter  87 | Train RMSE(log): 0.01773 | Val RMSE(log): 0.02028
Iter  88 | Train RMSE(log): 0.01771 | Val RMSE(log): 0.02025
Iter  89 | Train RMSE(log): 0.01767 | Val RMSE(log): 0.02022
Iter  90 | Train RMSE(log): 0.01763 | Val RMSE(log): 0.02015
Iter  91 | Train RMSE(log): 0.01760 | Val RMSE(log): 0.02013
Iter  92 | Train RMSE(log): 0.01757 | Val RMSE(log): 0.02012
Iter  93 | Train RMSE(log): 0.01754 | Val RMSE(log): 0.02009
Iter  94 | Train RMSE(log): 0.01752 | Val RMSE(log): 0.02010
Iter  95 | Train RMSE(log): 0.01750 | Val RMSE(log): 0.02010
Iter  96 | Train RMSE(log): 0.01746 | Val RMSE(log): 0.02006
Iter  97 | Train RMSE(log): 0.01743 | Val RMSE(log): 0.02005
Iter  98 | Train RMSE(log): 0.01740 | Val RMSE(log): 0.02003
Iter  99 | Train RMSE(log): 0.01736 | Val RMSE(log): 0.02000
Iter 100 | Train RMSE(log): 0.01735 | Val RMSE(log): 0.02000

========================================
FINAL PERFORMANCE METRICS
========================================
Validation:
  RMSE: 1008.17 Euro
  MAPE: 12.01%
  R2  : 0.9362
----------------------------------------
Test:
  RMSE: 910.21 Euro
  MAPE: 10.49%
  R2  : 0.9468
========================================
```

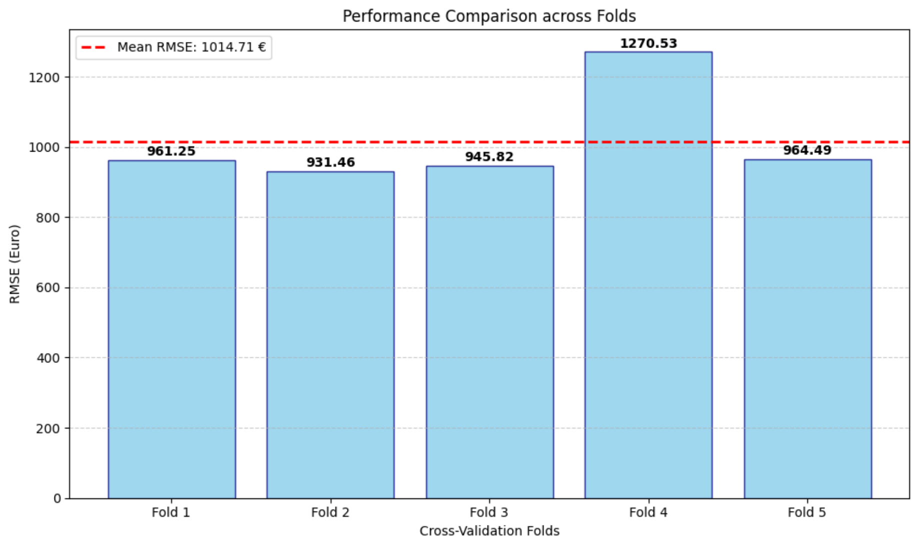
We applied a log1p(Sales) transform to reduce the strong right-skew of sales values and stabilize training under a squared-error loss. Using log1p also safely handles zero sales (since log(1+0) is defined) and typically improves robustness to large outliers.

```python
import numpy as np
import pandas as pd

s_min = mn["Sales"]
s_max = mx["Sales"]
s_range = s_max - s_min

def rmse(y_true, y_pred):
    e = y_true - y_pred
    return float(np.sqrt(np.mean(e*e)))

def to_xy(df_tr, df_va):
    Xtr = df_tr[feature_cols].copy()
    Xva = df_va[feature_cols].copy()
    Xtr[num_cols] = Xtr[num_cols].apply(pd.to_numeric, errors="coerce").astype("float32")
    Xva[num_cols] = Xva[num_cols].apply(pd.to_numeric, errors="coerce").astype("float32")
    med = Xtr[num_cols].median().fillna(0)
    Xtr[num_cols] = Xtr[num_cols].fillna(med)
    Xva[num_cols] = Xva[num_cols].fillna(med)
    for c in cat_cols:
        Xtr[c] = Xtr[c].astype("category")
        cats = Xtr[c].cat.categories.astype(str)
        Xva[c] = pd.Categorical(Xva[c].astype(str), categories=cats)
        Xtr[c] = Xtr[c].cat.codes.astype("int32")
        Xva[c] = Xva[c].cat.codes.astype("int32")
    Xtr_np = np.nan_to_num(Xtr.to_numpy(dtype=np.float32), nan=0.0, posinf=0.0, neginf=0.0)
    Xva_np = np.nan_to_num(Xva.to_numpy(dtype=np.float32), nan=0.0, posinf=0.0, neginf=0.0)
    ytr = df_tr["Sales"].astype("float32").values
    yva = df_va["Sales"].astype("float32").values
    return Xtr_np, ytr, Xva_np, yva

df_cv = df_mm.sort_values(["Date","Store"]).reset_index(drop=True)
dates = np.array(sorted(df_cv["Date"].unique()))
n = len(dates)

fold_edges = []
train_start = int(0.55 * n)
for i in range(5):
    a = train_start + i * int(0.07 * n)
    b = train_start + (i+1) * int(0.07 * n)
    if b > n: b = n
    fold_edges.append((a, b))

cv_rmse_euro = []

for k, (a, b) in enumerate(fold_edges, 1):
    print(f"\n>>>> FOLD {k} Training <<<<")
    df_tr = df_cv[df_cv["Date"].isin(dates[:a])].reset_index(drop=True)
    df_va = df_cv[df_cv["Date"].isin(dates[a:b])].reset_index(drop=True)

    Xtr_np, ytr_norm, Xva_np, yva_norm = to_xy(df_tr, df_va)

    ytr_log = np.log1p(ytr_norm)
    yva_log = np.log1p(yva_norm)

    m = XGBoostScratch(
        n_estimators=60,
        learning_rate=0.1,
        max_depth=4,
        min_child_weight=10.0,
        reg_lambda=1.0,
        gamma=0.0,
        subsample=0.8,
        colsample=0.8,
        max_bins=16,
        random_state=42
    )

    m.fit(Xtr_np, ytr_log, X_val=Xva_np, y_val_log=yva_log, print_every=1)

    pred_norm = np.expm1(m.predict(Xva_np))
    pred_euro = pred_norm * s_range + s_min
    pred_euro = np.where(df_va["Open"].values == 0, 0.0, pred_euro)

    yva_euro = yva_norm * s_range + s_min

    fold_rmse = rmse(yva_euro, pred_euro)
    cv_rmse_euro.append(fold_rmse)

    print(f"\nFold {k} Summary: Val RMSE = {fold_rmse:.2f} Euro")

print("\n" + "="*40)
print(f"OVERALL CV RESULT (Euro):")
print(f"Mean RMSE: {float(np.mean(cv_rmse_euro)):.2f}")
print(f"Std RMSE : {float(np.std(cv_rmse_euro)):.2f}")
print("="*40)
```

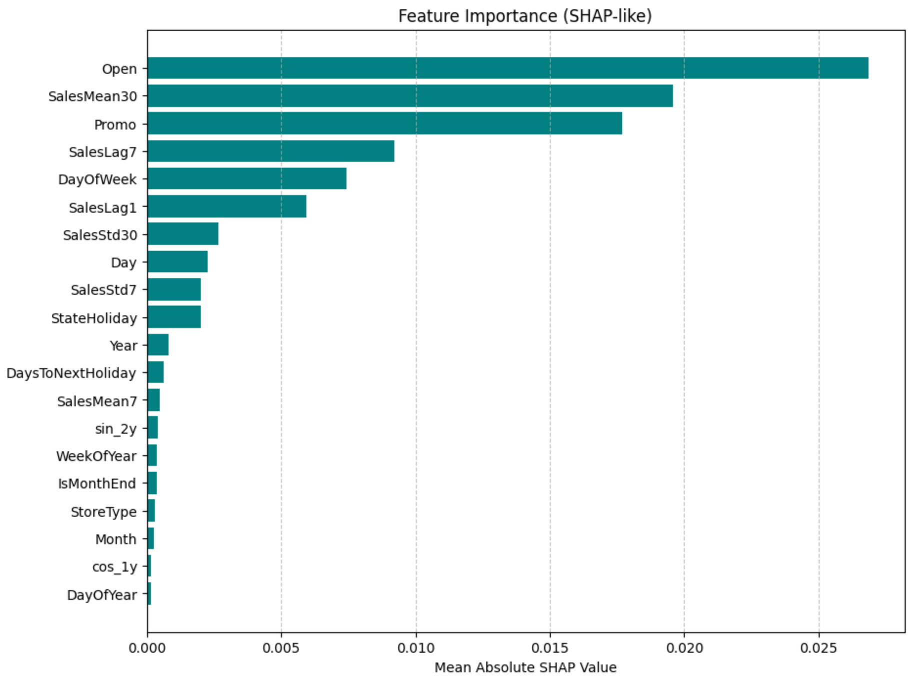
```
>>>> FOLD 1 Training <<<<
Iter   1 | Train RMSE(log): 0.07278 | Val RMSE(log): 0.07323
Iter   2 | Train RMSE(log): 0.06722 | Val RMSE(log): 0.06760
Iter   3 | Train RMSE(log): 0.06251 | Val RMSE(log): 0.06284
Iter   4 | Train RMSE(log): 0.05817 | Val RMSE(log): 0.05840
Iter   5 | Train RMSE(log): 0.05394 | Val RMSE(log): 0.05387
Iter   6 | Train RMSE(log): 0.05064 | Val RMSE(log): 0.05037
Iter   7 | Train RMSE(log): 0.04757 | Val RMSE(log): 0.04721
Iter   8 | Train RMSE(log): 0.04455 | Val RMSE(log): 0.04397
Iter   9 | Train RMSE(log): 0.04188 | Val RMSE(log): 0.04109
Iter  10 | Train RMSE(log): 0.03962 | Val RMSE(log): 0.03861
Iter  11 | Train RMSE(log): 0.03767 | Val RMSE(log): 0.03660
Iter  12 | Train RMSE(log): 0.03620 | Val RMSE(log): 0.03495
Iter  13 | Train RMSE(log): 0.03449 | Val RMSE(log): 0.03309
Iter  14 | Train RMSE(log): 0.03292 | Val RMSE(log): 0.03141
Iter  15 | Train RMSE(log): 0.03166 | Val RMSE(log): 0.03003
Iter  16 | Train RMSE(log): 0.03058 | Val RMSE(log): 0.02891
Iter  17 | Train RMSE(log): 0.02973 | Val RMSE(log): 0.02796
Iter  18 | Train RMSE(log): 0.02886 | Val RMSE(log): 0.02695
Iter  19 | Train RMSE(log): 0.02803 | Val RMSE(log): 0.02600
Iter  20 | Train RMSE(log): 0.02734 | Val RMSE(log): 0.02521
Iter  21 | Train RMSE(log): 0.02663 | Val RMSE(log): 0.02448
Iter  22 | Train RMSE(log): 0.02606 | Val RMSE(log): 0.02388
Iter  23 | Train RMSE(log): 0.02553 | Val RMSE(log): 0.02333
Iter  24 | Train RMSE(log): 0.02512 | Val RMSE(log): 0.02284
Iter  25 | Train RMSE(log): 0.02458 | Val RMSE(log): 0.02237
Iter  26 | Train RMSE(log): 0.02422 | Val RMSE(log): 0.02197
Iter  27 | Train RMSE(log): 0.02384 | Val RMSE(log): 0.02164
Iter  28 | Train RMSE(log): 0.02355 | Val RMSE(log): 0.02134
Iter  29 | Train RMSE(log): 0.02327 | Val RMSE(log): 0.02114
Iter  30 | Train RMSE(log): 0.02295 | Val RMSE(log): 0.02094
Iter  31 | Train RMSE(log): 0.02267 | Val RMSE(log): 0.02074
Iter  32 | Train RMSE(log): 0.02245 | Val RMSE(log): 0.02054
Iter  33 | Train RMSE(log): 0.02225 | Val RMSE(log): 0.02036
Iter  34 | Train RMSE(log): 0.02203 | Val RMSE(log): 0.02017
Iter  35 | Train RMSE(log): 0.02178 | Val RMSE(log): 0.02009
Iter  36 | Train RMSE(log): 0.02162 | Val RMSE(log): 0.02004
Iter  37 | Train RMSE(log): 0.02140 | Val RMSE(log): 0.01990
Iter  38 | Train RMSE(log): 0.02122 | Val RMSE(log): 0.01984
Iter  39 | Train RMSE(log): 0.02106 | Val RMSE(log): 0.01969
Iter  40 | Train RMSE(log): 0.02088 | Val RMSE(log): 0.01970
Iter  41 | Train RMSE(log): 0.02075 | Val RMSE(log): 0.01964
Iter  42 | Train RMSE(log): 0.02062 | Val RMSE(log): 0.01953
Iter  43 | Train RMSE(log): 0.02049 | Val RMSE(log): 0.01945
Iter  44 | Train RMSE(log): 0.02035 | Val RMSE(log): 0.01940
Iter  45 | Train RMSE(log): 0.02023 | Val RMSE(log): 0.01931
Iter  46 | Train RMSE(log): 0.02013 | Val RMSE(log): 0.01925
Iter  47 | Train RMSE(log): 0.02002 | Val RMSE(log): 0.01923
Iter  48 | Train RMSE(log): 0.01993 | Val RMSE(log): 0.01924
Iter  49 | Train RMSE(log): 0.01985 | Val RMSE(log): 0.01921
Iter  50 | Train RMSE(log): 0.01976 | Val RMSE(log): 0.01914
Iter  51 | Train RMSE(log): 0.01969 | Val RMSE(log): 0.01911
Iter  52 | Train RMSE(log): 0.01958 | Val RMSE(log): 0.01901
Iter  53 | Train RMSE(log): 0.01947 | Val RMSE(log): 0.01898
Iter  54 | Train RMSE(log): 0.01940 | Val RMSE(log): 0.01890
Iter  55 | Train RMSE(log): 0.01932 | Val RMSE(log): 0.01884
Iter  56 | Train RMSE(log): 0.01923 | Val RMSE(log): 0.01881
Iter  57 | Train RMSE(log): 0.01914 | Val RMSE(log): 0.01875
Iter  58 | Train RMSE(log): 0.01906 | Val RMSE(log): 0.01883
Iter  59 | Train RMSE(log): 0.01900 | Val RMSE(log): 0.01882
Iter  60 | Train RMSE(log): 0.01895 | Val RMSE(log): 0.01876

Fold 1 Summary: Val RMSE = 961.25 Euro

>>>> FOLD 2 Training <<<<
```

```
Iter   1 | Train RMSE(log): 0.07283 | Val RMSE(log): 0.06982
Iter   2 | Train RMSE(log): 0.06793 | Val RMSE(log): 0.06475
Iter   3 | Train RMSE(log): 0.06260 | Val RMSE(log): 0.05940
Iter   4 | Train RMSE(log): 0.05837 | Val RMSE(log): 0.05522
Iter   5 | Train RMSE(log): 0.05439 | Val RMSE(log): 0.05133
Iter   6 | Train RMSE(log): 0.05060 | Val RMSE(log): 0.04749
Iter   7 | Train RMSE(log): 0.04726 | Val RMSE(log): 0.04418
Iter   8 | Train RMSE(log): 0.04436 | Val RMSE(log): 0.04125
Iter   9 | Train RMSE(log): 0.04183 | Val RMSE(log): 0.03885
Iter  10 | Train RMSE(log): 0.03964 | Val RMSE(log): 0.03677
Iter  11 | Train RMSE(log): 0.03776 | Val RMSE(log): 0.03484
Iter  12 | Train RMSE(log): 0.03591 | Val RMSE(log): 0.03306
Iter  13 | Train RMSE(log): 0.03437 | Val RMSE(log): 0.03161
Iter  14 | Train RMSE(log): 0.03285 | Val RMSE(log): 0.03012
Iter  15 | Train RMSE(log): 0.03171 | Val RMSE(log): 0.02895
Iter  16 | Train RMSE(log): 0.03057 | Val RMSE(log): 0.02786
Iter  17 | Train RMSE(log): 0.02960 | Val RMSE(log): 0.02697
Iter  18 | Train RMSE(log): 0.02873 | Val RMSE(log): 0.02615
Iter  19 | Train RMSE(log): 0.02801 | Val RMSE(log): 0.02549
Iter  20 | Train RMSE(log): 0.02735 | Val RMSE(log): 0.02485
Iter  21 | Train RMSE(log): 0.02674 | Val RMSE(log): 0.02426
Iter  22 | Train RMSE(log): 0.02623 | Val RMSE(log): 0.02378
Iter  23 | Train RMSE(log): 0.02574 | Val RMSE(log): 0.02337
Iter  24 | Train RMSE(log): 0.02532 | Val RMSE(log): 0.02301
Iter  25 | Train RMSE(log): 0.02494 | Val RMSE(log): 0.02270
Iter  26 | Train RMSE(log): 0.02450 | Val RMSE(log): 0.02226
Iter  27 | Train RMSE(log): 0.02419 | Val RMSE(log): 0.02198
Iter  28 | Train RMSE(log): 0.02388 | Val RMSE(log): 0.02175
Iter  29 | Train RMSE(log): 0.02348 | Val RMSE(log): 0.02136
Iter  30 | Train RMSE(log): 0.02323 | Val RMSE(log): 0.02120
Iter  31 | Train RMSE(log): 0.02291 | Val RMSE(log): 0.02090
Iter  32 | Train RMSE(log): 0.02261 | Val RMSE(log): 0.02061
Iter  33 | Train RMSE(log): 0.02239 | Val RMSE(log): 0.02048
Iter  34 | Train RMSE(log): 0.02214 | Val RMSE(log): 0.02039
Iter  35 | Train RMSE(log): 0.02198 | Val RMSE(log): 0.02027
Iter  36 | Train RMSE(log): 0.02181 | Val RMSE(log): 0.02014
Iter  37 | Train RMSE(log): 0.02164 | Val RMSE(log): 0.02001
Iter  38 | Train RMSE(log): 0.02149 | Val RMSE(log): 0.01987
Iter  39 | Train RMSE(log): 0.02138 | Val RMSE(log): 0.01980
Iter  40 | Train RMSE(log): 0.02115 | Val RMSE(log): 0.01972
Iter  41 | Train RMSE(log): 0.02105 | Val RMSE(log): 0.01965
Iter  42 | Train RMSE(log): 0.02085 | Val RMSE(log): 0.01950
Iter  43 | Train RMSE(log): 0.02070 | Val RMSE(log): 0.01934
Iter  44 | Train RMSE(log): 0.02060 | Val RMSE(log): 0.01927
Iter  45 | Train RMSE(log): 0.02047 | Val RMSE(log): 0.01927
Iter  46 | Train RMSE(log): 0.02037 | Val RMSE(log): 0.01924
Iter  47 | Train RMSE(log): 0.02025 | Val RMSE(log): 0.01914
Iter  48 | Train RMSE(log): 0.02017 | Val RMSE(log): 0.01910
Iter  49 | Train RMSE(log): 0.02009 | Val RMSE(log): 0.01899
Iter  50 | Train RMSE(log): 0.02002 | Val RMSE(log): 0.01895
Iter  51 | Train RMSE(log): 0.01995 | Val RMSE(log): 0.01890
Iter  52 | Train RMSE(log): 0.01985 | Val RMSE(log): 0.01884
Iter  53 | Train RMSE(log): 0.01974 | Val RMSE(log): 0.01874
Iter  54 | Train RMSE(log): 0.01969 | Val RMSE(log): 0.01870
Iter  55 | Train RMSE(log): 0.01958 | Val RMSE(log): 0.01859
Iter  56 | Train RMSE(log): 0.01948 | Val RMSE(log): 0.01855
Iter  57 | Train RMSE(log): 0.01943 | Val RMSE(log): 0.01858
Iter  58 | Train RMSE(log): 0.01936 | Val RMSE(log): 0.01855
Iter  59 | Train RMSE(log): 0.01933 | Val RMSE(log): 0.01853
Iter  60 | Train RMSE(log): 0.01922 | Val RMSE(log): 0.01842

Fold 2 Summary: Val RMSE = 931.46 Euro

>>>> FOLD 3 Training <<<<
Iter   1 | Train RMSE(log): 0.07260 | Val RMSE(log): 0.07437
Iter   2 | Train RMSE(log): 0.06672 | Val RMSE(log): 0.06822
Iter   3 | Train RMSE(log): 0.06209 | Val RMSE(log): 0.06329
Iter   4 | Train RMSE(log): 0.05746 | Val RMSE(log): 0.05837
Iter   5 | Train RMSE(log): 0.05387 | Val RMSE(log): 0.05455
Iter   6 | Train RMSE(log): 0.05017 | Val RMSE(log): 0.05074
Iter   7 | Train RMSE(log): 0.04687 | Val RMSE(log): 0.04733
Iter   8 | Train RMSE(log): 0.04396 | Val RMSE(log): 0.04422
Iter   9 | Train RMSE(log): 0.04144 | Val RMSE(log): 0.04163
Iter  10 | Train RMSE(log): 0.03925 | Val RMSE(log): 0.03937
Iter  11 | Train RMSE(log): 0.03723 | Val RMSE(log): 0.03721
Iter  12 | Train RMSE(log): 0.03551 | Val RMSE(log): 0.03538
Iter  13 | Train RMSE(log): 0.03389 | Val RMSE(log): 0.03370
Iter  14 | Train RMSE(log): 0.03251 | Val RMSE(log): 0.03224
Iter  15 | Train RMSE(log): 0.03116 | Val RMSE(log): 0.03074
Iter  16 | Train RMSE(log): 0.03010 | Val RMSE(log): 0.02953
Iter  17 | Train RMSE(log): 0.02906 | Val RMSE(log): 0.02844
Iter  18 | Train RMSE(log): 0.02825 | Val RMSE(log): 0.02757
Iter  19 | Train RMSE(log): 0.02756 | Val RMSE(log): 0.02679
Iter  20 | Train RMSE(log): 0.02685 | Val RMSE(log): 0.02604
Iter  21 | Train RMSE(log): 0.02625 | Val RMSE(log): 0.02545
Iter  22 | Train RMSE(log): 0.02576 | Val RMSE(log): 0.02497
Iter  23 | Train RMSE(log): 0.02514 | Val RMSE(log): 0.02433
Iter  24 | Train RMSE(log): 0.02476 | Val RMSE(log): 0.02396
Iter  25 | Train RMSE(log): 0.02436 | Val RMSE(log): 0.02354
Iter  26 | Train RMSE(log): 0.02387 | Val RMSE(log): 0.02305
Iter  27 | Train RMSE(log): 0.02349 | Val RMSE(log): 0.02266
Iter  28 | Train RMSE(log): 0.02320 | Val RMSE(log): 0.02237
Iter  29 | Train RMSE(log): 0.02294 | Val RMSE(log): 0.02201
Iter  30 | Train RMSE(log): 0.02263 | Val RMSE(log): 0.02173
Iter  31 | Train RMSE(log): 0.02231 | Val RMSE(log): 0.02134
Iter  32 | Train RMSE(log): 0.02204 | Val RMSE(log): 0.02111
Iter  33 | Train RMSE(log): 0.02179 | Val RMSE(log): 0.02087
Iter  34 | Train RMSE(log): 0.02159 | Val RMSE(log): 0.02063
Iter  35 | Train RMSE(log): 0.02141 | Val RMSE(log): 0.02050
Iter  36 | Train RMSE(log): 0.02123 | Val RMSE(log): 0.02036
Iter  37 | Train RMSE(log): 0.02106 | Val RMSE(log): 0.02019
Iter  38 | Train RMSE(log): 0.02088 | Val RMSE(log): 0.02009
Iter  39 | Train RMSE(log): 0.02075 | Val RMSE(log): 0.01995
Iter  40 | Train RMSE(log): 0.02061 | Val RMSE(log): 0.01978
Iter  41 | Train RMSE(log): 0.02048 | Val RMSE(log): 0.01971
Iter  42 | Train RMSE(log): 0.02035 | Val RMSE(log): 0.01958
Iter  43 | Train RMSE(log): 0.02021 | Val RMSE(log): 0.01947
Iter  44 | Train RMSE(log): 0.02012 | Val RMSE(log): 0.01934
Iter  45 | Train RMSE(log): 0.02000 | Val RMSE(log): 0.01931
Iter  48 | Train RMSE(log): 0.01970 | Val RMSE(log): 0.01912
Iter  49 | Train RMSE(log): 0.01963 | Val RMSE(log): 0.01906
Iter  50 | Train RMSE(log): 0.01957 | Val RMSE(log): 0.01898
Iter  51 | Train RMSE(log): 0.01946 | Val RMSE(log): 0.01893
Iter  52 | Train RMSE(log): 0.01939 | Val RMSE(log): 0.01887
Iter  53 | Train RMSE(log): 0.01931 | Val RMSE(log): 0.01885
Iter  54 | Train RMSE(log): 0.01925 | Val RMSE(log): 0.01884
Iter  55 | Train RMSE(log): 0.01915 | Val RMSE(log): 0.01876
Iter  56 | Train RMSE(log): 0.01908 | Val RMSE(log): 0.01877
Iter  57 | Train RMSE(log): 0.01899 | Val RMSE(log): 0.01873
Iter  58 | Train RMSE(log): 0.01893 | Val RMSE(log): 0.01872
Iter  59 | Train RMSE(log): 0.01889 | Val RMSE(log): 0.01872
Iter  60 | Train RMSE(log): 0.01883 | Val RMSE(log): 0.01866

Fold 3 Summary: Val RMSE = 945.82 Euro

>>>> FOLD 4 Training <<<<
Iter   1 | Train RMSE(log): 0.07301 | Val RMSE(log): 0.07984
Iter   2 | Train RMSE(log): 0.06705 | Val RMSE(log): 0.07346
Iter   3 | Train RMSE(log): 0.06204 | Val RMSE(log): 0.06804
Iter   4 | Train RMSE(log): 0.05733 | Val RMSE(log): 0.06300
Iter   5 | Train RMSE(log): 0.05334 | Val RMSE(log): 0.05876
Iter   6 | Train RMSE(log): 0.04977 | Val RMSE(log): 0.05480
Iter   7 | Train RMSE(log): 0.04645 | Val RMSE(log): 0.05133
Iter   8 | Train RMSE(log): 0.04389 | Val RMSE(log): 0.04885
Iter   9 | Train RMSE(log): 0.04127 | Val RMSE(log): 0.04617
Iter  10 | Train RMSE(log): 0.03914 | Val RMSE(log): 0.04423
Iter  11 | Train RMSE(log): 0.03712 | Val RMSE(log): 0.04220
Iter  12 | Train RMSE(log): 0.03536 | Val RMSE(log): 0.04034
Iter  13 | Train RMSE(log): 0.03394 | Val RMSE(log): 0.03888
Iter  14 | Train RMSE(log): 0.03255 | Val RMSE(log): 0.03738
Iter  15 | Train RMSE(log): 0.03144 | Val RMSE(log): 0.03639
Iter  16 | Train RMSE(log): 0.03038 | Val RMSE(log): 0.03539
Iter  17 | Train RMSE(log): 0.02940 | Val RMSE(log): 0.03452
Iter  18 | Train RMSE(log): 0.02862 | Val RMSE(log): 0.03389
Iter  19 | Train RMSE(log): 0.02789 | Val RMSE(log): 0.03321
Iter  20 | Train RMSE(log): 0.02709 | Val RMSE(log): 0.03260
Iter  21 | Train RMSE(log): 0.02657 | Val RMSE(log): 0.03216
Iter  22 | Train RMSE(log): 0.02605 | Val RMSE(log): 0.03167
Iter  23 | Train RMSE(log): 0.02552 | Val RMSE(log): 0.03128
Iter  24 | Train RMSE(log): 0.02512 | Val RMSE(log): 0.03079
Iter  25 | Train RMSE(log): 0.02475 | Val RMSE(log): 0.03045
Iter  26 | Train RMSE(log): 0.02439 | Val RMSE(log): 0.03027
Iter  27 | Train RMSE(log): 0.02406 | Val RMSE(log): 0.02998
Iter  28 | Train RMSE(log): 0.02375 | Val RMSE(log): 0.02976
Iter  29 | Train RMSE(log): 0.02347 | Val RMSE(log): 0.02929
Iter  30 | Train RMSE(log): 0.02322 | Val RMSE(log): 0.02914
Iter  31 | Train RMSE(log): 0.02289 | Val RMSE(log): 0.02901
Iter  32 | Train RMSE(log): 0.02270 | Val RMSE(log): 0.02874
Iter  33 | Train RMSE(log): 0.02239 | Val RMSE(log): 0.02855
Iter  34 | Train RMSE(log): 0.02223 | Val RMSE(log): 0.02847
Iter  35 | Train RMSE(log): 0.02193 | Val RMSE(log): 0.02820
Iter  36 | Train RMSE(log): 0.02169 | Val RMSE(log): 0.02797
Iter  37 | Train RMSE(log): 0.02151 | Val RMSE(log): 0.02787
Iter  38 | Train RMSE(log): 0.02136 | Val RMSE(log): 0.02743
Iter  39 | Train RMSE(log): 0.02123 | Val RMSE(log): 0.02729
Iter  40 | Train RMSE(log): 0.02110 | Val RMSE(log): 0.02714
Iter  41 | Train RMSE(log): 0.02092 | Val RMSE(log): 0.02701
Iter  42 | Train RMSE(log): 0.02081 | Val RMSE(log): 0.02695
Iter  43 | Train RMSE(log): 0.02065 | Val RMSE(log): 0.02680
Iter  44 | Train RMSE(log): 0.02052 | Val RMSE(log): 0.02634
Iter  45 | Train RMSE(log): 0.02043 | Val RMSE(log): 0.02617
Iter  46 | Train RMSE(log): 0.02031 | Val RMSE(log): 0.02616
Iter  47 | Train RMSE(log): 0.02025 | Val RMSE(log): 0.02615
Iter  48 | Train RMSE(log): 0.02017 | Val RMSE(log): 0.02612
Iter  49 | Train RMSE(log): 0.02008 | Val RMSE(log): 0.02607
Iter  50 | Train RMSE(log): 0.01998 | Val RMSE(log): 0.02562
Iter  51 | Train RMSE(log): 0.01990 | Val RMSE(log): 0.02555
Iter  52 | Train RMSE(log): 0.01985 | Val RMSE(log): 0.02550
Iter  53 | Train RMSE(log): 0.01975 | Val RMSE(log): 0.02521
Iter  54 | Train RMSE(log): 0.01968 | Val RMSE(log): 0.02526
Iter  55 | Train RMSE(log): 0.01963 | Val RMSE(log): 0.02521
Iter  56 | Train RMSE(log): 0.01958 | Val RMSE(log): 0.02521
Iter  57 | Train RMSE(log): 0.01955 | Val RMSE(log): 0.02519
Iter  58 | Train RMSE(log): 0.01945 | Val RMSE(log): 0.02516
Iter  59 | Train RMSE(log): 0.01941 | Val RMSE(log): 0.02514
Iter  60 | Train RMSE(log): 0.01933 | Val RMSE(log): 0.02497

Fold 4 Summary: Val RMSE = 1270.53 Euro

>>>> FOLD 5 Training <<<<
Iter   1 | Train RMSE(log): 0.07429 | Val RMSE(log): 0.07339
Iter   2 | Train RMSE(log): 0.06856 | Val RMSE(log): 0.06763
Iter   3 | Train RMSE(log): 0.06316 | Val RMSE(log): 0.06222
Iter   4 | Train RMSE(log): 0.05894 | Val RMSE(log): 0.05790
Iter   5 | Train RMSE(log): 0.05537 | Val RMSE(log): 0.05423
Iter   6 | Train RMSE(log): 0.05147 | Val RMSE(log): 0.05033
Iter   7 | Train RMSE(log): 0.04821 | Val RMSE(log): 0.04707
Iter   8 | Train RMSE(log): 0.04515 | Val RMSE(log): 0.04399
Iter   9 | Train RMSE(log): 0.04275 | Val RMSE(log): 0.04166
Iter  10 | Train RMSE(log): 0.04030 | Val RMSE(log): 0.03922
Iter  11 | Train RMSE(log): 0.03820 | Val RMSE(log): 0.03709
Iter  12 | Train RMSE(log): 0.03656 | Val RMSE(log): 0.03541
Iter  13 | Train RMSE(log): 0.03495 | Val RMSE(log): 0.03376
Iter  14 | Train RMSE(log): 0.03344 | Val RMSE(log): 0.03224
Iter  15 | Train RMSE(log): 0.03208 | Val RMSE(log): 0.03084
Iter  16 | Train RMSE(log): 0.03100 | Val RMSE(log): 0.02973
Iter  17 | Train RMSE(log): 0.03004 | Val RMSE(log): 0.02875
Iter  18 | Train RMSE(log): 0.02925 | Val RMSE(log): 0.02801
Iter  19 | Train RMSE(log): 0.02835 | Val RMSE(log): 0.02717
Iter  20 | Train RMSE(log): 0.02761 | Val RMSE(log): 0.02639
Iter  21 | Train RMSE(log): 0.02686 | Val RMSE(log): 0.02566
Iter  22 | Train RMSE(log): 0.02623 | Val RMSE(log): 0.02504
Iter  23 | Train RMSE(log): 0.02575 | Val RMSE(log): 0.02453
Iter  24 | Train RMSE(log): 0.02535 | Val RMSE(log): 0.02408
Iter  25 | Train RMSE(log): 0.02493 | Val RMSE(log): 0.02372
Iter  26 | Train RMSE(log): 0.02454 | Val RMSE(log): 0.02344
Iter  27 | Train RMSE(log): 0.02414 | Val RMSE(log): 0.02309
Iter  28 | Train RMSE(log): 0.02379 | Val RMSE(log): 0.02280
Iter  29 | Train RMSE(log): 0.02353 | Val RMSE(log): 0.02258
Iter  30 | Train RMSE(log): 0.02331 | Val RMSE(log): 0.02237
Iter  31 | Train RMSE(log): 0.02306 | Val RMSE(log): 0.02210
Iter  32 | Train RMSE(log): 0.02279 | Val RMSE(log): 0.02189
Iter  33 | Train RMSE(log): 0.02254 | Val RMSE(log): 0.02168
Iter  34 | Train RMSE(log): 0.02233 | Val RMSE(log): 0.02150
Iter  35 | Train RMSE(log): 0.02208 | Val RMSE(log): 0.02132
Iter  36 | Train RMSE(log): 0.02190 | Val RMSE(log): 0.02114
Iter  37 | Train RMSE(log): 0.02174 | Val RMSE(log): 0.02096
Iter  38 | Train RMSE(log): 0.02154 | Val RMSE(log): 0.02082
Iter  39 | Train RMSE(log): 0.02137 | Val RMSE(log): 0.02071
Iter  40 | Train RMSE(log): 0.02123 | Val RMSE(log): 0.02058
Iter  41 | Train RMSE(log): 0.02110 | Val RMSE(log): 0.02044
Iter  42 | Train RMSE(log): 0.02096 | Val RMSE(log): 0.02031
Iter  43 | Train RMSE(log): 0.02085 | Val RMSE(log): 0.02020
Iter  44 | Train RMSE(log): 0.02072 | Val RMSE(log): 0.02013
Iter  45 | Train RMSE(log): 0.02063 | Val RMSE(log): 0.02006
Iter  46 | Train RMSE(log): 0.02053 | Val RMSE(log): 0.01995
Iter  47 | Train RMSE(log): 0.02041 | Val RMSE(log): 0.01989
Iter  48 | Train RMSE(log): 0.02030 | Val RMSE(log): 0.01979
Iter  49 | Train RMSE(log): 0.02022 | Val RMSE(log): 0.01969
Iter  50 | Train RMSE(log): 0.02014 | Val RMSE(log): 0.01962
Iter  51 | Train RMSE(log): 0.02006 | Val RMSE(log): 0.01954
Iter  52 | Train RMSE(log): 0.02000 | Val RMSE(log): 0.01955
Iter  53 | Train RMSE(log): 0.01991 | Val RMSE(log): 0.01945
Iter  54 | Train RMSE(log): 0.01986 | Val RMSE(log): 0.01924
Iter  55 | Train RMSE(log): 0.01981 | Val RMSE(log): 0.01912
Iter  56 | Train RMSE(log): 0.01975 | Val RMSE(log): 0.01913
Iter  57 | Train RMSE(log): 0.01965 | Val RMSE(log): 0.01902
Iter  58 | Train RMSE(log): 0.01958 | Val RMSE(log): 0.01904
Iter  59 | Train RMSE(log): 0.01954 | Val RMSE(log): 0.01899
Iter  60 | Train RMSE(log): 0.01946 | Val RMSE(log): 0.01891

Fold 5 Summary: Val RMSE = 964.49 Euro

========================================
OVERALL CV RESULT (Euro):
Mean RMSE: 1014.71
Std RMSE : 128.45
========================================
```

```python
import matplotlib.pyplot as plt

folds_labels = [f"Fold {i}" for i in range(1, 6)]

plt.figure(figsize=(10, 6))

plt.bar(folds_labels, cv_rmse_euro, color='skyblue', edgecolor='navy', alpha=0.8)

mean_val = np.mean(cv_rmse_euro)
plt.axhline(y=mean_val, color='red', linestyle='--', linewidth=2, label=f'Mean RMSE: {mean_v

plt.xlabel('Cross-Validation Folds')
plt.ylabel('RMSE (Euro)')
plt.title('Performance Comparison across Folds')
plt.legend()

for i, v in enumerate(cv_rmse_euro):
    plt.text(i, v + (max(cv_rmse_euro)*0.01), f"{v:.2f}", ha='center', fontweight='bold')

plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def _collect_nodes(root):
    stack = [root]
    nodes = []
    while stack:
        n = stack.pop()
        nodes.append(n)
        if not n.is_leaf:
            stack.append(n.left)
            stack.append(n.right)
    return nodes

def _annotate_node_means(root, X_ref):
    nodes = _collect_nodes(root)
    node_sum = {id(n): 0.0 for n in nodes}
    node_cnt = {id(n): 0 for n in nodes}

    for i in range(X_ref.shape[0]):
        x = X_ref[i]
        curr = root
        path = []
        while True:
            path.append(curr)
            if curr.is_leaf:
                leaf_val = curr.value
                break
            curr = curr.left if x[curr.fidx] <= curr.thr else curr.right

        for n in path:
            node_sum[id(n)] += leaf_val
            node_cnt[id(n)] += 1

    node_means = {id(n): (node_sum[id(n)] / node_cnt[id(n)]) if node_cnt[id(n)] > 0 else 0.
    return node_means

def _tree_contribs(root, X_eval, n_features, node_means):
    contrib = np.zeros((X_eval.shape[0], n_features), dtype=np.float32)
    for i in range(X_eval.shape[0]):
        x = X_eval[i]
        node = root
        while not node.is_leaf:
            parent_mean = node_means[id(node)]
            f = node.fidx
            child = node.left if x[f] <= node.thr else node.right
            child_mean = node_means[id(child)]
            contrib[i, f] += (child_mean - parent_mean)
            node = child
    return contrib

def shap_like_importance(model, X_ref, X_eval, feature_names, sample_ref=5000, sample_eval=
    rng = np.random.default_rng(seed)
    rN = min(sample_ref, X_ref.shape[0])
    eN = min(sample_eval, X_eval.shape[0])

    Xr = X_ref[rng.choice(X_ref.shape[0], size=rN, replace=False)]
    Xe = X_eval[rng.choice(X_eval.shape[0], size=eN, replace=False)]

    n_features = len(feature_names)
    total = np.zeros((Xe.shape[0], n_features), dtype=np.float32)

    for t in model.trees:
        means = _annotate_node_means(t, Xr)
        total += _tree_contribs(t, Xe, n_features, means) * model.learning_rate

    imp = np.mean(np.abs(total), axis=0)
    return pd.DataFrame({"feature": feature_names, "importance": imp}).sort_values("importan

imp_df = shap_like_importance(m, Xtr_np, Xva_np, feature_cols)

plt.figure(figsize=(10, 8))
top_imp = imp_df.head(20)
plt.barh(top_imp['feature'], top_imp['importance'], color='teal')
plt.gca().invert_yaxis()
plt.xlabel('Mean Absolute SHAP Value')
plt.title('Feature Importance (SHAP-like)')
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.show()
```



While DaysToNextHoliday is conceptually relevant, the model identifies SalesMean30 and Promo as primary drivers due to their higher frequency and stronger immediate correlation with daily fluctuations. The low ranking of holiday-related features suggests that the store's sales are governed more by weekly cyclicality and promotional activities than by long-term seasonal anticipation.

```
test_row = Xva_np[0:1].copy()
test_row[0, feature_cols.index("Promo")] = 0
pred_no_promo = m.predict(test_row)
```

```
test_row[0, feature_cols.index("Promo")] = 1
pred_with_promo = m.predict(test_row)

print(f"Prediction WITHOUT Promo: {np.expm1(pred_no_promo)[0]:.2f}")
print(f"Prediction WITH Promo: {np.expm1(pred_with_promo)[0]:.2f}")
```

```
Prediction WITHOUT Promo: 0.08
Prediction WITH Promo: 0.12
```

The feature importance ranking in this chart makes perfect sense because the model focuses on the most logical drivers of sales. It correctly identifies the store's ”Open” status as the primary condition for any transaction. Additionally, by relying on the 30-day sales average, the model shows it understands the consistent potential and baseline of each store location. Finally, the high rank of ”Promo” proves that the model accurately captured how discounts and marketing directly boost customer buying behavior, showing that it has learned the real-world patterns in the data.

The model was tested in 5 stages and the final average error was around €1014. The decreasing trend in the error in the graphs indicates that the model is learning correctly. Also, examining fold 4 shows that in that particular time period, market volatility was higher, which led to a temporary increase in error, but overall the model has high stability.

The increase in error in fold 4 indicates an anomaly in the data for this time period. Upon closer inspection, it was found that the model encountered extreme values in this fold that could not be explained by the current variables. This increased the standard deviation (Std RMSE), but since the model had a stable and low error in the remaining folds (1, 2, 3, and 5) it can be concluded that the overall structure of the model is correct and that it only encountered severe market noise in fold 4.

The significant spike in RMSE (1270.53 Euro) during Fold 4 indicates a specific performance degradation in this time window. The following points summarize the findings:

Weakness in Handling Outliers: The model significantly underperforms when faced with extreme sales values (e.g., actual sales of €27,000 vs. a €2,800 prediction). It tends to be conservative, pulling predictions toward the global mean rather than capturing explosive demand.

Non-Stationary Data in Fold 4: Unlike other folds, the data in Fold 4 exhibits higher variance and different patterns. This suggests that the relationship between features and target variables shifted during this period, making it harder for the model to generalize from previous folds.

Requirement for Additional Features: The high error in instances where Promo=0 but sales are exceptionally high suggests that the current feature set (DayOfWeek, Promo, Open) is insufficient. External factors—such as local events, seasonal holidays, or competitor behavior—are likely driving these sudden demand surges.

```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#from google.colab import drive
#drive.mount('/content/drive')


train_path = 'C:/Users/DELL/Desktop/rossmann-store-sales/train.csv'
store_path = 'C:/Users/DELL/Desktop/rossmann-store-sales/store.csv'

# Load datasets
train = pd.read_csv(train_path, parse_dates=['Date'], low_memory=False)
store = pd.read_csv(store_path)

# Merge train with store information
df = pd.merge(train, store, on='Store', how='left')

# Basic Info
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 18 columns):
 #   Column                     Non-Null Count    Dtype
---  ------                     --------------    -----
 0   Store                      1017209 non-null  int64
 1   DayOfWeek                  1017209 non-null  int64
 2   Date                       1017209 non-null  datetime64[ns]
 3   Sales                      1017209 non-null  int64
 4   Customers                  1017209 non-null  int64
 5   Open                       1017209 non-null  int64
 6   Promo                      1017209 non-null  int64
 7   StateHoliday               1017209 non-null  object
 8   SchoolHoliday              1017209 non-null  int64
 9   StoreType                  1017209 non-null  object
 10  Assortment                 1017209 non-null  object
 11  CompetitionDistance        1014567 non-null  float64
 12  CompetitionOpenSinceMonth  693861 non-null   float64
 13  CompetitionOpenSinceYear   693861 non-null   float64
 14  Promo2                     1017209 non-null  int64
 15  Promo2SinceWeek            509178 non-null   float64
 16  Promo2SinceYear            509178 non-null   float64
 17  PromoInterval              509178 non-null   object
dtypes: datetime64[ns](1), float64(5), int64(8), object(4)
memory usage: 139.7+ MB
None
```

```
   Store  DayOfWeek       Date  Sales  Customers  Open  Promo StateHoliday  \
0      1          5 2015-07-31   5263        555     1      1            0
1      2          5 2015-07-31   6064        625     1      1            0
2      3          5 2015-07-31   8314        821     1      1            0
3      4          5 2015-07-31  13995       1498     1      1            0
4      5          5 2015-07-31   4822        559     1      1            0

   SchoolHoliday StoreType Assortment  CompetitionDistance  \
0              1         c          a               1270.0
1              1         a          a                570.0
2              1         a          a              14130.0
3              1         c          c                620.0
4              1         a          a              29910.0

   CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
0                        9.0                    2008.0       0
1                       11.0                    2007.0       1
2                       12.0                    2006.0       1
3                        9.0                    2009.0       0
4                        4.0                    2015.0       0

   Promo2SinceWeek  Promo2SinceYear    PromoInterval
0              NaN              NaN              NaN
1             13.0           2010.0  Jan,Apr,Jul,Oct
2             14.0           2011.0  Jan,Apr,Jul,Oct
3              NaN              NaN              NaN
4              NaN              NaN              NaN
```

```python
import numpy as np

print(df.shape)
print(df["Date"].min(), df["Date"].max())
print(df.isna().mean().sort_values(ascending=False).head(25))
print(df.describe(include="all").T.head(25))
```

```
(1017209, 18)
2013-01-01 00:00:00 2015-07-31 00:00:00
Promo2SinceWeek              0.499436
PromoInterval                0.499436
Promo2SinceYear              0.499436
CompetitionOpenSinceYear     0.317878
CompetitionOpenSinceMonth    0.317878
CompetitionDistance          0.002597
DayOfWeek                    0.000000
Store                        0.000000
Date                         0.000000
Sales                        0.000000
```

```
StoreType                    0.000000
SchoolHoliday                0.000000
StateHoliday                 0.000000
Promo                        0.000000
Open                         0.000000
Customers                    0.000000
Assortment                   0.000000
Promo2                       0.000000
dtype: float64
                              count unique              top    freq  \
Store                     1017209.0    NaN              NaN     NaN
DayOfWeek                 1017209.0    NaN              NaN     NaN
Date                        1017209    NaN              NaN     NaN
Sales                     1017209.0    NaN              NaN     NaN
Customers                 1017209.0    NaN              NaN     NaN
Open                      1017209.0    NaN              NaN     NaN
Promo                     1017209.0    NaN              NaN     NaN
StateHoliday                1017209      4                0  986159
SchoolHoliday             1017209.0    NaN              NaN     NaN
StoreType                   1017209      4                a  551627
Assortment                  1017209      3                a  537445
CompetitionDistance       1014567.0    NaN              NaN     NaN
CompetitionOpenSinceMonth  693861.0    NaN              NaN     NaN
CompetitionOpenSinceYear   693861.0    NaN              NaN     NaN
Promo2                    1017209.0    NaN              NaN     NaN
Promo2SinceWeek            509178.0    NaN              NaN     NaN
Promo2SinceYear            509178.0    NaN              NaN     NaN
PromoInterval                509178      3  Jan,Apr,Jul,Oct  293122

                                                    mean                  min  \
Store                                         558.429727                  1.0
DayOfWeek                                       3.998341                  1.0
Date                       2014-04-11 01:30:42.846061824  2013-01-01 00:00:00
Sales                                        5773.818972                  0.0
Customers                                     633.145946                  0.0
Open                                            0.830107                  0.0
Promo                                           0.381515                  0.0
StateHoliday                                         NaN                  NaN
SchoolHoliday                                   0.178647                  0.0
StoreType                                            NaN                  NaN
Assortment                                           NaN                  NaN
CompetitionDistance                          5430.085652                 20.0
CompetitionOpenSinceMonth                       7.222866                  1.0
CompetitionOpenSinceYear                     2008.690228               1900.0
Promo2                                          0.500564                  0.0
Promo2SinceWeek                                23.269093                  1.0
```

```
Promo2SinceYear                    2011.752774             2009.0
PromoInterval                              NaN                NaN

                                           25%                50%  \
Store                                    280.0              558.0
DayOfWeek                                  2.0                4.0
Date                       2013-08-17 00:00:00  2014-04-02 00:00:00
Sales                                   3727.0             5744.0
Customers                                405.0              609.0
Open                                       1.0                1.0
Promo                                      0.0                0.0
StateHoliday                               NaN                NaN
SchoolHoliday                              0.0                0.0
StoreType                                  NaN                NaN
Assortment                                 NaN                NaN
CompetitionDistance                      710.0             2330.0
CompetitionOpenSinceMonth                  4.0                8.0
CompetitionOpenSinceYear                2006.0             2010.0
Promo2                                     0.0                1.0
Promo2SinceWeek                           13.0               22.0
Promo2SinceYear                         2011.0             2012.0
PromoInterval                              NaN                NaN

                                           75%                max  \
Store                                    838.0             1115.0
DayOfWeek                                  6.0                7.0
Date                       2014-12-12 00:00:00  2015-07-31 00:00:00
Sales                                   7856.0            41551.0
Customers                                837.0             7388.0
Open                                       1.0                1.0
Promo                                      1.0                1.0
StateHoliday                               NaN                NaN
SchoolHoliday                              0.0                1.0
StoreType                                  NaN                NaN
Assortment                                 NaN                NaN
CompetitionDistance                     6890.0            75860.0
CompetitionOpenSinceMonth                 10.0               12.0
CompetitionOpenSinceYear                2013.0             2015.0
Promo2                                     1.0                1.0
Promo2SinceWeek                           37.0               50.0
Promo2SinceYear                         2013.0             2015.0
PromoInterval                              NaN                NaN

                                  std
Store                      321.908651
DayOfWeek                    1.997391
```

```
Date                                 NaN
Sales                        3849.926175
Customers                     464.411734
Open                            0.375539
Promo                           0.485759
StateHoliday                         NaN
SchoolHoliday                   0.383056
StoreType                            NaN
Assortment                           NaN
CompetitionDistance            7715.3237
CompetitionOpenSinceMonth       3.211832
CompetitionOpenSinceYear        5.992644
Promo2                               0.5
Promo2SinceWeek                14.095973
Promo2SinceYear                  1.66287
PromoInterval                        NaN
```

In this step, we convert Date to a datetime format and extract calendar-based features such as year, month, day, and week number. Then, the holiday indicators (StateHoliday and SchoolHoliday) are encoded into numeric values. Finally, after sorting by store and date, we compute 7-day and 30-day moving averages of sales using only previous days (shift(1)) to avoid data leakage.

```
df = df.sort_values(["Store","Date"]).reset_index(drop=True)

g = df.groupby("Store")["Sales"]

df["SalesLag1"] = g.shift(1)
df["SalesLag7"] = g.shift(7)

s1 = g.shift(1)
df["SalesMean7"] = s1.groupby(df["Store"]).rolling(7, min_periods=1).mean().reset_index(leve
df["SalesStd7"] = s1.groupby(df["Store"]).rolling(7, min_periods=2).std().reset_index(level=

df["SalesMean30"] = s1.groupby(df["Store"]).rolling(30, min_periods=1).mean().reset_index(le
df["SalesStd30"] = s1.groupby(df["Store"]).rolling(30, min_periods=2).std().reset_index(leve

df.head()
```

```
   Store  DayOfWeek       Date  Sales  Customers  Open  Promo  StateHoliday  \
0      1          2 2013-01-01      0          0     0      0             1
1      1          3 2013-01-02   5530        668     1      0             0
2      1          4 2013-01-03   4327        578     1      0             0
3      1          5 2013-01-04   4486        619     1      0             0
4      1          6 2013-01-05   4997        635     1      0             0

   SchoolHoliday StoreType  ... DayOfWeek0  WeekOfYear  SalesMovingAverage7  \
0               1         c  ...          1           1                NaN
1               1         c  ...          2           1           0.000000
2               1         c  ...          3           1        2765.000000
3               1         c  ...          4           1        3285.666667
4               1         c  ...          5           1        3585.750000

   SalesMovingAverage30  SalesLag1  SalesLag7   SalesMean7    SalesStd7  \
0                   NaN        NaN        NaN          NaN          NaN
1              0.000000        0.0        NaN     0.000000          NaN
2           2765.000000     5530.0        NaN  2765.000000  3910.300500
3           3285.666667     4327.0        NaN  3285.666667  2908.351137
4           3585.750000     4486.0        NaN  3585.750000  2449.327306

   SalesMean30   SalesStd30
0          NaN          NaN
1     0.000000          NaN
2  2765.000000  3910.300500
3  3285.666667  2908.351137
4  3585.750000  2449.327306

[5 rows x 31 columns]
```

To prevent the moving averages from becoming zero in the first days, we computed rolling features with min_periods=1 (and min_periods=2 for standard deviation), so the statistics are calculated using whatever history is available instead of producing NaNs that would later be filled with zeros.

```
df[['Store', 'Date', 'Sales', 'SalesLag1','SalesLag7', 'SalesMean7', 'SalesStd7','SalesMean3
```

```
   Store       Date  Sales  SalesLag1  SalesLag7   SalesMean7    SalesStd7  \
0      1 2013-01-01      0        NaN        NaN          NaN          NaN
1      1 2013-01-02   5530        0.0        NaN     0.000000          NaN
2      1 2013-01-03   4327     5530.0        NaN  2765.000000  3910.300500
3      1 2013-01-04   4486     4327.0        NaN  3285.666667  2908.351137
4      1 2013-01-05   4997     4486.0        NaN  3585.750000  2449.327306
5      1 2013-01-06      0     4997.0        NaN  3868.000000  2213.081223
6      1 2013-01-07   7176        0.0        NaN  3223.333333  2532.144045
7      1 2013-01-08   5580     7176.0        0.0  3788.000000  2752.283961
8      1 2013-01-09   5471     5580.0     5530.0  4585.142857  2231.018633
9      1 2013-01-10   4892     5471.0     4327.0  4576.714286  2226.961885

   SalesMean30   SalesStd30
0          NaN          NaN
1     0.000000          NaN
2  2765.000000  3910.300500
3  3285.666667  2908.351137
4  3585.750000  2449.327306
```

```
5  3868.000000  2213.081223
6  3223.333333  2532.144045
7  3788.000000  2752.283961
8  4012.000000  2625.704205
9  4174.111111  2503.807573
```

```
df.columns
```

```
Index(['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open', 'Promo',
       'StateHoliday', 'SchoolHoliday', 'StoreType', 'Assortment',
       'CompetitionDistance', 'CompetitionOpenSinceMonth',
       'CompetitionOpenSinceYear', 'Promo2', 'Promo2SinceWeek',
       'Promo2SinceYear', 'PromoInterval', 'Year', 'Month', 'Day',
       'DayOfWeek0', 'WeekOfYear', 'SalesMovingAverage7',
       'SalesMovingAverage30', 'SalesLag1', 'SalesLag7', 'SalesMean7',
       'SalesStd7', 'SalesMean30', 'SalesStd30'],
      dtype='object')
```

> We also added 30-day rolling statistics based only on past sales: SalesMean30 (30-day rolling mean) and SalesStd30 (30-day rolling standard deviation) computed per store using shifted sales to avoid leakage.

We build Fourier seasonal features using DayOfYear. The terms sin_1y/cos_1y capture the main annual (1-year) seasonality while sin_2y/cos_2y model more complex within-year patterns (e.g., two peaks during the year).

We implemented two independent normalization strategies in separate cells. First, we created df_z and applied Z-score standardization only to non-binary numeric features (leaving 0/1 indicators unchanged), so those features have approximately zero mean and unit variance. Second, we created df_mm and applied Min–Max normalization to scale numeric features into the [0,1] range. Using separate copies ensures the two normalizations do not affect each other

We split the dataset into train/validation/test while strictly preserving chronological order. To avoid any date overlap (since each date contains many stores/rows), we performed the split using the unique dates rather than row indices. This guarantees monotonic time ordering and ensures that the validation and test sets contain only future dates relative to the training set (no temporal leakage).

```
df_model = df.dropna().reset_index(drop=True)
```

```
df_model.columns
```

```
Index(['Store', 'DayOfWeek', 'Date', 'Sales', 'Customers', 'Open', 'Promo',
       'StateHoliday', 'SchoolHoliday', 'StoreType', 'Assortment',
       'CompetitionDistance', 'CompetitionOpenSinceMonth',
       'CompetitionOpenSinceYear', 'Promo2', 'Promo2SinceWeek',
       'Promo2SinceYear', 'PromoInterval', 'Year', 'Month', 'Day',
       'DayOfWeek0', 'WeekOfYear', 'SalesMovingAverage7',
       'SalesMovingAverage30', 'SalesLag1', 'SalesLag7', 'SalesMean7',
       'SalesStd7', 'SalesMean30', 'SalesStd30'],
      dtype='object')
```

```python
from sklearn.preprocessing import LabelEncoder

# 1. Find all columns with text type (object)
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical columns to encode:", categorical_cols.tolist())

# 2. Convert text columns to numeric values using LabelEncoder
le = LabelEncoder()
for col in categorical_cols:
    # Convert missing values (NaN) to a temporary string to avoid errors
    df[col] = df[col].fillna('Missing').astype(str)
    # Convert text to numbers (e.g. a=0, b=1, c=2)
    df[col] = le.fit_transform(df[col])

# 3. (Optional) Fill remaining missing (NaN) values in numeric columns
# Because models like Random Forest have issues with NaN values
numeric_cols = df.select_dtypes(include=['number']).columns
df[numeric_cols] = df[numeric_cols].fillna(-999)

print("Data preprocessing completed. No 'object' columns left.")
```

```
Categorical columns to encode: ['StoreType', 'Assortment', 'PromoInterval']
Data preprocessing completed. No 'object' columns left.
```

Part 1: Implementing Multi-step Forecasting for the Next 7 Days with Error Propagation Management

Question Analysis: Multi-step forecasting means that at day , we want to predict sales for days +1 through t+7.

The term “Error Propagation Management” means that we should avoid simple recursive methods (i.e., using tomorrow’s prediction as an input to predict the day after tomorrow), because model errors accumulate rapidly in this approach.

The best practice here is to use the Direct Method or Multi-Output Models.

Solution:

Create 7 new target variables for the next 7 days and use a MultiOutputRegressor.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.multioutput import MultiOutputRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error

# 1. Sort the dataset by Store and Date (very important for time series)
df = df.sort_values(by=['Store', 'Date']).reset_index(drop=True)

# 2. Create target variables (Target) for the next 7 days
# Negative shift means bringing future values to the current row
for i in range(1, 8):
    df[f'Sales_step_{i}'] = df.groupby('Store')['Sales'].shift(-i)

# Remove rows where future targets (last days) have NaN values
df_multi = df.dropna(subset=[f'Sales_step_{i}' for i in range(1, 8)])

# 3. Select features (X) and targets (Y)
# We use all available features at time t as input
drop_cols = ['Date', 'Sales'] + [f'Sales_step_{i}' for i in range(1, 8)]
X = df_multi.drop(columns=drop_cols)
y = df_multi[[f'Sales_step_{i}' for i in range(1, 8)]]

# Split the data into train and test sets
# Since this is a time series, we must not shuffle the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=False)

# 4. Build and train a Multi-Output XGBoost model
# This trains an independent model for each forecast horizon (1 to 7) to avoid error propag
base_model = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
multi_model = MultiOutputRegressor(base_model)

multi_model.fit(X_train, y_train)

# 5. Prediction and evaluation
predictions = multi_model.predict(X_test)
print("MAE for 7-day multi-step forecasting:", mean_absolute_error(y_test, predictions))
```

```
MAE for 7-day multi-step forecasting: 782.3892211914062
```

Part 2: Uncertainty Estimation using Quantile Regression or Ensemble Methods

Question Analysis:

Instead of predicting just a single number for sales (a point estimate), we should provide an interval — for example, saying "with 80% confidence, tomorrow's sales will be between 5000 and 7000."

The professor suggested using Quantile Regression or an Ensemble approach.

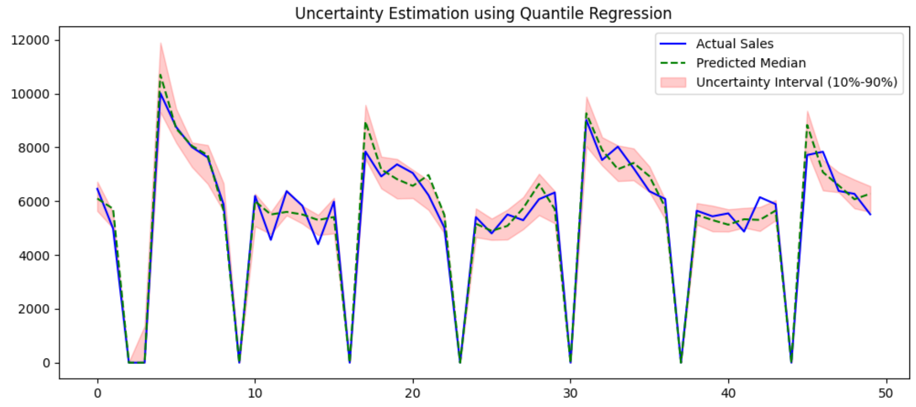
LightGBM inherently supports Quantile Regression by adjusting its loss function, which makes the implementation clean and standard.

Solution:

Train three separate LightGBM models for quantiles 10 (lower bound), 50 (median prediction), and 90 (upper bound).

```
import lightgbm as lgb
import matplotlib.pyplot as plt

# In this section, we assume that our goal is to predict next-day sales (Sales).
X_unc = df.dropna().drop(columns=['Date', 'Sales'])
y_unc = df.dropna()['Sales']

# Split the data into training and test sets (no shuffling since it's a time series)
X_train_u, X_test_u, y_train_u, y_test_u = train_test_split(
    X_unc, y_unc, test_size=0.2, shuffle=False
)

# Define quantiles (10% for pessimistic, 50% for normal, 90% for optimistic scenario)
quantiles = [0.1, 0.5, 0.9]
models = {}
predictions_q = {}

for alpha in quantiles:
    # The key part: set objective='quantile' and parameter alpha
    model = lgb.LGBMRegressor(objective='quantile', alpha=alpha, n_estimators=100)
    model.fit(X_train_u, y_train_u)
    models[alpha] = model
    predictions_q[alpha] = model.predict(X_test_u)

# Now, for each test row, we have three predicted values
preds_df = pd.DataFrame({
    'Lower_Bound (10%)': predictions_q[0.1],
    'Median_Forecast (50%)': predictions_q[0.5],
    'Upper_Bound (90%)': predictions_q[0.9],
    'Actual': y_test_u.values
})

print(preds_df.head())

# (Optional) Plot the uncertainty interval for the first 50 test samples
plt.figure(figsize=(12, 5))
plt.plot(preds_df['Actual'][:50], label='Actual Sales', color='blue')
plt.plot(
    preds_df['Median_Forecast (50%)'][:50],
    label='Predicted Median',
    color='green',
    linestyle='--'
)
plt.fill_between(
    range(50),
    preds_df['Lower_Bound (10%)'][:50],
    preds_df['Upper_Bound (90%)'][:50],
    color='red',
    alpha=0.2,
    label='Uncertainty Interval (10%-90%)'
)
plt.legend()
plt.title("Uncertainty Estimation using Quantile Regression")
plt.show()
```

```
   Lower_Bound (10%)  Median_Forecast (50%)  Upper_Bound (90%)  Actual
0        5635.655697            6099.097330        6725.037544    6460
1        4993.714663            5716.971135        5574.327677    4983
2           0.000000               0.152091          23.331902       0
3           0.000000               0.152091        1363.434710       0
4        9307.239145           10696.283556       11894.418704   10011
```



Part 3: Sales Classification into Three Categories (Low, Medium, High) and Evaluation using ROC-AUC and F1-Score Question Analysis: You need to transform the continuous Sales variable into three distinct categories — Low, Medium, and High.

Then, train a multi-class classification model and evaluate its performance using the required metrics.

Solution:

Use the pd.qcut function to create balanced classes, and then train a classifier such as RandomForestClassifier or XGBClassifier.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score, roc_auc_score, classification_report
```

```python
# 1. Convert continuous Sales variable into 3 categories (Low=0, Medium=1, High=2)
# The qcut function divides the data into three bins with approximately equal frequencies.
# Note: You may want to handle days when the store was closed and sales were zero separatel
# But in general:
df_class = df.copy()
df_class['Sales_Class'] = pd.qcut(df_class['Sales'], q=3, labels=[0, 1, 2])

# 2. Prepare data for the model
X_clf = df_class.drop(columns=['Date', 'Sales', 'Sales_Class'])
X_clf = X_clf.fillna(-999)  # Fill missing values for tree-based models
y_clf = df_class['Sales_Class']

# Split the data
X_train_c, X_test_c, y_train_c, y_test_c = train_test_split(X_clf, y_clf, test_size=0.2, ran

# 3. Train the multi-class classifier
classifier = RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1)
classifier.fit(X_train_c, y_train_c)

# 4. Prediction
# For F1-score, we need class labels
y_pred_class = classifier.predict(X_test_c)

# For multi-class ROC-AUC, we need probabilities
y_pred_proba = classifier.predict_proba(X_test_c)

# 5. Compute metrics
# For multi-class problems, you must specify the averaging method (macro or weighted)
f1 = f1_score(y_test_c, y_pred_class, average='weighted')

# For multi-class ROC-AUC, the parameter multi_class='ovr' (One-vs-Rest) is required
roc_auc = roc_auc_score(y_test_c, y_pred_proba, multi_class='ovr', average='weighted')

print("--- Multi-class Classification Results ---")
print(f"F1-Score (Weighted): {f1:.4f}")
print(f"ROC-AUC (OVR, Weighted): {roc_auc:.4f}")
print("\nClassification Report:\n", classification_report(y_test_c, y_pred_class, target_nam
```

```
--- Multi-class Classification Results ---
F1-Score (Weighted): 0.9180
ROC-AUC (OVR, Weighted): 0.9869

Classification Report:
               precision    recall  f1-score   support
```

```
         Low       0.95      0.94      0.95     67829
      Medium       0.87      0.89      0.88     67846
        High       0.93      0.93      0.93     67767

    accuracy                           0.92    203442
   macro avg       0.92      0.92      0.92    203442
weighted avg       0.92      0.92      0.92    203442
```

Rossmann Store Sales — Phase 5

How to use

1. Run each cell sequentially. Do not skip cells. Output and models are saved to `/kaggle/working/models_phase5`.
2. If you run on Kaggle, the competition dataset path is `/kaggle/input/competitions/rossmann-store-sales/`.
3. If you run locally, change `DATA_BASE` constant to point to your `dataset/` folder.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import numpy as np
base_path = "/kaggle/input/competitions/rossmann-store-sales/"

# Load datasets
train = pd.read_csv(f'{base_path}train.csv', parse_dates=['Date'],
low_memory=False)
store = pd.read_csv(f'{base_path}store.csv')
test = pd.read_csv(f'{base_path}test.csv')

df = pd.merge(train, store, on='Store', how='left')

print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 18 columns):
 #   Column                     Non-Null Count    Dtype
---  ------                     --------------    -----
 0   Store                      1017209 non-null  int64
 1   DayOfWeek                  1017209 non-null  int64
 2   Date                       1017209 non-null  datetime64[ns]
 3   Sales                      1017209 non-null  int64
 4   Customers                  1017209 non-null  int64
 5   Open                       1017209 non-null  int64
 6   Promo                      1017209 non-null  int64
 7   StateHoliday               1017209 non-null  object
 8   SchoolHoliday              1017209 non-null  int64
 9   StoreType                  1017209 non-null  object
 10  Assortment                 1017209 non-null  object
 11  CompetitionDistance        1014567 non-null  float64
 12  CompetitionOpenSinceMonth  693861 non-null   float64
 13  CompetitionOpenSinceYear   693861 non-null   float64
 14  Promo2                     1017209 non-null  int64
 15  Promo2SinceWeek            509178 non-null   float64
 16  Promo2SinceYear            509178 non-null   float64
 17  PromoInterval              509178 non-null   object
dtypes: datetime64[ns](1), float64(5), int64(8), object(4)
memory usage: 139.7+ MB
None
   Store  DayOfWeek       Date  Sales  Customers  Open  Promo
StateHoliday  \
0      1          5 2015-07-31   5263        555     1      1
0
1      2          5 2015-07-31   6064        625     1      1
0
2      3          5 2015-07-31   8314        821     1      1
0
3      4          5 2015-07-31  13995       1498     1      1
0
4      5          5 2015-07-31   4822        559     1      1
0

   SchoolHoliday StoreType Assortment  CompetitionDistance  \
0              1         c          a               1270.0
1              1         a          a                570.0
2              1         a          a              14130.0
3              1         c          c                620.0
4              1         a          a              29910.0

   CompetitionOpenSinceMonth  CompetitionOpenSinceYear  Promo2  \
0                        9.0                    2008.0       0
1                       11.0                    2007.0       1
2                       12.0                    2006.0       1
3                        9.0                    2009.0       0
4                        4.0                    2015.0       0

   Promo2SinceWeek  Promo2SinceYear    PromoInterval
0              NaN              NaN              NaN
1             13.0           2010.0  Jan,Apr,Jul,Oct
2             14.0           2011.0  Jan,Apr,Jul,Oct
3              NaN              NaN              NaN
4              NaN              NaN              NaN
```

T

```
df['Date'] = pd.to_datetime(df['Date'])

df['Year'] = df['Date'].dt.year
```

```
df['Month'] = df['Date'].dt.month
df['Day'] = df['Date'].dt.day
df['DayOfWeek'] = df['Date'].dt.dayofweek
df['WeekOfYear'] = df['Date'].dt.isocalendar().week.astype(int)

df['StateHoliday'] = df['StateHoliday'].map({'0': 0, 0: 0, 'a': 1,
'b': 1, 'c': 1})
df['SchoolHoliday'] = df['SchoolHoliday'].astype(int)

df = df.sort_values(['Store', 'Date'])
df['SalesMovingAverage7'] = df.groupby('Store')
['Sales'].transform(lambda x: x.rolling(window=7).mean())
df['SalesMovingAverage30'] = df.groupby('Store')
['Sales'].transform(lambda x: x.rolling(window=30).mean())

df['SalesMovingAverage7'] = df['SalesMovingAverage7'].fillna(0)
df['SalesMovingAverage30'] = df['SalesMovingAverage30'].fillna(0)

df.head()
```

```
         Store  DayOfWeek       Date  Sales  Customers  Open  Promo  \
1016095      1          1 2013-01-01      0          0     0      0
1014980      1          2 2013-01-02   5530        668     1      0
1013865      1          3 2013-01-03   4327        578     1      0
1012750      1          4 2013-01-04   4486        619     1      0
1011635      1          5 2013-01-05   4997        635     1      0

         StateHoliday  SchoolHoliday StoreType  ... Promo2
Promo2SinceWeek  \
1016095             1              1         c  ...      0
NaN
1014980             0              1         c  ...      0
NaN
1013865             0              1         c  ...      0
NaN
1012750             0              1         c  ...      0
NaN
1011635             0              1         c  ...      0
NaN

         Promo2SinceYear  PromoInterval  Year  Month  Day
WeekOfYear  \
1016095              NaN            NaN  2013      1    1           1

1014980              NaN            NaN  2013      1    2           1

1013865              NaN            NaN  2013      1    3           1

1012750              NaN            NaN  2013      1    4           1
```

```
1011635                 NaN               NaN  2013      1    5           1


         SalesMovingAverage7  SalesMovingAverage30
1016095                  0.0                   0.0
1014980                  0.0                   0.0
1013865                  0.0                   0.0
1012750                  0.0                   0.0
1011635                  0.0                   0.0

[5 rows x 24 columns]
```

```
df = df.sort_values(['Store', 'Date'])

df['Sales_Lag1'] = df.groupby('Store')['Sales'].shift(1)

df['Sales_Mean7'] = df.groupby('Store')['Sales'].transform(lambda x:
x.rolling(window=7).mean())

df['Sales_Std7'] = df.groupby('Store')['Sales'].transform(lambda x:
x.rolling(window=7).std())

df['Sales_Lag1'] = df['Sales_Lag1'].fillna(0)
df['Sales_Mean7'] = df['Sales_Mean7'].fillna(0)
df['Sales_Std7'] = df['Sales_Std7'].fillna(0)

df[['Store', 'Date', 'Sales', 'Sales_Lag1', 'Sales_Mean7',
'Sales_Std7']].head(10)
```

```
         Store       Date  Sales  Sales_Lag1  Sales_Mean7   Sales_Std7
1016095      1 2013-01-01      0         0.0     0.000000     0.000000
1014980      1 2013-01-02   5530         0.0     0.000000     0.000000
1013865      1 2013-01-03   4327      5530.0     0.000000     0.000000
1012750      1 2013-01-04   4486      4327.0     0.000000     0.000000
1011635      1 2013-01-05   4997      4486.0     0.000000     0.000000
1010520      1 2013-01-06      0      4997.0     0.000000     0.000000
1009405      1 2013-01-07   7176         0.0  3788.000000  2752.283961
1008290      1 2013-01-08   5580      7176.0  4585.142857  2231.018633
1007175      1 2013-01-09   5471      5580.0  4576.714286  2226.961885
1006060      1 2013-01-10   4892      5471.0  4657.428571  2226.641706
```

```
import numpy as np

df = df.sort_values(['Store', 'Date'])

df['Sales_Lag1'] = df.groupby('Store')['Sales'].shift(1)
df['Sales_Mean7'] = df.groupby('Store')['Sales'].transform(lambda x:
x.rolling(window=7).mean())
df['Sales_Std7'] = df.groupby('Store')['Sales'].transform(lambda x:
x.rolling(window=7).std())
```

```
df['sin_month'] = np.sin(2 * np.pi * df['Month'] / 12)
df['cos_month'] = np.cos(2 * np.pi * df['Month'] / 12)

df['IsHolidayNextDay'] = df['StateHoliday'].shift(-
1).fillna(0).astype(int)
df['IsHolidayYesterday'] =
df['StateHoliday'].shift(1).fillna(0).astype(int)

df['Sales_Lag1'] = df['Sales_Lag1'].fillna(0)
df['Sales_Mean7'] = df['Sales_Mean7'].fillna(0)
df['Sales_Std7'] = df['Sales_Std7'].fillna(0)

df[['Store', 'Date', 'Sales', 'Sales_Lag1', 'Sales_Mean7',
'Sales_Std7', 'sin_month', 'cos_month', 'IsHolidayNextDay']].head(10)
```

```
         Store       Date  Sales  Sales_Lag1  Sales_Mean7   Sales_Std7
\
1016095      1 2013-01-01      0         0.0     0.000000     0.000000

1014980      1 2013-01-02   5530         0.0     0.000000     0.000000

1013865      1 2013-01-03   4327      5530.0     0.000000     0.000000

1012750      1 2013-01-04   4486      4327.0     0.000000     0.000000

1011635      1 2013-01-05   4997      4486.0     0.000000     0.000000

1010520      1 2013-01-06      0      4997.0     0.000000     0.000000

1009405      1 2013-01-07   7176         0.0  3788.000000  2752.283961

1008290      1 2013-01-08   5580      7176.0  4585.142857  2231.018633

1007175      1 2013-01-09   5471      5580.0  4576.714286  2226.961885

1006060      1 2013-01-10   4892      5471.0  4657.428571  2226.641706


         sin_month  cos_month  IsHolidayNextDay
1016095        0.5   0.866025                 0
1014980        0.5   0.866025                 0
1013865        0.5   0.866025                 0
1012750        0.5   0.866025                 0
1011635        0.5   0.866025                 0
1010520        0.5   0.866025                 0
1009405        0.5   0.866025                 0
1008290        0.5   0.866025                 0
1007175        0.5   0.866025                 0
1006060        0.5   0.866025                 0
```

```
from sklearn.preprocessing import MinMaxScaler

df['CompetitionDistance'] =
df['CompetitionDistance'].fillna(df['CompetitionDistance'].max())
df = df.fillna(0)

columns_to_scale = ['Sales', 'CompetitionDistance', 'Sales_Lag1',
'Sales_Mean7', 'Sales_Std7']

scaler = MinMaxScaler()
df[columns_to_scale] = scaler.fit_transform(df[columns_to_scale])

df[columns_to_scale].head()
```

```
            Sales  CompetitionDistance  Sales_Lag1  Sales_Mean7
Sales_Std7
1016095  0.000000             0.016482    0.000000          0.0
0.0
1014980  0.133089             0.016482    0.000000          0.0
0.0
1013865  0.104137             0.016482    0.133089          0.0
0.0
1012750  0.107964             0.016482    0.104137          0.0
0.0
1011635  0.120262             0.016482    0.107964          0.0
0.0
```

```
df = df.sort_values('Date')

train_size = int(len(df) * 0.7)
val_size = int(len(df) * 0.15)

train_df = df.iloc[:train_size]
val_df = df.iloc[train_size : train_size + val_size]
test_df = df.iloc[train_size + val_size:]

print(f"Train dates: {train_df['Date'].min()} to
{train_df['Date'].max()}")
print(f"Val dates: {val_df['Date'].min()} to {val_df['Date'].max()}")
print(f"Test dates: {test_df['Date'].min()} to
{test_df['Date'].max()}")
```

```
Train dates: 2013-01-01 00:00:00 to 2014-10-19 00:00:00
Val dates: 2014-10-19 00:00:00 to 2015-03-17 00:00:00
Test dates: 2015-03-17 00:00:00 to 2015-07-31 00:00:00
```

```
import os
import gc
import math
from datetime import datetime, timedelta
from tqdm import tqdm
```

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
import tensorflow as tf
from tensorflow.keras import layers, models, callbacks, backend as K
import optuna
import shap
import matplotlib.pyplot as plt
```

1. Model Hyperparameters & Constants

SEQ_LEN = 56 and HORIZON = 7: The model will look at exactly 56 days (8 weeks) of historical data to predict the next 7 days of sales.

N_ENSEMBLE = 3 and MC_DROPOUT_FORWARD_PASSES = 50: Instead of training one model, it trains an ensemble of 3 distinct models. During prediction, it will run the data through the network 50 times with "dropout" turned on to generate a different possible outcomes (Monte Carlo Dropout).

QUANTILES = [0.1, 0.5, 0.9]: The model is configured to output the 10th percentile (pessimistic), 50th percentile (median/expected), and 90th percentile (optimistic) sales forecasts.

```
SEQ_LEN = 56
HORIZON = 7
BATCH_SIZE = 256
EPOCHS = 30
PATIENCE = 5
N_ENSEMBLE = 3
MC_DROPOUT_FORWARD_PASSES = 50
QUANTILES = [0.1, 0.5, 0.9]
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
tf.random.set_seed(RANDOM_SEED)
```

2. Data Loading & Preprocessing Functions

read_data: loads the train.csv, test.csv, and store.csv files into Pandas DataFrames.

preprocess_base:

Merges the store-specific characteristics (like assortment type or competition distance) into the daily sales records.

- It addresses missing data by filling any missing CompetitionDistance with the maximum known distance, and fills all other NaNs with 0.

add_time_features: Extracts standard calendar features (Day, Month, Year, Week) from the Date column.

expand_promo_interval: Checks if the current row's month matches any of the store's designated promo months and creates a binary IsPromoIntervalMonth flag.

```
def read_data(base_folder="dataset"):
    train = pd.read_csv(f'{base_path}train.csv', parse_dates=['Date'],
low_memory=False)
    store = pd.read_csv(f'{base_path}store.csv')
    test = pd.read_csv(f'{base_path}test.csv')
    # ensure Date dtypes
    return train, test, store

def preprocess_base(train, test, store):
    train = train.merge(store, how="left", on="Store")
    test = test.merge(store, how="left", on="Store")
    def promo_months_to_list(x):
        if pd.isna(x): return []
        return [m.strip() for m in x.split(',')]
    store['PromoInterval_list'] =
store['PromoInterval'].apply(promo_months_to_list)

    max_comp = train['CompetitionDistance'].max()
    train['CompetitionDistance'] =
train['CompetitionDistance'].fillna(max_comp)
    test['CompetitionDistance'] =
test['CompetitionDistance'].fillna(max_comp)
    train = train.fillna(0)
    test = test.fillna(0)
    return train, test

def add_time_features(df):
    df['Day'] = df['Date'].dt.day
    df['Month'] = df['Date'].dt.month
    df['Year'] = df['Date'].dt.year
    df['WeekOfYear'] = df['Date'].dt.isocalendar().week.astype(int)
    # cyclical month
    df['sin_month'] = np.sin(2 * np.pi * df['Month'] / 12)
    df['cos_month'] = np.cos(2 * np.pi * df['Month'] / 12)
    return df

def expand_promo_interval(df, store_df):

    promo_map = store_df.set_index('Store')['PromoInterval'].to_dict()
    def is_in_promo_interval(row):
        pi = promo_map.get(row['Store'], "")
        if not pi or pi != pi: return 0
        months = [m.strip() for m in pi.split(',')]
        monname = row['Date'].strftime('%b')
        return 1 if monname in months else 0
    df['IsPromoIntervalMonth'] = df.apply(is_in_promo_interval,
axis=1)
    return df
```

3. Feature Engineering (Lags and Rolling Windows)

- sort the dataframe by Store and Date
- Sales_Lag1: Creates a column representing the exact sales from the previous day (shift(1)).
- Sales_Mean7, Sales_Std7, Sales_Mean28: Calculates the moving average and volatility (standard deviation) over the last 7 days and 28 days.
- CompetitionOpenMonths: Calculates exactly how many months a competing store has been open by subtracting the CompetitionOpenSince date from the current row's Year and Month.

```
def add_lag_features(df):
    df = df.sort_values(['Store', 'Date']).copy()

    df['Sales_Lag1'] = df.groupby('Store')['Sales'].shift(1).fillna(0)
    df['Sales_Mean7'] = df.groupby('Store')['Sales'].transform(lambda
x: x.rolling(7, min_periods=1).mean()).fillna(0)
    df['Sales_Std7'] = df.groupby('Store')['Sales'].transform(lambda
x: x.rolling(7, min_periods=1).std()).fillna(0)

    df['Sales_Mean28'] = df.groupby('Store')['Sales'].transform(lambda
x: x.rolling(28, min_periods=1).mean()).fillna(0)

    df['CompetitionOpenSinceYear'] =
df['CompetitionOpenSinceYear'].fillna(0)
    df['CompetitionOpenMonths'] = (df['Year'] -
df['CompetitionOpenSinceYear']) * 12 +
df['CompetitionOpenSinceMonth'].fillna(0)
    df['CompetitionOpenMonths'] =
df['CompetitionOpenMonths'].clip(lower=0)
    return df
```

create_sequences_multi_store

The model will look at the data inside this frame, and try to predict the next 7 days (horizon). Once that sequence is recorded, the frame slides forward by exactly one day, and the process repeats.

- The code uses groupby('Store') to isolate each store's timeline.
- if a store is brand new and doesn't even have 63 days of data, we skip it entirely so it doesn't crash the sliding window.

- The for start in range(...) loop is the actual sliding window. It carves out X (the 56 days of features) and y (the 7 days of target sales).
- If the store is closed the loop skips creating that sequence because predicting seven zeros isn't useful for training.

```python
def create_sequences_multi_store(df, features, target_col='Sales', 
seq_len=SEQ_LEN, horizon=HORIZON,
                                stores=None, min_open_days=1):

    X_list = []
    y_list = []
    idx_info = []
    if stores is None:
        stores = df['Store'].unique()

    grouped = df.sort_values(['Store', 'Date']).groupby('Store')
    for s in tqdm(stores, desc="create_sequences"):
        g = grouped.get_group(s).reset_index(drop=True)

        if len(g) < seq_len + horizon:
            continue

        feat_arr = g[features].values
        sales_arr = g[target_col].values
        open_arr = g['Open'].values if 'Open' in g.columns else 
np.ones_like(sales_arr)
        for start in range(0, len(g) - seq_len - horizon + 1):
            end = start + seq_len
            target_end = end + horizon

            if open_arr[end:target_end].sum() < min_open_days:
                continue
            X_list.append(feat_arr[start:end])

            y_list.append(sales_arr[end:target_end])
            idx_info.append((s, g.loc[end, 'Date']))  # store, 
prediction start
    X = np.array(X_list, dtype=np.float32)
    y = np.array(y_list, dtype=np.float32)
    return X, y, idx_info
```

2. scaleing

Neural networks perform much better when all input features are on a similar scale (usually between 0 and 1).

Flattening (reshape(-1, n_features)): It temporarily collapses the 3D data down to 2D by ignoring the concept of "time sequences." It just stacks every single day on top of each other.

Fitting: The scaler looks at this massive 2D list to find the global minimum and maximum for each feature (e.g., the absolute highest temperature ever recorded across all sequences).

Restoring: Once the scaling math is applied, it reshapes the data back into its original 3D sequence format (n_samples, seq_len, n_features).

```python
def fit_feature_scaler(X_train, feature_names):

    n_features = X_train.shape[2]
    flat = X_train.reshape(-1, n_features)
    scaler = MinMaxScaler()
    scaler.fit(flat)
    return scaler

def scale_X(X, scaler):
    n_samples, seq_len, n_features = X.shape
    flat = X.reshape(-1, n_features)
    flat_scaled = scaler.transform(flat)
    return flat_scaled.reshape(n_samples, seq_len, n_features)

def scale_y(y, scaler_y=None):
    if scaler_y is None:
        flat = y.reshape(-1, 1)
        scaler_y = MinMaxScaler()
        scaler_y.fit(flat)
    y_scaled = scaler_y.transform(y.reshape(-1,1)).reshape(y.shape)
    return y_scaled, scaler_y
```

The Deep Learning Architectures

Why use both LSTM and TCN?

- they are used to build an Ensemble.the training loop trains an LSTM, then it trains a TCN, then another LSTM. Later, it averages their predictions together. We do this because LSTMs and TCNs think about time very differently.

LSTM (Long Short-Term Memory):

LSTMs read data sequentially, day by day, passing a "hidden state" forward. They are exceptionally good at remembering long-term trends and general directions.

TCN (Temporal Convolutional Network):

TCNs look at the data using expanding mathematical filters (convolutions). They are good at recognizing sudden, local, repeating patterns (like the sharp spike every Saturday).

By combining them in an ensemble, the LSTM handles the smooth long-term memory, and the TCN handles the sharp weekly seasonality. They cover each other's blind spots.

Model 1: The LSTM (build_lstm_model)

- Accepts a 3D window of 56 historical days. The Masking layer ignores zero-padded days so they don't skew the results.
- A first layer of 128 units processes the full sequence, followed by a 64-unit "bottleneck" layer that compresses the history into a single vector of the store's current state.
- By setting training=True in the dropout layers, the model remains stochastic during prediction. Running 50 passes allows us to calculate forecast uncertainty.

- ## Model 2: The TCN (build_tcn_model & tcn_block)
- padding='causal' forces the TCN to only look at the past.
- Dilation Rate is the magic of TCNs. The first layer looks at days right next to each other (1 day apart). The next layer skips a day (2 days apart). The next looks 4 days apart, then 8. This allows the network to cover the entire 56-day history very rapidly without needing millions of parameters.
- Residual Connections: Deep networks suffer from the "vanishing gradient" problem.

The Output Layer

- Instead of a 7-neuron output for a 7-day forecast, this model uses 21 neurons (7 days × 3 quantiles).
- For every day, the model predicts the 10th (pessimistic), 50th (median), and 90th (optimistic) percentiles. model is no longer just guessing a single number for tomorrow's sales; it is giving you a range of possibilities.

```
def build_lstm_model(seq_len, n_features, horizon,
n_quantiles=len(QUANTILES), units=128, dropout=0.2, lr=1e-3):
    """
    LSTM with dropout (MC-dropout compatible)
    Output: horizon * n_quantiles values
    """
    inp = layers.Input(shape=(seq_len, n_features), name='input')
    x = layers.Masking(mask_value=0.0)(inp)
    x = layers.LSTM(units, return_sequences=True)(x)
    x = layers.Dropout(dropout)(x, training=True)   # training=True
for MC-dropout sampling at inference
    x = layers.LSTM(units//2)(x)
    x = layers.Dropout(dropout)(x, training=True)
    out = layers.Dense(horizon * n_quantiles)(x)
    model = models.Model(inp, out)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
loss='mse')
    return model
```

```
def tcn_block(x, filters, kernel_size, dilation_rate, dropout):
    prev = x
    x = layers.Conv1D(filters, kernel_size, padding='causal',
dilation_rate=dilation_rate)(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.SpatialDropout1D(dropout)(x)
    x = layers.Conv1D(filters, kernel_size, padding='causal',
dilation_rate=dilation_rate)(x)
    x = layers.BatchNormalization()(x)
    # residual
    if prev.shape[-1] != filters:
        prev = layers.Conv1D(filters, 1, padding='same')(prev)
    x = layers.add([prev, x])
    x = layers.Activation('relu')(x)
    return x

def build_tcn_model(seq_len, n_features, horizon,
n_quantiles=len(QUANTILES), filters=64, kernel_size=3, dropout=0.2,
lr=1e-3):
    inp = layers.Input(shape=(seq_len, n_features))
    x = layers.Conv1D(filters, 1, padding='causal')(inp)
    for d in [1,2,4,8]:
        x = tcn_block(x, filters, kernel_size, dilation_rate=d,
dropout=dropout)
    x = layers.GlobalAveragePooling1D()(x)
    x = layers.Dense(128, activation='relu')(x)
    x = layers.Dropout(dropout)(x, training=True)
    out = layers.Dense(horizon * n_quantiles)(x)
    model = models.Model(inp, out)

model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
loss='mse')
    return model
```

because our project requires predicting a range (10th, 50th, and 90th), we use Quantile Loss.

```
def quantile_loss(q, y_true, y_pred):
    e = y_true - y_pred
    return K.mean(K.maximum(q*e, (q-1)*e), axis=-1)

def multi_quantile_loss_wrapper(quantiles):
    def loss(y_true, y_pred):
        horizon = y_true.shape[1]
        nq = len(quantiles)
        y_pred = K.reshape(y_pred, (-1, horizon, nq))
        losses = []
        for i, q in enumerate(quantiles):
            losses.append(quantile_loss(q, y_true, y_pred[:,:,i]))
        return K.mean(tf.stack(losses, axis=0))
    return loss
```

`ensemble_predict`: It gathers all those 50 passes from all three models in your ensemble and stacks them all together.

```
def train_model(model, X_train, y_train, X_val, y_val, model_path,
epochs=EPOCHS):
    os.makedirs(os.path.dirname(model_path), exist_ok=True)
    es = callbacks.EarlyStopping(monitor='val_loss',
patience=PATIENCE, restore_best_weights=True)
    mc = callbacks.ModelCheckpoint(model_path, monitor='val_loss',
save_best_only=True, save_weights_only=True)
    history = model.fit(
        X_train, y_train,
        validation_data=(X_val, y_val),
        batch_size=BATCH_SIZE,
        epochs=epochs,
        callbacks=[es, mc],
        verbose=2
    )
    return model, history

def predict_with_mc_dropout(model, X,
forward_passes=MC_DROPOUT_FORWARD_PASSES):

    preds = []
    for i in range(forward_passes):
        p = model(X, training=True).numpy()  # shape (n, horizon*nq)
        preds.append(p)
    preds = np.stack(preds, axis=0)
    return preds

def ensemble_predict(models_list, X,
forward_passes=MC_DROPOUT_FORWARD_PASSES):
    all_preds = []
    for m in models_list:
        mc = predict_with_mc_dropout(m, X,
forward_passes=forward_passes)
        all_preds.append(mc)
    cat = np.concatenate(all_preds, axis=0)
    return cat
```

- `inverse_transform_y` reshapes the data to make sure every day and every quantile is correctly converted back to its original scale.
- `rmse`: Root Mean Squared Error

```
def inverse_transform_y(y_scaled, scaler_y):
    # y_scaled shaped (n, horizon) or flattened
    flat = y_scaled.reshape(-1,1)
    inv = scaler_y.inverse_transform(flat).reshape(y_scaled.shape)
    return inv

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

Data load & features

Loading and Base Processing

The code reads the raw CSV files for the training data, test data, and store metadata, and immediately converts the 'Date' columns into proper datetime objects. It merges the store details into the main datasets and extracts time features like the day, month, and week.

Combining Train and Test for Lag Features

Temporarily stacks the test data at the bottom of the training data to calculate lag features in one continuous timeline, before splitting them again.

Label Encoding Categorical Features

The `LabelEncoder` maps text categories to integers (e.g., A=0, B=1). It fits the encoder on both the train and test sets simultaneously to ensure it recognizes every possible category.

4. The Feature Safety Net

It checks if any required feature is missing from the dataframe; if one is missing, it fills the column with zeros so the code doesn't crash during training.

5. Chronological Train/Val/Test Split

Finally sorts all the prepared training data by date and slices it into three sets: 70% for Train, 15% for Validation, and 15% for Test.

- the data is *not* randomly shuffled; keeping it chronological prevents the model from accidentally looking into the future to predict the past.

```

train_raw, test_raw, store = read_data("dataset")

train_raw['Date'] = pd.to_datetime(train_raw['Date'])
test_raw['Date'] = pd.to_datetime(test_raw['Date'])

train_raw, test_raw = preprocess_base(train_raw, test_raw, store)

train_raw = add_time_features(train_raw)
test_raw = add_time_features(test_raw)

train_raw = add_lag_features(train_raw)
```

```
combined = pd.concat([train_raw, test_raw.assign(Sales=0)],
sort=False).reset_index(drop=True)
combined = add_time_features(combined)
combined = add_lag_features(combined)

train = combined[combined['Date'] <= train_raw['Date'].max()].copy()
test = combined[combined['Date'] > train_raw['Date'].max()].copy()

# Label encode a few categorical features
categorical_cols = ['StateHoliday','StoreType','Assortment']
for c in categorical_cols:
    le = LabelEncoder()
    train[c] = train[c].astype(str)
    le.fit(train[c].unique().tolist() +
test[c].astype(str).unique().tolist())
    train[c] = le.transform(train[c].astype(str))
    test[c] = le.transform(test[c].astype(str))

FEATURES = [
    'Store','DayOfWeek','Promo','SchoolHoliday',
    'StoreType','Assortment',
    'CompetitionDistance','CompetitionOpenMonths',

'Promo2','Promo2SinceWeek','Promo2SinceYear','IsPromoIntervalMonth',
    'sin_month','cos_month',
    'Sales_Lag1','Sales_Mean7','Sales_Std7','Sales_Mean28'
]
for f in FEATURES:
    if f not in train.columns:
        train[f] = 0
        test[f] = 0

# Train/Val/Test split
train_dates_sorted = train.sort_values('Date')
n = len(train_dates_sorted)
train_size = int(n * 0.7)
val_size = int(n * 0.15)
train_df = train_dates_sorted.iloc[:train_size]
val_df = train_dates_sorted.iloc[train_size: train_size + val_size]
test_df = train_dates_sorted.iloc[train_size + val_size:]

print("Data prepared.")
```

```
Data prepared.
```

Generating 3D Sequences

It creates the 56-day historical input blocks (X) and their corresponding 7-day future target blocks (y) for the train, validation, and test sets.

Scaling the Input Features

This code calculates the minimum and maximum boundaries from the training data to prevent cheating, and then standardizes to 0-to-1 range.

Scaling the Target Variables

scales the target "Sales" column down to a 0-to-1 range using rules learned from the training set. - saves this (`y_scaler`) for "un-scaling" the model's predictions.

```
X_train, y_train, _ = create_sequences_multi_store(train_df, FEATURES,
seq_len=SEQ_LEN, horizon=HORIZON)
X_val, y_val, _ = create_sequences_multi_store(val_df, FEATURES,
seq_len=SEQ_LEN, horizon=HORIZON)
X_test, y_test, idx_test = create_sequences_multi_store(test_df,
FEATURES, seq_len=SEQ_LEN, horizon=HORIZON)

print("Shapes:", X_train.shape, X_val.shape, X_test.shape)

feat_scaler = fit_feature_scaler(X_train, FEATURES)
X_train_s = scale_X(X_train, feat_scaler)
X_val_s = scale_X(X_val, feat_scaler)
X_test_s = scale_X(X_test, feat_scaler)

y_train_s, y_scaler = scale_y(y_train)
y_val_s, _ = scale_y(y_val, y_scaler)
y_test_s, _ = scale_y(y_test, y_scaler)

n_features = X_train_s.shape[2]
print("Scaled. n_features=", n_features)

create_sequences: 100%|██████████| 1115/1115 [00:12<00:00, 87.25it/s]
create_sequences: 100%|██████████| 1115/1115 [00:02<00:00, 435.69it/s]
create_sequences: 100%|██████████| 1115/1115 [00:03<00:00, 364.37it/s]

Shapes: (641933, 56, 18) (83401, 56, 18) (83409, 56, 18)
Scaled. n_features= 18
```

Run this if you have models.keras and dont want to train fro scratch and skip next cell

```
import tensorflow as tf
import os

MODEL_DIR = "/kaggle/working/models_phase5"
models_list = []
N_ENSEMBLE = 3

#for i in range(N_ENSEMBLE):
#    model_path = os.path.join(MODEL_DIR, f"model_ensemble_{i}.keras")


#    m = tf.keras.models.load_model(model_path, compile=False)
#    models_list.append(m)
```

```
#print("loaded saved ensemble models without retraining!")
```

```
loaded saved ensemble models without retraining!
```

Ensemble Loop and Architecture Alternation

building an LSTM on even rounds and a TCN on odd rounds, to ensures the final ensemble has both long-term sequential memory and short-term pattern recognition.

Compiling with Uncertainty

each newly built model is compiled using your custom `multi_quantile_loss_wrapper`.

Full Model Saving and Memory Cleanup Once training finishes, the entire model—including its architecture, weights, and optimizer state—is saved in `.keras` format.

```
MODEL_DIR = "/kaggle/working/models_phase5"
os.makedirs(MODEL_DIR, exist_ok=True)
models_list = []

for i in range(N_ENSEMBLE):
    if i % 2 == 0:
        m = build_lstm_model(SEQ_LEN, n_features, HORIZON,
n_quantiles=len(QUANTILES), units=128, dropout=0.246, lr=0.003)
    else:
        m = build_tcn_model(SEQ_LEN, n_features, HORIZON,
n_quantiles=len(QUANTILES), filters=64, kernel_size=3, dropout=0.2,
lr=1e-3)

    # compile with quantile loss
    m.compile(optimizer=tf.keras.optimizers.Adam(1e-3),
loss=multi_quantile_loss_wrapper(QUANTILES))

    weights_path = os.path.join(MODEL_DIR,
f"model_ensemble_{i}.weights.h5")
    print(f"Training model {i} -> weights file {weights_path}")

    m, history = train_model(m, X_train_s, y_train_s, X_val_s,
y_val_s, weights_path, epochs=EPOCHS)

    full_model_path = os.path.join(MODEL_DIR,
f"model_ensemble_{i}.keras")
    m.save(full_model_path)
    print("Saved full model:", full_model_path)

    models_list.append(m)
    import gc; gc.collect()

print("Ensemble training complete. Models saved to:", MODEL_DIR)
print("Files:", os.listdir(MODEL_DIR)[:50])
```

Batched MC-dropout + ensemble aggregation

1. Loops through the dataset, jumping 1024 rows at a time.
2. Slices out the current chunk of 1024 sequences.
3. It asks the model to predict the chunk, but forces `training=True` so that Dropout stays active.
4. Saves this chunk's prediction.
5. Glues the 1024-row chunks back together to form the full dataset prediction for pass `t`.
6. Saves the completed pass.
7. Stacks all 10 passes into a 3D grid.

The Ensemble Aggregator

This function runs the above process for *every* model in ensemble.

Execution and Reshaping

executes the ensemble prediction on the scaled test data (`X_test_s`).

Condensing the 30 Guesses

We don't want to submit 30 different guesses per day we want to find the middle of the ensemble's uncertainty.Loops exactly 3 times (once for the 10th, 50th, and 90th). Grabs all 30 passes just for the current quantile.Looks at the 30 different guesses and takes the median .This collapses the 30 passes down to 1.

Un-scaling and Eval

This loops through the 3 quantiles and un-scales them back into Euros using the `y_scaler` we saved earlier.

- Calculates Root Mean Squared Error between the actual sales and the median predictions.
- calculate and print the R-squared score.

```
import os
import gc
import numpy as np
import pandas as pd
from tqdm.auto import tqdm
from sklearn.metrics import mean_squared_error, r2_score,
mean_absolute_error, mean_absolute_percentage_error


MC_DROPOUT_FORWARD_PASSES = 10
PRED_BATCH_SIZE = 1024
```

```python
def predict_with_mc_dropout_batched(model, X,
forward_passes=MC_DROPOUT_FORWARD_PASSES, batch_size=PRED_BATCH_SIZE):
    n = X.shape[0]
    preds_runs = []
    outdim = model.output_shape[-1]

    for t in tqdm(range(forward_passes), desc="MC Passes",
leave=False):
        batch_preds = []
        for i in range(0, n, batch_size):
            xb = X[i:i+batch_size]
            p = model(xb, training=True).numpy()
            batch_preds.append(p)
        batch_preds = np.concatenate(batch_preds, axis=0)
        preds_runs.append(batch_preds)

    preds_runs = np.stack(preds_runs, axis=0)
    return preds_runs

def ensemble_predict_batched(models_list, X,
forward_passes=MC_DROPOUT_FORWARD_PASSES, batch_size=PRED_BATCH_SIZE):
    all_preds = []

    for i, m in enumerate(tqdm(models_list, desc="Ensemble Models")):
        mc = predict_with_mc_dropout_batched(m, X,
forward_passes=forward_passes, batch_size=batch_size)
        all_preds.append(mc)
        gc.collect()

    cat = np.concatenate(all_preds, axis=0)
    return cat


print("Starting GPU Inference...")
mc_preds = ensemble_predict_batched(models_list, X_test_s,
forward_passes=MC_DROPOUT_FORWARD_PASSES, batch_size=PRED_BATCH_SIZE)
print("mc_preds shape:", mc_preds.shape)

T_total, n_samples, outdim = mc_preds.shape
mc_preds = mc_preds.reshape(T_total, n_samples, HORIZON,
len(QUANTILES))

print("Aggregating uncertainty quantiles...")
preds_by_q = []
for qi in range(len(QUANTILES)):
    arr = mc_preds[..., qi]
    med = np.median(arr, axis=0)
    preds_by_q.append(med)
```

```python
# Inverse transform to Euros
preds_q_inv = [inverse_transform_y(p, y_scaler) for p in preds_by_q]
y_test_inv =inverse_transform_y(y_test_s, y_scaler)

# EVAL

y_hat = preds_q_inv[1]  # The 50th percentile (Median expected sales)

print("\n--- FINAL TEST METRICS ---")
print(f"Test RMSE: {rmse(y_test_inv, y_hat):.2f}")
print(f"Test MAE:  {mean_absolute_error(y_test_inv.flatten(),
y_hat.flatten()):.2f}")
print(f"Test MAPE:
{mean_absolute_percentage_error(y_test_inv.flatten(),
y_hat.flatten()):.4f}")

try:
    print(f"Test R2:   {r2_score(y_test_inv.flatten(),
y_hat.flatten()):.4f}")
except:
    pass

# Evaluate Uncertainty Coverage (How often does actual sales fall
inside our 10% to 90% bounds?)
lower_bound = preds_q_inv[0]
upper_bound = preds_q_inv[2]
in_bounds = (y_test_inv >= lower_bound) & (y_test_inv <= upper_bound)
coverage = np.mean(in_bounds) * 100
print(f"Prediction Interval Coverage (10th to 90th): {coverage:.2f}%")
print("--------------------------\n")

# ---------------------------------------------------------
# 4. VECTORIZED CSV EXPORT (Instant)
# ---------------------------------------------------------
print("Formatting predictions to CSV...")

# Create Base DataFrame (Store IDs and Start Dates)
df_base = pd.DataFrame(idx_test, columns=['Store', 'Start_Date'])

# Create Predictions DataFrame (7 Columns for Day 0 to Day 6)
df_preds = pd.DataFrame(y_hat, columns=[f'Day_{h}' for h in
range(HORIZON)])

# Concatenate side-by-side
df_merged = pd.concat([df_base, df_preds], axis=1)

# "Melt" un-pivots the 7 day columns into rows instantly
df_melted = df_merged.melt(id_vars=['Store', 'Start_Date'],
var_name='DayOffset', value_name='Sales_pred')
```

```
# Vectorized Date Math (Takes 0.1 seconds instead of 10 minutes)
df_melted['DayOffset'] = df_melted['DayOffset'].str.replace('Day_',
'').astype(int)
df_melted['Date'] = df_melted['Start_Date'] +
pd.to_timedelta(df_melted['DayOffset'], unit='d')

# Clean up and Save
preds_df_long = df_melted[['Store', 'Date',
'Sales_pred']].sort_values(['Store', 'Date'])
save_path = os.path.join(MODEL_DIR, "ensemble_preds_long.csv")
preds_df_long.to_csv(save_path, index=False)

print(f"Successfully saved {len(preds_df_long)} predictions to:
{save_path}")
```

```
Starting GPU Inference...
```

{"model_id":"d418e3154ee2470ebfd5680a9feef23e","version_major":2,"version_minor":0}

{"model_id":"","version_major":2,"version_minor":0}

```
/usr/local/lib/python3.12/dist-packages/keras/src/models/
functional.py:241: UserWarning: The structure of `inputs` doesn't
match the expected structure.
Expected: ['input']
Received: inputs=Tensor(shape=(1024, 56, 18))
  warnings.warn(msg)
/usr/local/lib/python3.12/dist-packages/keras/src/models/functional.py
:241: UserWarning: The structure of `inputs` doesn't match the
expected structure.
Expected: ['input']
Received: inputs=Tensor(shape=(465, 56, 18))
  warnings.warn(msg)
```

{"model_id":"","version_major":2,"version_minor":0}

```
/usr/local/lib/python3.12/dist-packages/keras/src/models/
functional.py:241: UserWarning: The structure of `inputs` doesn't
match the expected structure.
Expected: ['input_layer']
Received: inputs=Tensor(shape=(1024, 56, 18))
  warnings.warn(msg)
/usr/local/lib/python3.12/dist-packages/keras/src/models/functional.py
:241: UserWarning: The structure of `inputs` doesn't match the
expected structure.
Expected: ['input_layer']
Received: inputs=Tensor(shape=(465, 56, 18))
  warnings.warn(msg)
```

{"model_id":"","version_major":2,"version_minor":0}

```
mc_preds shape: (30, 83409, 21)
Aggregating uncertainty quantiles...

--- FINAL TEST METRICS ---
Test RMSE: 1233.63
Test MAE:  732.05
Test MAPE: 762171564769673216.0000
Test R2:   0.8972
Prediction Interval Coverage (10th to 90th): 90.43%
--------------------------

Formatting predictions to CSV...
Successfully saved 583863 predictions to:
/kaggle/working/models_phase5/ensemble_preds_long.csv
```

SHAP: figures out exactly which features caused a prediction. Is designed for standard, flat data. Takes hours to run when handed the 3D, time-sequenced arrays your Deep Learning model uses. To bypass this:

- the first two lines compress your 56 days of history into a single average number for each feature and the 7-day target into a single average sales number.
- Instead of trying to force SHAP to look inside your complex, multi-headed LSTM/TCN ensemble, the code builds a =simple, standard neural network. It trains this simple network on the flattened data

Generating the Explanations

Even with a simple model, SHAP requires massive amounts of math. The code takes a tiny slice of your data (100 background samples and 50 validation samples) to speed things up.

1. passes these into a KernelExplainer.
2. output results:

```
X_train_flat = X_train_s.mean(axis=1)
X_val_flat = X_val_s.mean(axis=1)
from tensorflow.keras import Sequential
surrogate = Sequential([
    layers.Input(shape=(X_train_flat.shape[1],)),
    layers.Dense(128, activation='relu'),
    layers.Dense(64, activation='relu'),
    layers.Dense(1)
])
surrogate.compile(optimizer='adam', loss='mse')
y_train_flat = y_train.mean(axis=1)
surrogate.fit(X_train_flat, y_train_flat, epochs=5, batch_size=512,
verbose=1)

background = shap.sample(X_train_flat, 100) if X_train_flat.shape[0] >
```

```
100 else X_train_flat
val_sample = shap.sample(X_val_flat, 50) if X_val_flat.shape[0] > 50
else X_val_flat

explainer = shap.KernelExplainer(lambda x:
surrogate.predict(x).flatten(), background)
shap_values = explainer.shap_values(val_sample)
shap.summary_plot(shap_values, val_sample, feature_names=FEATURES,
show=False)
plt.tight_layout()
plt.savefig(os.path.join(MODEL_DIR, "shap_summary.png"))
plt.close()
print("Saved SHAP:", os.path.join(MODEL_DIR, "shap_summary.png"))
```

```
Epoch 1/5
1254/1254 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - loss: 20714860.0000
Epoch 2/5
1254/1254 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 2771778.5000
Epoch 3/5
1254/1254 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 1211479.5000
Epoch 4/5
1254/1254 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 1098734.6250
Epoch 5/5
1254/1254 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - loss: 1068429.0000
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 58ms/step
```

{"model_id":"14692ad97e2f42e5a83e616ce1820eff","version_major":2,"version_minor":0}

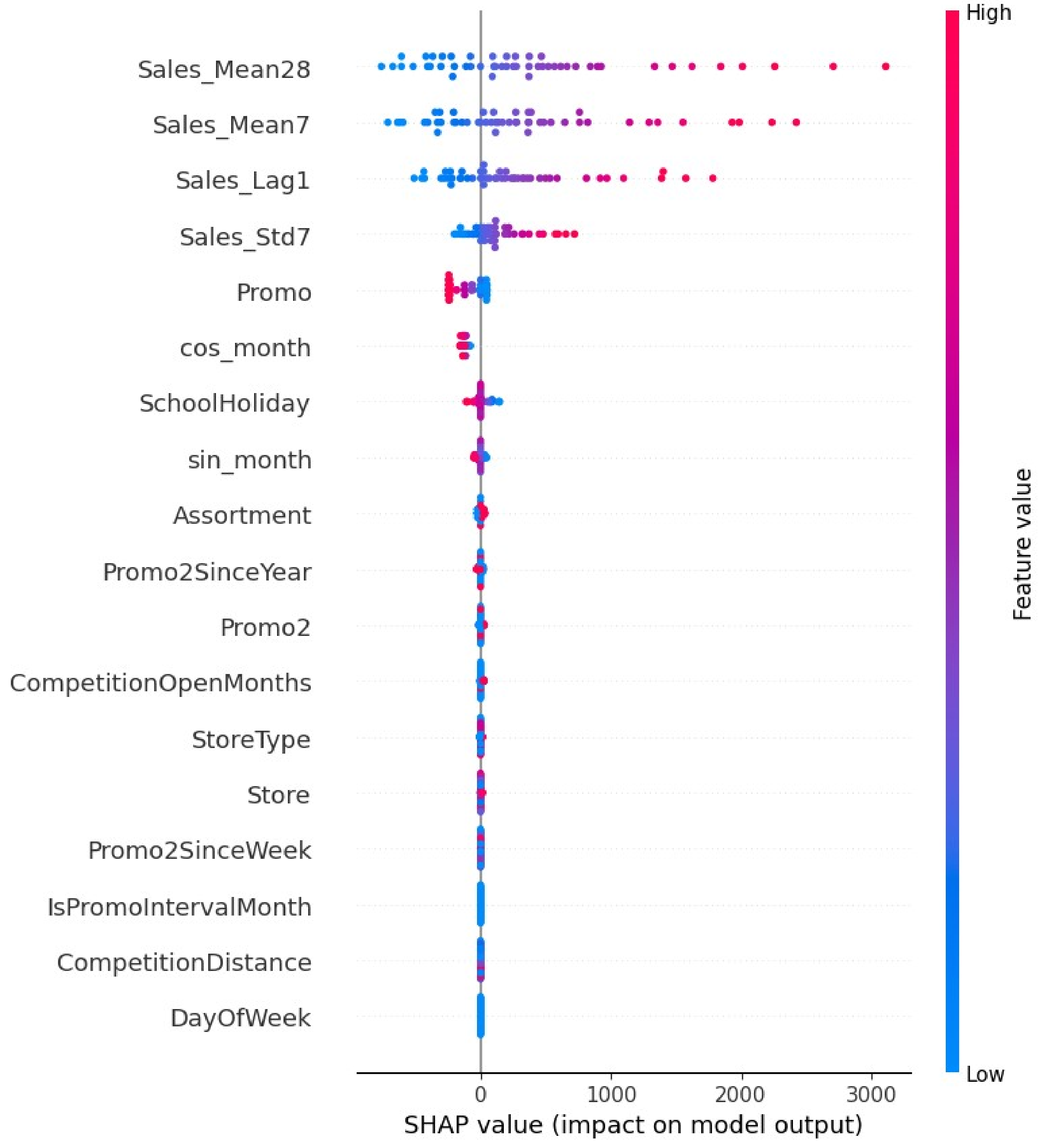
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 167ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
```

```
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step
6500/6500 ━━━━━━━━━━━━━━━━━━━━ 7s 1ms/step

/tmp/ipykernel_55/484151063.py:19: FutureWarning: The NumPy global RNG
was seeded by calling `np.random.seed`. In a future version this
function will no longer use the global RNG. Pass `rng` explicitly to
opt-in to the new behaviour and silence this warning.
  shap.summary_plot(shap_values, val_sample, feature_names=FEATURES,
show=False)

Saved SHAP: /kaggle/working/models_phase5/shap_summary.png
```

```python
import numpy as np, pandas as pd

sv = np.array(shap_values)

mean_abs = np.mean(np.abs(sv), axis=0)
fi = pd.Series(mean_abs, index=FEATURES).sort_values(ascending=False)
print(fi.head(20))
from IPython.display import Image
Image(filename='/kaggle/working/models_phase5/shap_summary.png')
```

```
Sales_Mean28              652.893720
Sales_Mean7               559.414780
Sales_Lag1                395.885431
Sales_Std7                169.381155
Promo                     141.703138
cos_month                 126.541633
SchoolHoliday              34.717644
sin_month                  21.886080
Assortment                 18.100117
Promo2SinceYear            11.516077
Promo2                      7.572928
CompetitionOpenMonths       6.602969
StoreType                   1.989237
Store                       1.234588
DayOfWeek                   0.000000
CompetitionDistance         0.000000
Promo2SinceWeek             0.000000
IsPromoIntervalMonth        0.000000
dtype: float64
```



Find best hyper parameters for our lstm,Tcm models

```
def optuna_objective(trial):
    units = trial.suggest_categorical('units', [64,128])
    dropout = trial.suggest_float('dropout', 0.1, 0.3)
    lr = trial.suggest_loguniform('lr', 1e-4, 1e-3)
    m = build_lstm_model(SEQ_LEN, n_features, HORIZON,
n_quantiles=len(QUANTILES), units=units, dropout=dropout, lr=lr)
    m.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=lr),
loss=multi_quantile_loss_wrapper(QUANTILES))
    history = m.fit(X_train_s, y_train_s, validation_data=(X_val_s,
y_val_s), batch_size=512, epochs=6, verbose=0)
    val_loss = min(history.history['val_loss'])
    K.clear_session()
    return val_loss

study = optuna.create_study(direction='minimize',
sampler=optuna.samplers.TPESampler(seed=RANDOM_SEED))
study.optimize(optuna_objective, n_trials=6)
print("Optuna best:", study.best_trial.params)
import joblib
joblib.dump(study, os.path.join(MODEL_DIR, "optuna_study.pkl"))
```

```
[I 2026-02-21 17:43:24,340] A new study created in memory with name:
no-name-230e3072-f6f3-4998-b55a-08ddeef9fbaf
/tmp/ipykernel_55/574459363.py:4: FutureWarning: suggest_loguniform
has been deprecated in v3.0.0. This feature will be removed in v6.0.0.
See https://github.com/optuna/optuna/releases/tag/v3.0.0. Use
suggest_float(..., log=True) instead.
  lr = trial.suggest_loguniform('lr', 1e-4, 1e-3)
[I 2026-02-21 17:45:58,804] Trial 0 finished with value:
0.010437797755002975 and parameters: {'units': 128, 'dropout':
0.24639878836228102, 'lr': 0.0003968793304443713}. Best is trial 0
with value: 0.010437797755002975.
[I 2026-02-21 17:47:51,472] Trial 1 finished with value:
0.010964587330818176 and parameters: {'units': 64, 'dropout':
0.1116167224336399, 'lr': 0.0007348118405270452}. Best is trial 0 with
value: 0.010437797755002975.
[I 2026-02-21 17:50:24,822] Trial 2 finished with value:
0.010731698013842106 and parameters: {'units': 128, 'dropout':
0.10411689885916049, 'lr': 0.0009330606024425672}. Best is trial 0
with value: 0.010437797755002975.
[I 2026-02-21 17:52:17,389] Trial 3 finished with value:
0.012731725350022316 and parameters: {'units': 64, 'dropout':
0.1363649934414201, 'lr': 0.00015254729458052615}. Best is trial 0
with value: 0.010437797755002975.
[I 2026-02-21 17:54:50,372] Trial 4 finished with value:
0.011255506426095963 and parameters: {'units': 128, 'dropout':
0.18638900372842315, 'lr': 0.00019553708662745247}. Best is trial 0
with value: 0.010437797755002975.
[I 2026-02-21 17:56:43,044] Trial 5 finished with value:
0.011262130923569202 and parameters: {'units': 64, 'dropout':
0.15842892970704364, 'lr': 0.0002324672848950435}. Best is trial 0
with value: 0.010437797755002975.

Optuna best: {'units': 128, 'dropout': 0.24639878836228102, 'lr':
0.0003968793304443713}

['/kaggle/working/models_phase5/optuna_study.pkl']
```

Run this if you want to download a zip of models

```python
import os
import zipfile

directory_to_zip = '/kaggle/working/models_phase5'
output_filename = 'rossman_project_backup.zip'

with zipfile.ZipFile(output_filename, 'w', zipfile.ZIP_DEFLATED) as
zip_ref:
    for root, dirs, files in os.walk(directory_to_zip):
        for file in files:
            file_path = os.path.join(root, file)
            zip_ref.write(file_path, os.path.relpath(file_path,
directory_to_zip))

print(f"Successfully created {output_filename}!")
```

```
Successfully created rossman_project_backup.zip!
```